Detecting Scareware by Mining Variable Length Instruction Sequences

Raja Khurram Shahzad
School of Computing
Blekinge Institute of Technology
SE-371 32 Karlskrona, Sweden
rks@bth.se

Niklas Lavesson
School of Computing
Blekinge Institute of Technology
SE-371 32 Karlskrona, Sweden
nla@bth.se

Abstract—**Scareware is a recent type of malicious software that may pose financial and privacy-related threats to novice users. Traditional countermeasures, such as anti-virus software, require regular updates and often lack the capability of detecting novel (unseen) instances. This paper presents a scareware detection method that is based on the application of machine learning algorithms to learn patterns in extracted variable length opcode sequences derived from instruction sequences of binary files. The patterns are then used to classify software as legitimate or scareware but they may also reveal interpretable behavior that is unique to either type of software. We have obtained a large number of real world scareware applications and designed a data set with 550 scareware instances and 250 benign instances. The experimental results show that several common data mining algorithms are able to generate accurate models from the data set. The Random Forest algorithm is shown to outperform the other algorithms in the experiment. Essentially, our study shows that, even though the differences between scareware and legitimate software are subtler than between, say, viruses and legitimate software, the same type of machine learning approach can be used in both of these dissimilar cases.**

Keywords- Scareware; Instruction Sequence; Classification

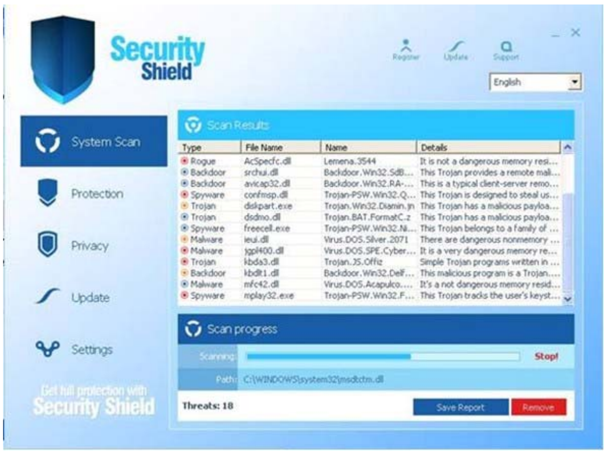
I. Introduction

This paper addresses the problem of detecting scareware, i.e., scam software with different forms of malicious payloads [1], and presents a machine learning-based approach for detection of this type of software. Many reports have been published in the media regarding malicious software (malware) such as viruses and worms. Such reports have arguably increased the awareness of novice computer users about basic security issues. Consequently, users are becoming more conscious about the security and privacy of their systems and data. Through media, friends, colleagues and security experts, users have been advised to install detection software such as anti-virus software or other protective measures. Success stories about the detection and mitigation of virus and worm threats have probably also played a part in enhancing the general security awareness. However, personal computers are becoming more and more attractive targets for cyber criminals due in part to the electronic financial transactions that are nowadays performed by private citizens from their personal computers. Notable examples of such transactions include Internet bank sessions and online credit card payments. This development has caused a shift in the focus of the malware authors from self-spreading malware such as worms, which are easily detectable due to their distribution techniques, to privacy invasive malware [2].

A recent addition to the family of privacy invasive malware is known as rogue software, rogueware, or scareware. In the remainder of this paper, the latter term will be used to denote this type of software. Scareware represents scam applications that usually masquerade as security applications such as anti-malware software or more specifically anti-virus software. In reality, however, scareware provides a minimum level of security or no security at all [3][4]. This type of software is especially crafted to include fake scanning dialogs, artificial progress bars and fake alerts [5]. Scareware may display fake lists of infected files and sometimes such lists are so unsophisticatedly generated that they include files that may not even exist on the computer or they may be incompatible with the operating system [6]. Fig 1. shows the fake scanning dialog of a particular scareware Rouge:W32/Winwebsec. The fake scanning processes and the fake results are of course used to scare users into believing that their system has been compromised and that it is infected with malicious content. The fake presentations are essentially carried out to convince the user that they need anti-virus software or some other form of protection. As a remedy for the fake situation, scareware offers a free download, which may also be bundled with other malware (such as trojans) or it may facilitate the installation of additional malware. Scareware may also trick users into paying registration fees in order to scan their system more thoroughly to remove (fake) warnings. Such an example is shown in Fig 2. which is the screenshot of payment screen displayed by Rpuge:W32/Winwebsec. The additional malware, which has been installed instead of protective software, remains on the targeted computer regardless of whether the registration fee is actually paid or not. Such additional malware is typically used to collect personal data of the user or to launch different forms of attacks.

A. Background

In 2003, Secure Works observed that spam advertisements for fake anti-virus software were being sent to users by the utilization of a vulnerability in the Microsoft Messenger Service [7]. Two years later, in 2005, Microsoft reported about the presence of scareware on web sites and web servers. Since then and arguably due to the overwhelming financial incentive to malware authors, the scareware has been increasing. The scareware distribution mechanism is different from other malware (such as viruses or worms). Scareware reaches the target machine by employment of social engineering, stealth techniques, or both of these approaches. User interaction is required when scareware is distributed through social engineering. For this purpose, advertisements are either sent via spam e-mail or posted on popular social networking websites. Scareware is misleadingly marketed as legitimate software and, with user interaction; it is downloaded and installed on personal computers. When it comes to stealth techniques, vulnerabilities in web browsers or other popular software are exploited in order to employ a so-called drive-by download mechanism. Essentially, scareware is downloaded and installed without any user interaction using such a mechanism. It has been reported that the monetary conversion rate of the fees obtained for fake scanning services can be as much as 1.36% which can result in a gross income of $21,000 - $35000 for a period of 44 days [2]. Panda Labs reported that an approximate overall gross income of 34 million per month is generated by scareware [8]. Late in 2009, Symantec Corporation reported about 43 million installation attempts from more than 240 distinct families of scareware [9]. Recently a Swedish newspaper Aftonbladet reported that according to U.S Department of Justice, 9, 60,000 users were victims of rouge which caused a loss of 460 million krona [10]. This alarming situation has received the attention of legitimate security software companies. CA Global Security Advisor, Secure Works and Microsoft published advisories about scareware, which describe the general functionality of scareware and tips for identifying this type of software [6]. To reduce the probability of being fooled by scareware, novice users are advised to install legitimate anti-malware software. However, the problem with such software is that users need to update it on regular basis as novel families of scareware are continuously appearing.



Type	File Name	Name	Details
Rogue	AcSpecfc.dll	Lemena.3544	It is not a dangerous memory resi...
Backdoor	srchui.dll	Backdoor.Win32.SdB...	This Trojan provides a remote mali...
Backdoor	avicap32.dll	Backdoor.Win32.RA-...	This is a typical client-server remo...
Spyware	confmsp.dll	Trojan-PSW.Win32.Q...	This Trojan is designed to steal us...
Trojan	diskpart.exe	Trojan.Win32.Diamin.jn	This Trojan has a malicious payloa...
Trojan	dsdmo.dll	Trojan.BAT.FormatC.z	This Trojan has a malicious payloa...
Spyware	freecell.exe	Trojan-PSW.Win32.Ni...	This Trojan belongs to a family of ...
Malware	ieui.dll	Virus.DOS.Silver.2071	There are dangerous nonmemory ...
Malware	jgpl400.dll	Virus.DOS.SPE.Cyber...	It is a very dangerous memory re...
Trojan	kbda3.dll	Trojan.JS.Offiz	Simple Trojan programs written in ...
Backdoor	kbdlt1.dll	Backdoor.Win32.Delf...	This malicious program is a Trojan....
Malware	mfc42.dll	Virus.DOS.Acapulco....	It's a not dangerous memory resid...
Spyware	mplay32.exe	Trojan-PSW.Win32.F...	This Trojan tracks the user's keyst...

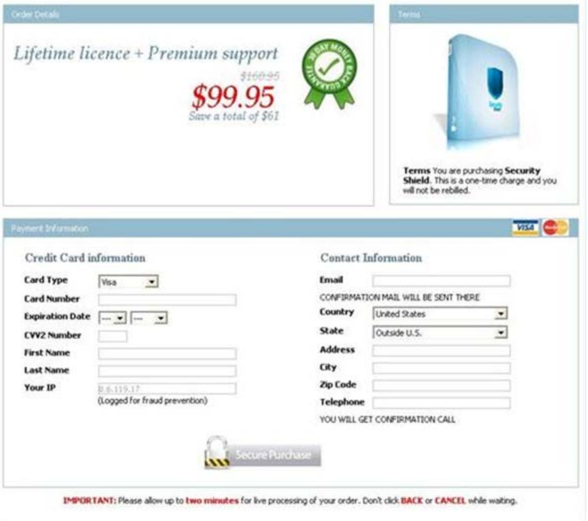
Path: C:\WINDOWS\system32\msdtctm.dll

Threats: 18

Figure 1. Scanning screenshot of Rouge:W32/Winwebsec [11]

B. *Traditional Countermeasures*

Current anti-malware programs primarily rely on either signature-based methods or heuristic-based methods for the detection of scareware; techniques that were originally developed for detecting computer viruses. The signature-based approach revolves around the use of signature databases that contain byte strings that are unique to different instances of software. If these databases are allowed to become more than a couple of weeks old, the detection rate will be significantly reduced due to the fact that the approach cannot detect recent scareware for which it lacks recorded signatures [6]. The second approach, which relies on heuristic-based methods, is based on more general rules that, e.g., may define malicious or benign behavior. For both methods, anti-malware vendors need to catch novel instances, analyze them, create new signatures or rules and then update their databases. It is safe to say that, between database updates, users may be exposed to novel scareware instances. Thus, it is important for users to have up-to-date and reliable protective measures.



Lifetime licence + Premium support

~~$160.95~~

$99.95

Save a total of $61

Terms You are purchasing **Security Shield**. This is a one-time charge and you will not be rebilled.

Credit Card information

Card Type Visa

Card Number

Expiration Date --- ---

CVV2 Number

First Name

Last Name

Your IP 8.6.119.17
(Logged for fraud prevention)

Contact Information

Email
CONFIRMATION MAIL WILL BE SENT THERE

Country United States

State Outside U.S.

Address

City

Zip Code

Telephone
YOU WILL GET CONFIRMATION CALL

IMPORTANT: Please allow up to **two minutes** for live processing of your order. Don't click **BACK** or **CANCEL** while waiting.

Figure 2. Payment screenshot from Rouge:W32/Winwebsec [11]

C. *Scope and Aim*

In this paper, we present results from an experimental evaluation of a new scareware detection method. The aim of this method is to extend the traditional heuristic detection approach by employing machine learning. The objectives of the study are to assess the performance of the proposed method, which can be described as an automated system for extracting typical behavior of scareware and benign software in the shape of variable length instruction sequences, and to analyze such fragments of behavior in order to improve upon the existing knowledge about scareware detection.

D. *Outline*

The remainder of this paper is organized as follows. Section 2 presents related work by first introducing necessary concepts and terminology in Section 2.1 and then reviewing related studies in Section 2.2. Section 3 then describes the employed methodology and the data preprocessing steps. Section 4 reviews the experimental procedure. The subsequent sections present the experimental results and the analysis. Finally, Section 5 concludes the paper and gives some pointers to future work.

II. Related Work

A. *Concepts and Terminology*

To overcome the deficiency of traditional techniques concerning the detection of novel instances, the potential of various approaches, such as agent-based technologies and artificial neural networks have been investigated. Data mining (DM) and machine learning (ML) methods have been extensively investigated in the field of text classification and have showed promising results for many applications. As we shall see, it is possible to benefit from this area of research when addressing the scareware detection problem. However, the idea of using DM and ML methods for making the malware detection process automated and for extending the heuristic-based detection approach for traditional malware is not new; it originates from a study conducted in 2001 [12].

The process of ML-based malware classification essentially follows standard classification and can thus be divided into two sub stages: training and testing. During the training stage, classifiers are generated from training sets that feature some type of extracted malware and benign file information as well as the predetermined classification of each file and the predictive performance of the classifiers is then evaluated during the testing stage.

For malware classification, data sets have been prepared using various representations of files and by using different features that are either present in the files or obtained from any kind of meta analysis (for example, runtime generated digital footprints). Features that are commonly extracted from a binary file include: byte code *n*-grams, printable strings and instruction sequences. The *n*-gram is a sequence of *n* characters or *n* extracted words. Other features that are present in binary files and that may also be used include system calls (to application programming interfaces). The use of opcodes as an alternative form of representation has also been suggested [13]. An assembly instruction contains an operation code (opcode) and maybe one or more operands for performing the operation. Opcodes or sequences of opcodes may be represented using *n*-grams, which, in turn, can be viewed upon as words or terms if the learning problem is defined as a text categorization problem.

In text categorization, text files are commonly represented using the bag of words model, which is based on Salton's vector space model [14]. A vocabulary of words or terms is extracted from the so-called document set. For each term (*t*) in the vocabulary, its frequency (*f*) in a single document (*d*) and in the entire set (*D*) is calculated. A weight is assigned to each term, usually equal to its *f* in *d*; such weights are denoted term frequencies (*tf*). When the frequency (F) of each term is calculated in *D*, this is called Document Frequency (DF). The *tf* value of a term is further divided by the frequency of the most frequent term in the document, i.e., *max(tf)* to obtain a normalized Term Frequency (TF) within the range of [0-1] as shown in Equation 1. An extended version of TF-DF is TF *Inverse Document Frequency* (TF-IDF), which combines TF and DF as shown in Equation 2; where N is the number of documents in the entire data set and DF is number of *d* in which *t* appears.

$$\textit{Term Frequency} = \frac{\mathrm{tf}}{\max(\mathrm{tf})} \quad (1)$$

$$\textit{TF Inverse Document Frequency} = \mathrm{TFx}\log\left(\frac{\mathrm{N}}{\mathrm{DF}}\right) \quad (2)$$

The problem of *n*-gram-based malware classification in this context is perhaps different from the general text categorization case since a huge vocabulary or very large feature sets have to be produced. The size of the vocabulary creates two problems: most ML algorithms cannot directly process the vocabulary and a vast number of terms in the vocabulary do not provide any valuable information for classification. Therefore, it is necessary to obtain a subset of features by applying feature selection. The Categorical Proportional Difference (CPD) algorithm is a rather recent example of such an algorithm. In a number of text categorization experiments, CPD has outperformed other traditional feature selection algorithms such as: chi-square, information gain, mutual information, and odds ratio [15]. CPD represents a measure of the degree to which a word contributes in discriminating a specific class from other classes [15]. The possible outcome of CPD falls between [-1 – 1] where a CPD value close to -1 indicates that a word occurs in an equal number of instances in all classes and a value of 1 or in proximity to 1 indicates that a word occurs only in one class. *A* is the number of times word *w* and class *c* occur together and let *B* the number of times word *w* occurs without class *c*, then we may define CPD for a particular word *w* and class *c* as shown in Equation 3:

$$\mathrm{CPD}(\mathrm{w},\mathrm{c}) = \frac{\mathrm{A}-\mathrm{B}}{\mathrm{A}+\mathrm{B}} \quad (3)$$

The reduced feature set can now be converted, e.g., into the Attribute-Relation File Format (ARFF). ARFF files are structured ASCII text files that include a set of data instances, each described by a set of features [16]. ARFF files are used as input to the Waikato Environment for Knowledge Analysis (Weka) [16] before applying learning algorithms in order to build and analyze classifiers. Essentially, Weka is a suite of machine learning algorithms and analysis tools for solving or analyzing data mining problems. There are, of course, many alternatives to Weka but we argue that this workbench is particularly fitting for developing our approach since it is released as open source and may be tuned, extended, or changed in any way.

B. Related Directions of Research

Opcodes have already been used to build signature databases that can be searched to detect different variants of worms [17]. To avoid the problem of having to manually update the databases of the scanners, data mining algorithms were later used as part of a scientific study to build a generic scanner [18]. In this study, experiments were performed on two different data sets: the first data set contained the opcode of each instruction and the second data set contained the opcode as well as the first operand of each instruction. The frequency of appearance in the virus class and in the benign class was used as a basis for feature selection. Results showed that the first data set produced better results than the second. In another study, opcode *n*-grams of different sizes were constructed to detect novel malware. By addressing the class imbalance problem properly, an accuracy of 96% was achieved [19]. The idea of using variable length instruction sequences was conceived as part of an attempt to detect worms. Frequently occurring instruction sequences were analyzed using ensemble learners to classify novel instances of worms. In an attempt to detect a more recent type of malware, called spyware, hexadecimal *n*-grams were used to represent binary files [20]. The most common *n*-grams for each class together with overall high frequency *n*-grams were used as features for building the classifiers. The spyware detection rate was recorded to be 90.5%.

Hexadecimal *n*-grams have been used extensively as features in traditional malware classification problems. Experiments have been performed on viruses, worms and trojans. These types of malware are typically very distinct from the standard benign software program. Moreover, only a few studies have used only the opcode from the instruction as the feature of choice [17][18][19]. Today, very little is known about the appropriateness of using opcodes or instruction sequences as features when trying to detect the type of malware that is more similar to benign software in terms of behavior. In this paper, we investigate the concept of scareware which, to the best of our knowledge, has not been investigated in terms of how well it can be detected by mining instruction sequences.

III. Methodology

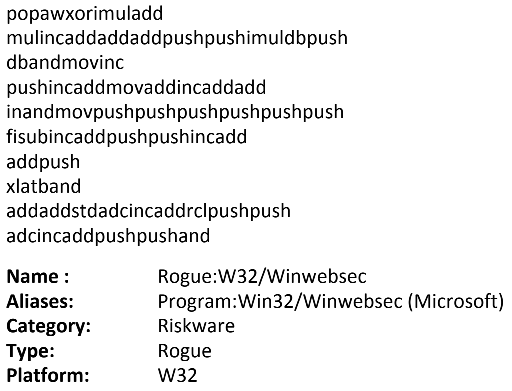
Generalizing the scareware detection method so it can detect novel instances can arguably be regarded as quite important for user protection. Another problem regarding the detection of scareware is that it may resemble legitimate software to such extents that it is difficult to detect differences. Recently, data mining classification algorithms have been heavily applied in order to automate and extend the heuristic-based methods for detection of traditional malware. It is therefore of interest to investigate how well such classification algorithms can detect scareware. Consequently, we present a static analysis method based on data mining, which extends the general heuristic detection approach. In this context, a dynamic analysis method is used to detect malware instances by investigating runtime footprints while static analysis is carried out on files without any runtime execution. Our data set contains Windows-based executable scareware and benign files and this choice was made since the Windows operating system is still the most commonly used operating system, especially for novice users, and it is often considered more vulnerable than, e.g., Unix-based operating systems. We have disassembled our initial file database into instruction sequences and then we extracted the opcodes from each instruction. The extracted opcodes were combined into ordered lists, instruction sequences (IS), to produce our vocabulary. Each word in vocabulary is of variable length. We have used TF-IDF and CPD for generating the final data sets.

A. File Sampling and Data Set Design

As the threat of scareware is relatively new compared to, say, viruses and worms, there is unfortunately no default or public data set available for researchers to build classification models from. Therefore, we have created a data set of 800 files out of which 550 are scareware (provided by Lavasoft from their scareware collection [4]). The remaining 250 files are benign and were downloaded from the website download.com [21]. This website claims that the software provided is spyware free. However, after downloading software from the website and scanning it with a commercial version of the F-Secure Client Security software [22], we discovered that some files were actually infected by so-called riskware. The infected instances were removed from the data set.

B. Extraction and Data Preparation

For the purpose of our experiment, we needed to convert our data set to a format that could be processed by learning algorithms. We decided to represent files by using extracted instruction sequences as features. The advantage of using IS as a primary feature is that IS represent program control flow blocks, which cannot be presented by binary or hexadecimal *n*-grams or printable strings. Moreover each IS in this study represents a function that can be located within the actual program for the purpose of deeper analysis, even though such a step is out of scope in the presented paper. We disassembled each program using the Netwide disassembler (Ndisasm) [23], which was configured to auto-synchronous mode to avoid misalignment between the data and code segments [23]. The generated output, from all the file segments, was stored in regular text files and each entry contains the memory address of the instruction field as well as the opcode and the operands.



```
popawxorimuladd
mulincaddaddaddpushpushimuldbpush
dbandmovinc
pushincaddmovaddincaddadd
inandmovpushpushpushpushpushpush
fisubincaddpushpushincadd
addpush
xlatband
addaddstdadcincaddrclpushpush
adcincaddpushpushand
```

Name :	Rogue:W32/Winwebsec
Aliases:	Program:Win32/Winwebsec (Microsoft)
Category:	Riskware
Type:	Rogue
Platform:	W32

Figure 3. Instruction Sequence extracted from Rouge:W32/Winwebsec

The disassembled files were further processed through a parser to obtain the instruction sequences (ordered lists of opcodes). During the extraction process, the end of an instruction sequence was determined by identifying a conditional or unconditional control transfer instruction or function boundary. It is worth noting that these identified control transfer instructions (such as: call, iret, jmp or jnz) were not included in the generated instruction sequences. In this way, we obtained variable length instruction sequences. Each row in output contains single IS. Fig. 3 shows the instruction sequences extracted from a scareware Rouge:W32/Winwebsec along-with some other related information of this particular scareware.

C. Feature Selection

Feature selection is performed to measure the correlation of each feature with its class (scareware or benign). It is also performed to estimate the role of that specific feature in classification task. The measures used for feature selection by any feature selection methods are not biased to any classification algorithm or class which helps us in comparing the performances of different classification algorithm.

Our disassembled files were in text format and each file can be read as text string so we decided to use the bag of words model, since it has been proven to be a suitable model for similar problems. We used the *StringToWordVector* filter in Weka to parse each string, extract the vocabulary and produce word vectors. For our experiment, each word represents a unique IS. We used TF-IDF for the weight calculation of each word. Our vocabulary features top 1,625 unique words. We decided to perform a secondary feature selection to eliminate features which will not contribute significant in classification task. We applied CPD to obtain reduced feature sets. As it is difficult to know beforehand the optimal number of features to remove, we decided to generate a number of data sets where each set was generated by keeping a different number of attributes. This process resulted in 19 reduced data sets for which 5-95% of the original features were kept.

IV. Experiment

The aim of the experiment is to evaluate classifier performance on the task of detecting scareware by learning from variable length instruction sequences and to assess the impact of feature selection using categorical proportional difference. Learning algorithms can be categorized according to their learning bias, that is, by the way their designs restrict the search space and dictate how this space is traversed. When categorizing the commonly used algorithms in this manner, a rather small number of algorithm families can be identified, e.g.: tree inducers, rule set inducers, neural networks, instance-based learners, and Bayesian learners. We have tried to select at least one representative algorithm from each family. As our study extends the heuristic based detection technique which uses rules set so we used families of algorithms which either uses rules or help in developing rule set. These families of algorithms are rules based and decision tree. Except these families we also used support vector machine, Bayesian theorem based algorithms and nearest neighbor concepts for classification. All the algorithms were used at their default configuration in WEKA.

A. Learning algorithms

1) ZeroR

ZeroR is a rule-based algorithm. ZeroR works as a random guesser, modeling a user that makes an uninformed decision about software by always predicting the majority class (the class to which most of the data instances belong) [16]. 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. This algorithm tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances they cover) and accuracy [24]. A data instance is classified as positive if a rule matches; otherwise it is classified as negative. JRip also features an optimization step in which redundant or bad rules are discarded.

3) J48

J48 is a decision-tree-based learning algorithm, which uses the concept of information entropy [25]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed. For partitioning, J48 uses the attribute with the highest information gain and stops if all instances of same class are present in the subset. In learning, they adopt top-down approach and traverse the tree to make a set of rules, which is used for classification.

4) Sequential Minimal Optimization (SMO)

SMO belongs to support vector machines. During classification, SMO finds the optimal hyper-plane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [26].

5) Naive Bayes

Naive Bayes (NB) 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. Therefore, presence (or absence) of a particular feature of a class is not dependent on the presence (or absence) of any other feature [27].

6) IBk

IB*k* is *k*-nearest neighbor classifier which uses Euclidean distance [28]. Predictions from the neighbors is obtained and weighted according to their distance from test instance. Majority class of closest *k* neighbors is assigned to new instance.

7) Random Forest

Random Forest (RF) is an ensemble learner. A specified number of decision trees are created and their mode is obtained for prediction predictions [29]. Being an ensemble learner, it has superiority of having combined decision which is not the case for other algorithms therefore it is expected to produce better accuracy than single decision tree.

B. Evaluation

We tested each learning algorithm by performing 10 fold cross-validation (CV) tests to ensure that the generated classifiers are not tested on the training data. Confusion matrices were generated by using the responses from classifiers. The following four measures defined the elements of the generated confusion matrices: True Positives (TP) represent the correctly identified scareware programs, False Positives (FP) represent legitimate software that has been classified as scareware, True Negatives (TN) represent correctly identified legitimate programs and False Negatives (FN) represent scareware programs that were incorrectly classified as legitimate software applications. We argue that the false negatives carry the highest cost from the users' perspective.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified scareware, as shown in Equation 4. False Negative Rate, which is the percentage of wrongly identified benign programs (see Equation 5), and Accuracy (ACC), the percentage of correctly identified programs (see Equation 6). The last evaluation measure used 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

TABLE I. LEARNING ALGORITHM AUC RESULTS FOR DIFFERENT LEVELS OF FEATURE SELECTION.

Data	SMO	Naive Bayes	IB*k*	Jrip	J48	R Forest
5%	0.717(0.119)	0.787(0.030)	0.778(0.030)	0.657(0.129)	0.500(0.000)	0.781(0.030)
10%	0.812(0.038)	0.829(0.044)	0.851(0.036)	0.809(0.040)	0.788(0.031)	0.857(0.036)
15%	0.860(0.045)	0.802(0.049)	0.896(0.037)	0.817(0.054)	0.869(0.043)	0.937(0.027)
20%	0.878(0.040)	0.814(0.045)	0.928(0.036)	0.868(0.047)	0.877(0.048)	0.958(0.021)
25%	0.864(0.042)	0.809(0.045)	0.924(0.034)	0.872(0.044)	0.874(0.054)	0.959(0.020)
30%	0.883(0.041)	0.804(0.045)	0.908(0.043)	0.885(0.038)	0.876(0.056)	0.960(0.021)
35%	0.885(0.041)	0.805(0.045)	0.927(0.036)	0.883(0.044)	0.880(0.053)	0.962(0.022)
40%	0.880(0.043)	0.815(0.045)	0.938(0.032)	0.885(0.041)	0.901(0.043)	0.964(0.018)
45%	0.892(0.041)	0.832(0.044)	0.930(0.034)	0.887(0.043)	0.904(0.047)	0.965(0.021)
50%	0.900(0.036)	0.855(0.043)	0.932(0.031)	0.893(0.043)	0.896(0.048)	0.966(0.019)
55%	0.906(0.035)	0.897(0.040)	0.928(0.033)	0.893(0.041)	0.896(0.051)	0.969(0.020)
60%	**0.910(0.033)**	**0.923(0.033)**	**0.935(0.029)**	**0.894(0.047)**	**0.900(0.044)**	**0.972(0.017)**
65%	0.910(0.031)	0.879(0.056)	0.938(0.028)	0.893(0.042)	0.894(0.046)	0.972(0.017)
70%	0.909(0.031)	0.710(0.056)	0.938(0.028)	0.901(0.038)	0.893(0.047)	0.970(0.019)
75%	0.909(0.031)	0.667(0.044)	0.938(0.028)	0.898(0.039)	0.893(0.047)	0.968(0.021)
80%	0.909(0.031)	0.657(0.043)	0.938(0.028)	0.901(0.039)	0.916(0.041)	0.970(0.021)
85%	0.909(0.031)	0.656(0.043)	0.938(0.029)	0.898(0.043)	0.915(0.037)	0.971(0.018)
90%	0.911(0.031)	0.658(0.042)	0.939(0.027)	0.896(0.039)	0.910(0.036)	0.970(0.019)
95%	0.915(0.031)	0.668(0.044)	0.938(0.028)	0.901(0.039)	0.906(0.038)	0.971(0.022)

instances in the data set [16]. Therefore it plots the DR on the x-axis in function of the False Positive Rate on the y-axis at different points. The higher AUC of an algorithm indicates that this algorithm is more robust and better in classification. In many situations, accuracy can also be a reasonable estimator of performance (the performance on completely new data). However, AUC has the benefits of being independent of class distribution and cost [30] unless the skewness of the class distribution is extreme.

$$\textit{Detection Rate} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad (4)$$

$$\textit{False Negative Rate} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} \qquad (5)$$

$$\textit{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (6)$$

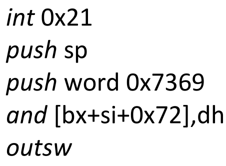
C. Results

The main experimental results, that is, the AUC of the seven included algorithms on the 19 data sets, are shown in Table I. In this table, ZeroR is used as a baseline (and can be regarded as a random guesser) with AUC of 0.500 (0.000) for all the datasets. All algorithms had performed better than base algorithm. Random Forest outperformed the other algorithms and its best performance (DR of 0.977, FNR of 0.023 and FPR of 0.197) was recorded at the 60% keep level (a data set with 974 features).

The Naive Bayes yielded an acceptable detection rate (0.857) but its FPR at different data sets was too high (i.e. up to 0.688) for practical use. Moreover, Naive Bayes also exhibited a high variance in performance related to the different data sets. Due to this behavior, it is not possible to consider this algorithm as reliable for the studied problem. On the same data set other algorithms also achieved either the highest AUC or near to the highest value with ignorable differences such as SMO achieved AUC 0.910 and highest AUC with 95% features was 0.915, IB*k* achieved 0.935 while the highest AUC was 0.938 for dataset of 65% features. JRip and J48 both achieved AUC of 0.894 and 0.900 respectively while their highest AUC was 0.901 and 0.916 for data sets with 70% and 80% features. Due to these ignorable minor differences in results, we considered that dataset with 60% features is a better option for our problem.

D. Analysis

We created 19 different datasets and each dataset was having 5% less features than its successor. Experimental results indicated that a step of 5% was not enough to create significant difference in the result. NB has been an exception to this, which showed a random trend with increased or decreased percentage of features. However if we look at the datasets created with the difference of 10% features then the difference in results is quite prominent. If the step is increased up to 20% difference of features, then a clear and understandable difference of results is present. If we review the overall performances on the various data sets, it is clear that the performance of most algorithms was quite high on the 60% feature selection level. It seems that number of kept features at this level is properly balanced with the number of instances from each class.



```
int 0x21
push sp
push word 0x7369
and [bx+si+0x72],dh
outsw
```

Figure 4. A disassembled function in one particular scareware instance

In order to understand the classification process and to find interesting features, we analyzed the models generated by JRip, J48, and SMO. Models created by other algorithms cannot be visualized, so it was not possible to perform their analysis. We found three kinds of features i.e. features present only in scareware, features present only in legitimate, features which were treated differently by different algorithms. Table II. shows some selected features with a high impact on the classification decision. Features 1 and 7 are used to indicate scareware by all three models. However, features 2, 5, 8, and 9 were considered as a scareware indicative feature by two algorithms but were ignored by the remaining algorithms. Features 2, 3, 6, and 10 seem to be considered as legitimate software indicative features by all algorithms. Finally, feature 4 is regarded as a legitimate indicator by JRip but as a scareware indicator by SMO.

In order to demonstrate the information provided by a single feature, we traced the features from Table II. to the disassembled binary files. One such example is provided in Fig. 4. As per our understanding the function in Fig. 4 seems to indicate an attempt to transfer some specific string data to the user for display or transfer from user to some other end. The particular contents of the memory are not available to us since we are performing a static analysis and thus to get a deeper understanding, we would have to manually analyze a larger portion of disassembled code. However, it is clear that some functionality is present only in scareware instances, which would suggest that it is possible to differentiate them from benign files on a general level. However, it would be hard for a human expert to detect and analyze such subtle differences; therefore we argue that our automatic approach is superior, especially when considering the fact that regular applications can contain several thousands of lines of code.

TABLE II. SELECTED FEATURES AND THEIR NUMBER OF OCCURRENCE IN EACH CLASS

F.No	Feature	Jrip		J48		SMO	
		S	L	S	L	S	L
1	pushpushandoutsw	>0		>0		-0.142	
2	ormovadd			>0		-0.0813	
3	inswpopaw		>=1				0.1940
4	incoutswoutsb		>=1			-0.0195	
5	addmovmovmovcmp			>0		-0.0615	
6	leadb		>1		>0		0.2016
7	dbdecmov	>0		>0		-0.0848	
8	outswarplfs			>0		-0.0223	
9	movpushmov	>0				-0.0572	
10	Pushaddpush		>1				0.1704

S – Scareware, L – Legitimate software.

V. CONCLUSIONS AND FUTURE WORK

We have extended the heuristic-based detection technique using a variable length instruction sequence mining approach for the purpose of scareware detection. Since scareware is a rather recent software security threat, there are no publicly available data sets to generate classification models from. We have therefore obtained a large sample of scareware applications and designed an algorithm for extracting instruction sequences from these applications (and similarly for legitimate software). The data sets used in this study will be publically available at http://www.bth.se/com/rks. The experimental results are promising: the Random Forest algorithm managed to yield an AUC score of 0.972 after the complete data set was processed using the categorical proportional difference feature selection algorithm. Moreover, the results also indicate that our method is trustworthy since the false negative rate (the rate of scareware classified as legitimate) is considerably low (0.023). For future work, we aim to conduct further experiments on an even larger collection of scareware and benign files. We also plan to employ a hybrid identification method, which would integrate variable length instruction sequences with features extracted from, e.g., the end user license agreement or the information about the system calls a particular program makes.

REFERENCES

[1] Microsoft, “Rogue Security Software | Fake Virus Alerts | Scareware.” [Online]. Available: http://www.microsoft.com/security/antivirus/rogue.aspx. [Accessed: 12-Jan-2011].

[2] G. Cluley, “Sizing Up The Malware Threat - Key Malware Trends for 2010,” *Network Security*, vol. 2010, no. 4, pp. 8-10.

[3] The Washington Post, “Security Fix - Massive Profits Fueling Rogue Antivirus Market.” [Online]. Available: http://voices.washingtonpost.com/securityfix/2009/03/obscene_profits_fuel_rogue_ant.html?wprss=securityfix. [Accessed: 13-Jan-2011].

[4] Lavasoft AB, “Lavasoft.” [Online]. Available: http://www.lavasoft.com/. [Accessed: 13-Jul-2010].

[5] M. Cova, C. Leita, O. Thonnard, A. Keromytis, and M. Dacier, “An Analysis of Rogue AV Campaigns,” in *Recent Advances in Intrusion Detection*, pp. 442–463.

[6] M. A. Rajab, L. Ballard, P. Mavrommatis, N. Provos, and X. Zhao, “The Nocebo Effect on The Web: An Analysis of Fake Anti-virus Distribution,” in *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more (LEET 10)*, 2010, pp. 3-3.

[7] J. Stewart, “Windows Messenger Popup Spam on UDP Port 1026.” [Online]. Available: http://www.secureworks.com/research/threats/popup-spam/. [Accessed: 19-Nov-2010].

[8] L. Corrons, “The Business of Rogueware,” *Web Application Security*, pp. 7–7, 2010.

[9] Symantec Corporation, “Symantec Report on Rogue Security Software Press Kit.” [Online]. Available: http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=istr_rogue_security. [Accessed: 20-Jan-2011].

[10] “Aftonbladet.” [Online]. Available: http://www.aftonbladet.se/nyheter/article13218796.ab. [Accessed: 24-Jun-2011].

[11] “Swedish Windows Security User Group.” [Online]. Available: http://winsec.se/?cat=166. [Accessed: 23-Jun-2011].

[12] 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)*, pp. 38-49.

[13] S. Dolev and N. Tzachar, “Malware Signature Builder and Detection for Executable Code,” U.S. Patent EP2189920.

[14] G. Salton, A. Wong, and C. S. Yang, “A Vector Space Model for Automatic Indexing,” *Communications of the ACM*, vol. 18, pp. 613–620.

[15] 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.

[16] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[17] 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.

[18] Jau-Hwang Wang, P. S. Deng, Yi-Shen Fan, Li-Jing Jaw, and Yu-Ching Liu, “Virus Detection using Data Mining Techinques,” in *Proceedings of IEEE 37th Annual 2003 International Carnahan Conference on Security Technology*, 2003, pp. 71-76.

[19] 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.

[20] 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.

[21] CNET, “Free Software Downloads.” [Online]. Available: http://download.cnet.com/. [Accessed: 02-Jan-2010].

[22] F-Secure Corporation, “F-Secure - A Global IT Security & Antivirus Provider.” [Online]. Available: http://www.f-secure.com/. [Accessed: 02-Oct-2010].

[23] “The Netwide Assembler: NASM.” [Online]. Available: http://www.nasm.us/. [Accessed: 13-Jul-2010].

[24] 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.

[25] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[26] J. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines,” *Technical Report MST-TR-98-14. Microsoft Research*, 1998.

[27] C. Feng and D. Michie, “Machine Learning of Rules and Trees,” in *Machine learning, neural and statistical classification*, Ellis Horwood, 1994, pp. 50-83.

[28] T. Cover and P. Hart, “Nearest Neighbor Pattern Classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, Jan. 1967.

[29] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[30] F. 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 (ICML 98)*, 1998, pp. 445-53.