COMPUSOFT, An international journal of advanced computer technology, 3 (4), April-2014 (Volume-III, Issue-IV)

ISSN:2320-0790

An International Journal of Advanced Computer Technology

Implementnig Data Mining for Detection of Malware from Code

Dharmesh Kumar Babubhai Patel[1], **Sahjanand Harshadbhai Bhatt** [2]
[1]Shree P.M.Patel College of Computer Science and Technology, Anand, India
[2]Shree P.M.Patel Institute of P.G.Studies and Research in Applied Science, Anand, India

Abstract: In this paper we discuss various data mining techniques that we have successfully applied for cyber security. This research investigates the use of data mining methods for malware (malicious programs) detection and proposed a framework as an alternative to the traditional signature detection methods. These applications include malicious code detection by mining binary executables by anomaly detection, and data stream mining. A serious security threat today is malicious executables, especially new, unseen malicious executables often arriving as email attachments. These new malicious executables are created at the rate of thousands every year and pose a serious security threat. Our research is closely related to information retrieval and classification techniques and borrows a number of ideas from the field. Current anti-virus systems attempt to detect these new malicious programs with heuristics generated by hand. This approach is costly and oftentimes ineffective. We present a data-mining framework that detects new, previously unseen malicious executables accurately and automatically. The data-mining framework automatically found patterns in our data set and used these patterns to detect a set of new malicious binaries. Comparing our detection methods with a traditional signature based method; this method is more than doubles the current detection rates for new malicious executables.

Introduction:

Computer virus detection has evolved into malicious program detection since Cohen first formalized the term computer virus in 1983 [3]. Malicious programs 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 [4]. A malicious executable is defined to be a program that performs a malicious function, such as compromising a system's security, damaging a system or obtaining sensitive information without the user's permission. And build a scanner that accurately detects malicious executables before they have been given a chance to run. One of the primary problems faced by the virus community is to devise methods for detecting new malicious programs that have not yet been analyzed [1]. Eight to ten malicious programs are created every day and most cannot be accurately detected until signatures have been generated for them.

Current virus scanner technology has two parts: a **signature-based detector** and a **heuristic classifier** that detects new viruses [2]. The classic signature-based detection algorithm relies on signatures (unique telltale strings) of known malicious executables to generate detection models. Heuristic classifiers are generated by a group of virus experts to detect new malicious programs. This kind of analysis can be time-consuming and oftentimes still fail to detect new malicious executables.

Data mining refers to extracting or 'mining' interesting knowledge from large amounts of data [5].It provides a means of extracting previously unknown, predictive information from the base of accessible data in data warehouses. Data mining tools use sophisticated, automated algorithms to discover hidden patterns, correlations, and relationships among organizational data. These tools are used to predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions [6].

Data mining methods detect patterns in large amounts of data, Using data mining methods, our goal is to automatically design such as byte code, and use these patterns to detect future instances in similar data. Introduction to some of the Data Mining Methods are described below:

Definitions

This section introduces various computer security terms to the reader. Since our work deals with computer virus and worms, a more detailed account of these categories is presented than any other area in security.

Computer Security

Computer security is the effort to create a secure computing platform, designed so that agents (users or programs) can only perform actions that have been allowed.

Malware

Any program that is purposefully created to harm the computer system operations or data is termed as malicious programs. Malicious programs include viruses, worms, trojans, backdoors, adwares, spywares, bots, rootkits etc. All malwares are sometimes loosely termed as virus (viruses, worms, trojans specifically) Commercial anti-malware products are still called antivirus.

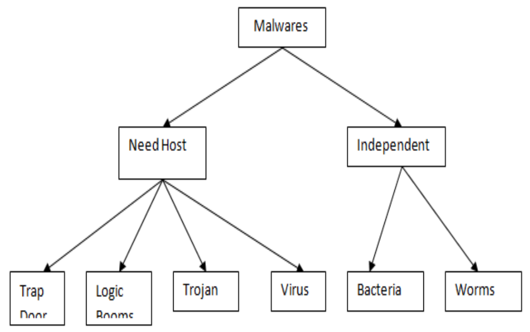


Figure 1 Classification of Malwares

Background and Related work Review

We consider details of most relevant malware detection techniques in this section. In recent years many malware researchers have focused on data mining to detect unknown malwares. Data mining is the process of analyzing electronically stored data by automatically searching for patterns [7]. Machine learning algorithms have been used widely for different data mining problems to detect patterns and to find correlations between data instances and attributes. Many researchers have used n-grams or API calls as their primary type of feature that are used to represent malware instances in a suitable format for data mining purposes. Shultz et al. [8] proposed a method using data mining techniques for detecting new malicious executables. Three different types of features are extracted from the executables, i.e. the list of DLLs used by the binary, the list of DLL function calls, and number of different system calls used within each DLL. Also they analyze byte sequences extracted from the hexdump of an executable. The data set consisted of 4,266 files out of which 3,265 were malicious and 1,001 were legitimate or benign programs. A rule induction algorithm called Ripper [9] was applied to find patterns in the DLL data. A learning algorithm Naïve Bayes (NB), which is based on Bayesian statistics, was used to find patterns in the string data and n-grams of byte sequences were used as input data for the Multinomial Naïve Bayes algorithm. A data set is partitioned in two data sets, i.e., a test data set and a training data set. This is to allow for performance testing on data that are independent from the data used to generate the classifiers. The Naïve Bayes algorithm, using strings as input data, yielded the highest classification performance with an accuracy of 97.11%. The authors compared their results with traditional signature-based methods and claimed that the data mining-based detection rate of new malware was twice as high in comparison to the signature-based algorithm. A similar approach was used by J. Z. Kolter et al. [10], where they use n-gram analysis and data mining approaches to detect malicious executables in the wild. The authors used a hexdump utility to convert each executable to hexadecimal code in an ASCII format and produced n-gram features by combining each four-byte sequence into a single term. Their primary dataset consisted of 1971 clean and 1651 malicious programs They used different classifiers including Instance based Learner, TFIDF, Naive-Bayes, Support vector machines, Decision tree, boosted Naive- Bayes, SVMs and boosted decision tree. They used information gain to select valued features which are provided as input to all classifiers. The area under an ROC curve (AUC) is a more complete measure compared with the detection accuracy as they reported [11]. AUCs show that the boosted decision trees outperform rest of the classifiers for both classification problems. M. Siddiqui et al. [12] used Data Mining for detection of Worms. They used variable length instruction sequence. Their Primary data set consists of 2,775 Windows PE files, in which in which 1,444 were worms and 1,330 were benign. They performed detection of compilers, common packers and crypto before disassembly of files. Sequence reduction was performed and 97% of the sequences were removed. They used Decision Tree, Bagging and Random Forest models using. Random forest performed slightly better than the others.

Dynamic malware analysis techniques have previously focused on obtaining reliable and accurate information on execution of malicious programs [14,15]. As it was mentioned in the introduction, the main focus of our work lies in automatic processing of information collected from dynamic malware analysis. Two techniques for behavior-based malware analysis using clustering of behavior reports have been recently proposed [16, 17]. Both methods transform reports of observed behavior into sequences and use sequential distances (the normalized compression distance and the edit distance, respectively) to group them into clusters which are believed to correspond to malware families. The main difficulty of clustering methods stems from their unsupervised nature, i.e., the lack of any external information provided to guide analysis of data. Let us illustrate some practical problems of clustering-based approaches

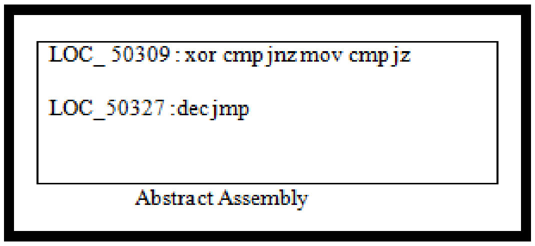
Methodology:

This chapter provides a description of, the much required, theoretical foundation for our work and the general framework that we developed to carry out our experiments.

The data mining process to be consisted of five steps.

- Problem statement and formulation of hypothesis
- Data collection
- Data preprocessing
- Model estimation
- Model interpretation

In this paper we present a virus detection approach through data mining. For that we used some virus files from corpus data set and some viruses generate from vcl32 virus kit. First of all we take 2000 virus files from corpus data set and vcl32 virus generator. Then through IDpro disassemble, disassemble all virus file and generate ASM files from those. In a disassemble, assembly instructions are organized into basic blocks. We make logic assembly and abstract assembly from those files. Disassemble will generate a label for each basic block automatically. We believe that basic block capture the structure of instruction sequences and we process the instructions and make basic blocks. That code is "logic assembly" code [5]. Each assembly instruction consists of opcode and operands. We use only opcode and ignore the operands and prefix because that say behavior of program. The resulting assembly code is called "abstract assembly" [5]. Final abstract assembly as show below



```
LOC_ 50309 : xor cmp jnz mov cmp jz

LOC_50327 :dec jmp
```

Abstract Assembly

Figure 2 Example of abstract assembly

Major steps in our work

- Make virus data sets.
- Disassemble virus files using any disassemble.
- Generate abstract assembly opcode.
- Feature selection algorithm.

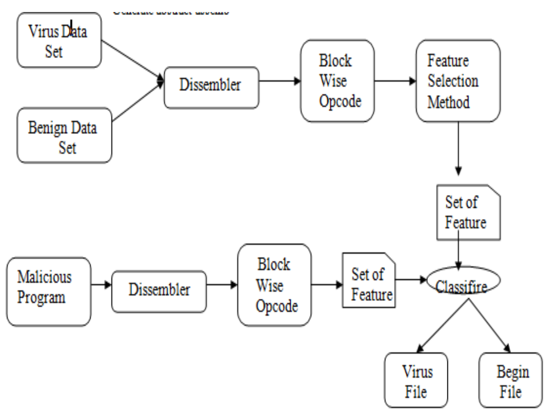


Figure 3 Major Steps in our Approach

Make virus data sets.

For our work we generated a virus data set for the 200 file and 500 file.

Disassemble virus files using any disassemble.

We converted the virus definition using the disassembler to convert it into non executable format. Accordingly disassembler will translate the virus definition to the non executable format.

Generate abstract assembly opcode.

We generated the opcode for the translated virus definition for our work.

Feature Selection

The features for our classifier are instruction associations. To select appropriate instruction associations, we use the following two criteria:

1. The instruction associations should be frequent in the training data set. If it occurs very rarely, we would rather consider this instruction association is a noise and not use it as our features.

2. The instruction association should be an indicator of malicious code.

To satisfy the criteria, we only extract frequent instruction associations from training dataset. Only frequent instruction associations can be considered as our features. We use a variation of Apriori algorithm to generate all three types of frequent instruction

associations from abstract assembly. One parameter of Apriori algorithm is "minimum support". It is the minimal frequency of frequent associations among all data. More specifically, it is the minimum percentage of basic blocks that contains the instruction sequences in our case. Normalized count is the frequency of that instruction sequence divided by the total number of basic blocks in abstract assembly. We can also use N gram approach to find feature set from that data.
Then select top L features as our feature set. For one executable in training dataset, we count the number of basic blocks containing the feature, normalized by the number of basic blocks of that executable. We process every executable in our training dataset, and eventually we generate the input for our classifier as like Naive Bayes, Ripper .

Following Steps are Shown Basic Architecture

Step 1: Disassemble all files and generate abstract assembly.
Step 2: Find frequency of each instruction association (IA) according type 1 and 2
Step 3: Sort all instruction sequence and select top 10 sequences of length k.
Step 4: Take ith no. of training files (virus and benign) and find frequency of each IA at block level.
Step 5: Make table of selected IA frequencies from training files.
Step 6: Repeat step 4 and 5 for Type 1, 2 and length 2, 3 IA.

Algorithm
Find the frequent item sets: the sets of items that have minimum support
INPUT: Set of virus files (V)
OUTPUT: Set of top instruction sequences (L).
In order to generate set of instruction sequences we have set of virus file. In each virus file we have no. of basic blocks. Form the basic blocks occurrence of instruction sequences is calculated, which is called as instruction association. This algorithm repeats until all set of virus file encountered. Finally we select top L sequences which are
Called as top L virus features.
Following are the basic steps for generating top L instruction sequences.

```
1. For (each virus file Vi in V) do
2. For (each basic block Bij in Vi) do
3. Record all sequences of length sl found in
   Bij (with out repetition)
4. Increase count of all instruction sequences.
5. End For
6. End For
7. Select top L sequences.
```

What we used in our work:

Virus data set:
(i) 200 files from corpus data set
(ii) 500 files from vcl32 generator

IDA Pro: Disassembler to generate ASM file from malicious files.
Virus Code: ASM file of any virus file
Opcode selector: select opcode from asm files and make logic assembly and abstract assembly.
Abstract assembly: Opcode of all virus file as per basic blocks.

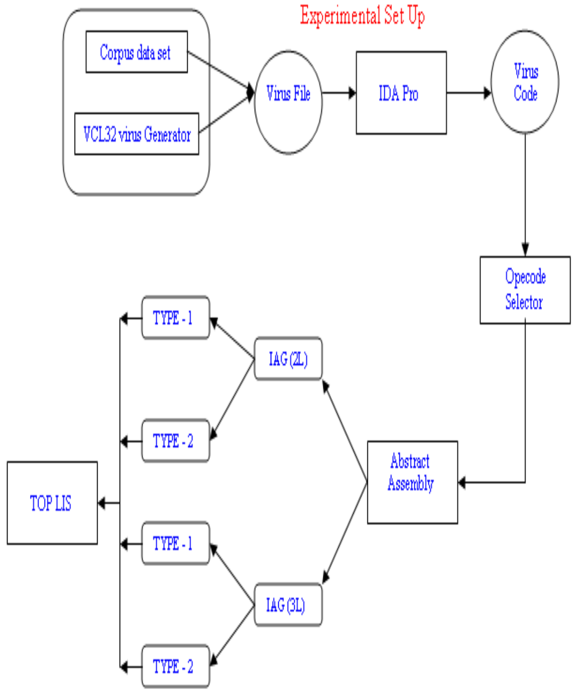
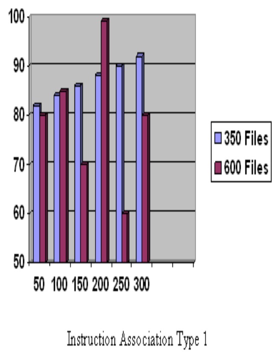


Figure 4 Virus Data set
In above fig virus files are generated from VCL32 and corpus data set. Through idpro disassemble we generate instruction code of those files. We present whole model for select top L feature from malicious data set. We generate a data set of malicious programs and disassemble all files. Then we use opcode selector for refine virus code and generate abstract assembly.

Results
Experimental Setup
(A) Model Trained by Neural Network Classifier
Following results are comparison between 600 files and 350 files trained by NN model. Graph shows better results for NN model which is trained by 600 files as compared to NN model trained by 350 files.

From the help of Graph it is concluded that NN model trained by more files produces better results.



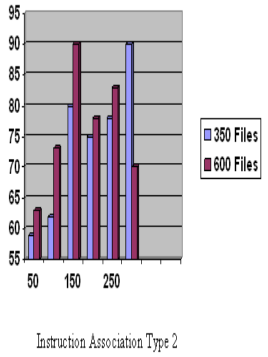
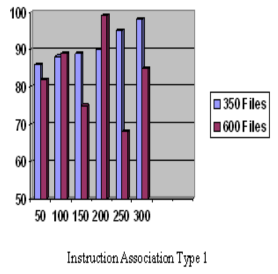


Figure 5 Figure 6

(B) Model Trained by SVM Classifier

Following results are comparison between 600 files and 350 files trained by SVM model. Next graph shows better we implemented a feature search method that focuses on selecting features that are applicable to different families of viruses. This ensured that our classifier does not rely on signatures. In experimental testing our method achieved better performance as compared to some of older virus detection techniques. By using both SVM and NN models, the selected features which are used by the classifier produce overall support within the data set. This indicates that our feature search method produces features which are more useful while detecting new unseen viruses. We also introduced an evaluation method for virus classifiers that tests more convincingly its ability to detect new viruses. Our results show that system which uses family non-specific features performs better results. In future work we propose focusing on reducing the false positive rate, by using a large number of benign files, or by training our classifier using a cost matrix and setting a higher cost to misclassifying negative examples. This would involve by using a set of older viruses in the training set and a set of more recent ones in the test set.



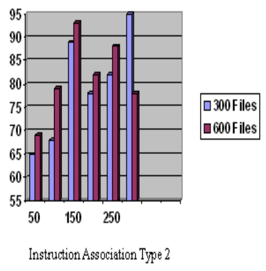


Figure 7 Figure 8

Conclusion:

This dissertation provided an introduction to the malware research using data mining techniques. We applied a four tier hierarchy to organize the research and included data mining in the top tier of detection methods. Our work closely resembles information retrieval and classification techniques, where we replaced text documents with computer programs and applied similar techniques. Unlike previous attempts at applying data mining techniques at a syntactic level, using n-grams, we introduced a variable length instruction sequence that inherently captures program control flow information and hence provides a semantic level interpretation to the program analysis. In our quest to choose the best classification model, we compared different feature selection methods and classifiers and provided empirical results. We proposed using association analysis for feature selection and automatic signature extraction and were able to receive as low as 1.9% false positive rate and as high as 93% overall accuracy on novel malwares with the algorithms that we devised. In the end future work is proposed to extend the association analysis method for improved classification and time and space complexity for the algorithms.

References

[1]Steve R. White. Open Problems in Computer Virus Research. Virus Bulletin Conference, 1998.

[2] Dmitry Gryaznov. Scanners of the Year 2000: Heuristics.Proceedings of the 5th International Virus Bulletin, 1999

[3] Fred Cohen. Computer Viruses. PhD thesis, University of Southern California, 1985.

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

[5] The Data Mine: www.the-data-mine.com

[6] KDnuggets - Data Mining, Web Mining, and Knowledge Discovery Guide: www.kdnuggets.com

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

[8] M. G. Schultz, E. Eskin, E. Z., and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in Proceedings of the IEEE Symp. on Security and Privacy, pp. 38-49, 2001.

[9] W. Cohen, ."Fast effective rule induction,." Proc. 12th International Conference on Machine Learning, pp. 115-23, San Francisco, CA: Morgan Kaufmann Publishers, 1995.

[10] J. Z. Kolter and M. A. Maloof, “Learning to Detect Malicious Executables in the wild,” in Proceedings of the ACM Symp. on Knowledge Discovery and Data Mining (KDD), pp. 470-478, August 2004.

[11] T. Fawcett, “ROC Graphs: Notes and Practical Considerations for Researchers”, TR HPL-2003-4, HP Labs, USA, 2004.

[12] M. Siddiqui, M. C. Wang, J. Lee, Detecting Internet worms Using Data Mining Techniques, Journal of Systemics, Cybernetics and Informatics, volume 6 - number 6, pp: 48-53, 2009.

[13] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A tool for analyzing malware. In Proceedings of EICAR 2006, April 2006.

[14] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. Journal in Computer Virology, 2:67–77, 2006.

[15] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario.Automated classification and analysis of internet malware. In Proceedings of the 10th Symposium on Recent Advances in Intrusion Detection (RAID’07), pages 178–197, 2007.

[16] T. Lee and J. J. Mody. Behavioral classification. In Proceedings of EICAR 2006, April 2006.

[17] Dan Ellis. “Worm Anatomy and Model.” In Proceedings of the 2003 ACM workshop on Rapid malcode WORM ’03, pp. 42–50, 2003.