Data mining methods for malware detection using instruction sequences

Article · January 2008

CITATIONS
34

READS
2,993

3 authors, including:

Muazzam Ahmed Siddiqui
King Abdulaziz University
41 PUBLICATIONS **573** CITATIONS

SEE PROFILE

Morgan Wang
University of Central Florida
38 PUBLICATIONS **1,324** CITATIONS

SEE PROFILE

DATA MINING METHODS FOR MALWARE DETECTION USING INSTRUCTION SEQUENCES

Muazzam Siddiqui
Institute of Simulation & Training
University of Central Florida
siddiqui@mail.ucf.edu

Morgan C. Wang
Department of Statistics and Actuarial Sciences
University of Central Florida
cwang@mail.ucf.edu

Joohan Lee
School of Electrical Engineering & Computer Science
University of Central Florida
jlee@cs.ucf.edu

ABSTRACT
Malicious programs pose a serious threat to computer security. Traditional approaches using signatures to detect malicious programs pose little danger to new and unseen programs whose signatures are not available. The focus of the research is shifting from using signature patterns to identify a specific malicious program and/or its variants to discover the general malicious behavior in the programs. This paper presents a novel idea of automatically identifying critical instruction sequences that can classify between malicious and clean programs using data mining techniques. Based upon general statistics gathered from these instruction sequences we formulated the problem as a binary classification problem and built logistic regression, neural networks and decision tree models. Our approach showed 98.4% detection rate on new programs whose data was not used in the model building process.

KEY WORDS
Data Mining, Malware Detection, Binary Classification, Static Analysis, Disassembly, Instruction Sequences

1 Introduction

Computer virus detection has evolved into malicious program detection since Cohen first formalized the term computer virus in 1983 [10]. Malicious programs, commonly termed as malwares, can be classified into viruses, worms, trojans, spywares, adwares and a variety of other classes and subclasses that sometimes overlap and blur the boundaries among these classes [18]. The most common detection method is the signature based detection that makes the core of every commercial anti-virus program. To avoid detection by the traditional signature based algorithms, a number of stealth techniques have been developed by the malicious code writers. The inability of traditional signature based detection approaches to catch these new breed of malicious programs has shifted the focus of virus research to find more generalized and scalable features that can identify malicious behavior as a process instead of a single static signature.

The analysis can roughly be divided into static and dynamic analysis. In the static analysis the code of the program is examined without actually running the program while in dynamic analysis the program is executed in a real or virtual environment. Static analysis, while free from the execution overhead, has its limitation when there is a dynamic decision point in the program control flow. Dynamic analysis monitors the execution of the program to identify behavior that might be deemed malicious. These two approaches are combined also [17] where dynamic analysis is applied only at the decision-making points in the program control flow.

In this paper we present a static analysis method using data mining techniques to automatically extract critical behavior from malicious and clean programs. We introduce the idea of using instruction sequences extracted from the disassembly of malicious and clean programs as the primary classification feature. The behavior represented by these sequences of instructions is deemed critical if they can classify between malicious and clean programs.

The difference among our approach and other static analysis approaches mentioned in the related research section are as follows.

First, the proposed approach applies data mining as a complete process from data preparation to model building. Although data preparation is a very important step in a data mining process, almost all existing static analysis techniques mentioned in the related research section did not discuss this step in detail except [19]. Second, all features were instruction sequences extracted by the disassembly instead of using fixed length of bytes such as n-gram. The advantages are:

1. These instruction sequences can be traced back to their original location in the program for a further analysis of their associated operations.

2. These features can be grouped together to form additional derived features to increase classification accuracy.

3. A significant number of sequences that appeared in

only clean program or malicious programs can be eliminated to speed up the modeling process.

4. The classifier obtained can achieve 98% detection rate for new and unseen malicious programs.

It is worth noting that a dataset prepared for a neural network classifier might not be suitable for other data mining techniques such as decision tree or random forest.

2 Related Research

Data mining has been the focus of many virus researchers in the recent years to detect unknown viruses. A number of classifiers have been built and shown to have very high accuracy rates. [12] introduced the idea of using tell-tale signs to use general program patterns instead of specific signatures. The tell-tale signs reflect specific program behavior and actions that identify a malicious activity. Though a tell-tale sign like a sequence of specific functions calls seems a promising identifier, yet they did not provide any experimental results for unknown malicious programs.

The idea of tell-tale signs was furthered by [8] and they included program control and data flow graphs in the analysis. Based upon the tell-tale signs idea they defined a security policy using a security automata. The flow graphs are subjected to these security automata to verify against any malicious activity. The method is applied to only one malicious program. No other experimental results were reported to describe algorithm efficiency, especially on unseen data.

[19] developed PEAT (The Portable Executable Analysis Tool) to detect structural anomalies inside a program. PEAT rested on the basic principle that the inserted code in a program disrupts its structural integrity and hence by using statistical attributes and visualization tools this can be detected. The visualization tools plot the probability of finding some specific subject of interest in a particular area of the program. These subjects include sequence of bytes, their ASCII representation, their disassembly representation and memory access via register offsets. Statistical analysis was done on instruction frequencies, instruction patterns, register offsets, jump and call offsets, entropy of opcode values and code and ASCII probabilities. The experimental results were provided for only one malicious program.

Several researchers have used hexadecimal dump of the executable files to identify malicious programs. The dump is broken down into small chunks of pre-defined length called n-grams where n represents the length of the chunk. These n-grams were used to build a variety of classifiers to classify malicious programs from benign ones. As the number of n-grams from the dump is usually very high several techniques from text classification [20] have been employed to select the best features.

[7] used n-grams as features to build multiple neural network classifiers and adopted a voting strategy to predict the final outcome. Their dataset consisted of 53902 clean files and 72 variant sets of different viruses. For clean files, n-grams were extracted from the entire file while only those portions of a virus file are considered that remain constant through different variants of the same virus. A simple threshold pruning algorithm was used to reduce the number of n-grams to use as features. The results they reported are not very promising and even according to them, the procedure is not sufficient to be used as sole criteria in a virus scanner.

In another data mining approach, [15] used three different types of features and a variety of classifiers to detect malicious programs. Their primary dataset contained 3265 malicious and 1001 clean programs. They applied RIPPER (a rule based system) to the DLL dataset. Strings data was used to fit a Naive Bayes classifier while n-grams were used to train a Multi-Naive Bayes classifier with a voting strategy. No n-gram reduction algorithm was reported to be used. Instead data set partitioning was used and 6 Naive-Bayes classifiers were trained on each partition of the data. They used different features to built different classifiers that do not pose a fair comparison among the classifiers. Naive-Bayes using strings gave the best accuracy in their model.

A similar approach was used by [11], where they built different classifiers including Instance-based Learner, TFIDF, Naive-Bayes, Support vector machines, Decision tree, boosted Naive-Bayes, SVMs and boosted decision tree. Their primary dataset consisted of 1971 clean and 1651 malicious programs. Information gain was used to choose top 500 n-grams as features. Best efficiency was reported using the boosted decision tree J48 algorithm.

[5] used n-grams to build class profiles using kNN algorithm. Their dataset was small with 25 malicious and 40 benign programs. As the dataset is relatively small no n-gram reduction was reported. They reported 98% accuracy rate on a three-fold cross validation experiment. It would be interesting to see how the algorithm scale as a bigger dataset is used.

Besides n-grams, another common feature used for static analysis is the sequence of API calls. Most of the research that does not include data mining as a process compare the sequence of API calls against a pre-defined security policy encoded as finite state automata [8], or against a known malware sequence using similarity measures such as Euclidean distance, sequence alignment or some other similarity function [16]. Static analysis can also be assisted by dynamic analysis especially when dealing with encrypted and polymorphic malicious programs to extract the sequence of API calls [13]. Alternatively static analysis can be used to identify the location of system calls within the executable which can be monitored at runtime to verify that every observed system call is made from a location identified using static analysis [14]. Other techniques include monitoring windows registry for abnormal access [6] and extending software verification methods to detect obfuscated malicious programs [9]. All of this work stated above that does not include data mining as a process used very few samples to validate their techniques. The security

```
cmp     [ebp+var_8], 9
jge     short loc_40216D
mov     edx, [ebp+var_8]
mov     eax, [ebp+var_8]
mov     cl, byte_4095A0[eax]
mov     byte ptr [ebp+edx+lParam], cl
jmp     short loc_40214C
mov     [ebp+var_2F], 20h
mov     [ebp+var_8], 0
jmp     short loc_402183
```

Figure 1. Portion of the output of disassembled file.

policies needed human experts to devise general characteristics of malicious programs.

Data preparation is a very important step in a data mining process. Except [19], none of the authors presented above, discussed their dataset in detail. Malicious programs used by these researchers are very eclectic in nature exhibiting different program structures and applying the same classifier to every program does not guarantee similar results. Encrypted and polymorphic viruses always look different with each infection. Using the same n-grams patterns to identify such malicious programs will not work for any new infection.

3 Data Processing

Our primary dataset consisted of 820 Windows PE files divided equally into malicious and clean programs. The clean programs were obtained from a PC running Windows XP. These include small Windows applications such as calc, notepad, etc and other application programs running on the machine. The malicious programs were downloaded from [4]. The dataset was consisted of a wide range of programs created using different compilers and resulting in a sample set of uniform representation. Each malicious program in our dataset belonged to one of the three main categories of virus, worm and trojan. We only considered Win32 file viruses.

Binaries were transformed to a disassembly representation that can be parsed to extract features. The disassembly was obtained using Datarescues' IDA Pro [1] with proper plug-ins installed for files packed with UPX and other packers. From these disassembled files we extracted instruction sequences that served as the primary source for the features in our dataset. A sequence is defined to be instructions in succession until a conditional or unconditional branch instruction and/or a function boundary is reached. Instruction sequences thus obtained are of variable lengths. We only considered the opcode and the operands were discarded from the analysis. Figure 1 shows a portion of the disassembly of the program abcwin.exe.

A parser written in PHP translated the disassembly in figure 1 to instruction sequences as depicted in figure 2. Each row in the dataset represents a single instruction sequence. The first column indicates if it is a clean or virus

```
0 abcwin.txt loc_258 2 cmp jge
0 abcwin.txt loc_259 5 mov mov mov mov jmp
0 abcwin.txt loc_261 3 mov mov jmp
```

Figure 2. Instruction sequences

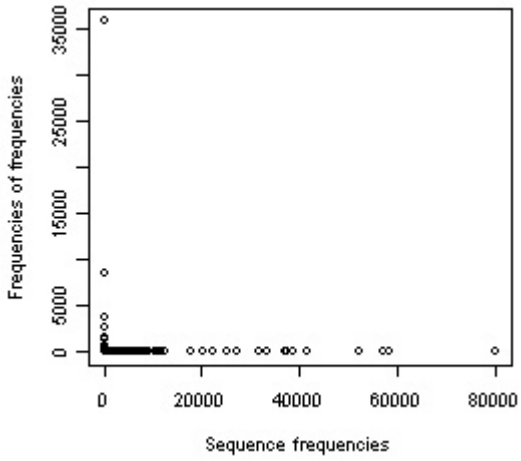

Figure 3. Scatter plot of sequence frequencies and their frequency of occurrence.

file, second column carries the file name. Third column carries the sequence name that the parser assigned to it. From fourth column and on, the instruction sequence starts.

The dataset did not contain any polymorphic or metamorphic viruses, as the static disassembly of such files produces invalid results for two reasons. First, the encryption is not broken easily like any unmodified UPX decompression and second and more important is that the malware will be mutated with any other infection spawning a different set of instruction sequences.

3.1 Feature Extraction

The raw disassembly of the virus and clean files resulted in 1510421 sequences. Among them 62608 unique sequences were identified with different frequencies of occurrence. Each sequence was considered as a potential feature. Figure 3 displays the scatter plot of sequence frequencies and the number of times those frequencies were spotted in the files. Points close to the y-axis are rare sequences while the points close to the x-axis and on the farther right corner of the plot represents the most occurring sequences. Thus the instruction sequences that were spotted once in the dataset were 35830 while only one sequences was spotted 80030 times. Figure 4 displays the same plot with the outliers removed. This shows the actual operating region from where the dataset can be built. Rare sequences will reduce the classifier to a signature detection engine while the ones that are too frequent cannot be used as identifiers.

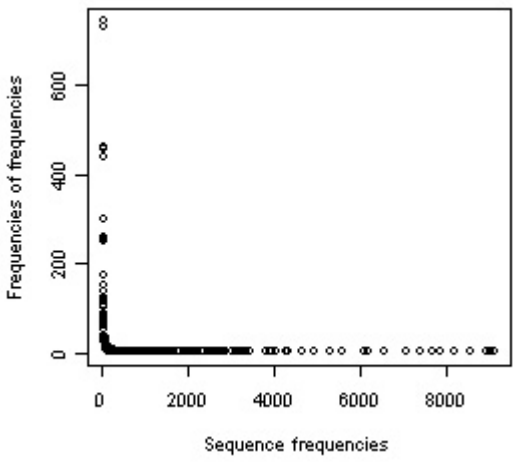

Figure 4. Scatter plot of sequence frequencies and their frequency of occurrence with outliers removed.

3.2 Feature Selection

The frequency of occurrence of each sequence in the entire data was considered to be the primary selection criteria. Sequences with less than 10% frequency of occurrence were identified as rare items and were not included in the dataset. This removed 98.2% of the sequences and only 1134 sequences were selected. The dataset consisted of frequency of occurrence of each of these sequences in each file. A binary target variable identified each file as virus or clean.

Using the occurrence frequency as the primary data item in the dataset enabled us to consider the features as count variables.

3.3 Data Preprocessing

A Chi-Square test of independence was performed for each feature to determine if a relationship exists between the feature and the target variable. The variables were transformed to their binary representation on a found/not found basis to get a 2-way contingency table. Using a p-value of 0.01 for the test resulted in the removal of about half of the features that did not showed any statistically significant relationship with the target. Features having their entire occurrence in one class only were also removed from the analysis, as they tend to bias the classifier. Out of initial 1134 features, 633 remained after this step. Some features have too many levels to implement data mining methods such as logistic regression and neural networks efficiently. An m-to-n mapping was used by applying the ChiMerge technique to collapse the number of levels within a feature.

4 Experiments

The data was partitioned into 70% training and 30% test data. We fitted three models to the training data using SAS Enterprise Miner. These included logistic regression, neural network and decision tree.

4.1 Logistic Regression

Regression is a statistical technique that determines the best model relating a dependent variable to a set of independent variables. Logistic regression is the type of regression that estimates the probability that the dependent variable will assume a value, instead of estimating the value of the variable. In our case logistic regression model was used to calculate the posterior probabilities that a given observation belonged to malicious class or benign.

In the Enterprise Miner implementation of logistic regression, we only considered main effects due to large number of input features. Two-factor interactions and polynomial terms were also not considered for the same reason.

4.2 Neural Network

A neural network is a network of simple processing elements (neurons) that can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. It can be used to model complex nonlinear relationship between inputs and outputs by learning from experiential knowledge expressed through the connections between the processing elements. We used neural network model to learn to classify between malicious and benign class by repetitively presenting observations from both classes.

The model used MLP architecture with 10 hidden units and misclassification as the model selection criteria.

4.3 Decision Tree

A decision tree recursively partitions the predictor space to model the relationship between predictor variables and categorical response variable. Using a set of input-output samples a tree is constructed. The learning system adopts a top-down approach that searches for a solution in a part of the search space. Traversing the resultant tree gives a set of rules that finally classified each observation into the given classes. We used the decision tree model to obtain a set of rules that can classify each sample into either malicious or benign class.

The decision tree model we used in Enterprise Miner used entropy as split criterion with a maximum depth of 30.

Figure 5 displays the training ROC curves for each model.

5 Results

We tested the models using the test data. Confusion matrices were created for each classifier using the actual and predicted responses. The following four estimates define the members of the matrix.

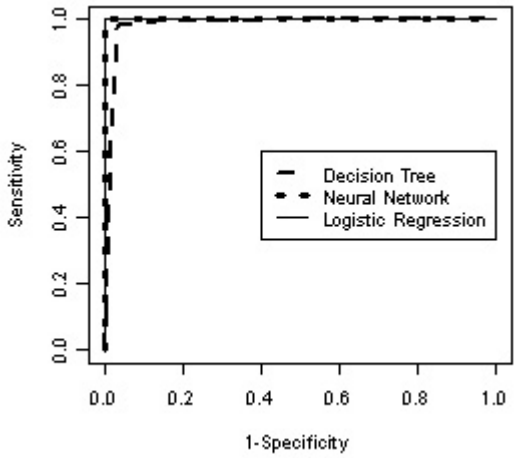

Figure 5. ROC curve comparing regression, neural network and decision tree training results.

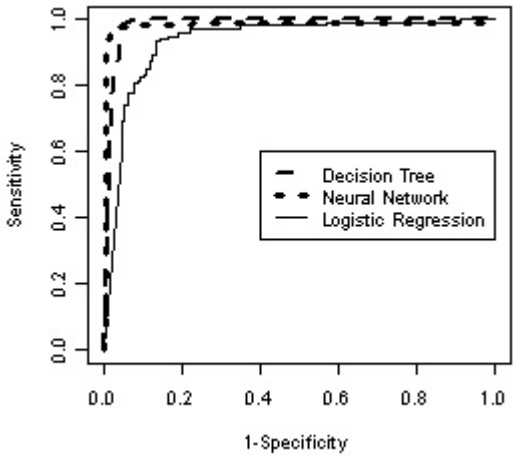

Figure 6. ROC curve comparing regression, neural network and decision tree test results.

Table 1. Experimental results

Classifier	Detection Rate	False Alarm Rate	Overall Accuracy
Logistic Regression	95%	17.5%	88.6%
Neural Network	97.6%	4.1%	97.6%
Decision Tree	98.4%	4.9%	96.7%

True Positive (TP): Number of correctly identified malicious programs.

False Positive (FP): Number of wrongly identified benign programs.

True Negative (TN): Number of correctly identified benign programs.

False Negative (FN): Number of wrongly identified malicious programs.

The performance of each classifier was evaluated using the detection rate, false alarm rate and overall accuracy that can be defined as follows:

Detection Rate: Percentage of correctly identified malicious programs.

$DetectionRate = \frac{TP}{TP+FN}$

False Alarm Rate: Percentage of wrongly identified benign programs.

$FalseAlarmRate = \frac{FP}{TN+FP}$

Overall Accuracy: Percentage of correctly identified programs.

$OverallAccuracy = \frac{TP+TN}{TP+TN+FP+FN}$

Table 1 displays the experimental results for each classifier.

Figure 6 displays the ROC curves for test data for each model. The slight bent in the curve for decision tree exhibits that the classifier was more prone to false positive rate although it displayed a slightly better detection rate. Neural network gave a slightly better false positive rate with a lower detection rate. Logistic regression did not perform well on the test data as the ROC curve for training displays overfitting for the model. Table 2 displays the area under the ROC curve for each model. The better performance of decision tree can be attributed to the fact that the dataset included count variables. Decision tree as a logic based model performed better on the discrete data as opposed to neural network or logistic regression that are more suited for continuous variables.

Table 2. Area under the ROC curve for each classifier.

Classifier	AUC
Logistic Regression	0.9357
Neural Network	0.9815
Decision Tree	0.9817

6 Conclusions

In this paper we presented a data mining framework to detect malicious programs. The primary feature used for the process was the frequency of occurrence of various instruction sequences. The effect of using such a feature set is two fold as the instruction sequences can be traced back to the original code for further analysis in addition to being used in the classifier. We used the sequences common to both malicious and benign programs to remove any biases caused by the features that have all their occurrences in one class only. We showed 98.4% detection rate with a 4.9% false positive rate.

7 Future Work

We provided a proof of concept work for our proposed data mining framework to detect malwares. The dataset used did not include the new breed of viruses with mutation capabilities. Our next step is to experiment with a larger sample set including polymorphic viruses. To deal with these type of viruses the static analysis will be facilitated by dynamic analysis using emultors such as VMWare [3]. The virus will be allowed to decrypt itself to obtain the actual representation that can be used for disassembly. An analyzer program such as PEiD [2] will be used to gather more statistics about the compiler used to create the program and potential packers with encryption routines used. As decision tree performed best on our dataset, it is natural to use random forest in the next step to see the classification gain. Bagging and boosting will be used in addition to random forest.

References

[1] IDA Pro Disassembler. http://www.datarescue.com/idabase/index.htm.

[2] PEiD. http://peid.has.it/.

[3] VMware. http://www.vmware.com/.

[4] VX Heavens. http://vx.netlux.org.

[5] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04) - Volume 02*, pages 41–42, 2004.

[6] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo. Detecting malicious software by monitoring anomalous windows registry accesses. Technical report, 2001.

[7] W. Arnold and G. Tesauro. Automatically generated win32 heuristic virus detection. In *Virus Bulletin Conference*, pages 123–132, 2000.

[8] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. *Symposium on Requirements Engineering for Information Security (SREIS'01)*, 2001.

[9] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium (Security'03)*, pages 169–186, 2003.

[10] F. Cohen. *Computer Viruses*. PhD thesis, University of Southern California, 1985.

[11] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[12] R. W. Lo, K. N. Levitt, and R. A. Olsson. Mcf: A malicious code filter. *Computers and Security*, 14(6):541–566, 1995.

[13] A. Mori. Detecting unknown computer viruses - a new approach -. *Lecture Notes in Computer Science*, pages 226–241, 2004.

[14] J. C. Rabek, R. I. Khazan, S. M. Lewandowski, and R. K. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 76–82, 2003.

[15] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, 2001.

[16] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables. In *20th Annual Computer Security Applications Conference*, pages 326–334, 2004.

[17] Symantec. Understanding heuristics: Symantec's bloodhound technology. Technical report, Symantec Corporation, 1997.

[18] P. Szor. *The Art of Computer Virus Research and Defense*. Addison Wesley for Symantec Press, New Jersey, 2005.

[19] M. Weber, M. Schmid, M. Schatz, and D. Geyer. A toolkit for detecting and analyzing malicious software. In *Proceedings of the 18th Annual Computer Security Applications Conference*, page 423, 2002.

[20] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420, 1997.