A Layer based Internet Worm Detection and Classification

[1]M. Madhu Purna Chandra Rao, [2]Y. Chittibabu, [3]Dr. P. Harini
[1,2,3]Dept. of CSE, St.Ann's College of Engg & Technology, Chirala, AP, India

Abstract
Internet worms pose a somber threat to computer safety. Traditional draw near using signatures to sense worms pose a slight risk of the zero day boat-hire. The spotlight of malware study is variable from with signature model to support the mean behavior show by the malware. This paper there a new idea of remove variable length tuition series that can distinguish worms from a clean list with data removal system. The study was facilitated by the program sort outflow in arranging controlled in the instructional series. Stand upon common data gather round from these order sequences we devise the problem as a binary classification problem and built tree found classifiers including choice tree, bagging and chance forest. Or come close to showed 95.6% result rate on novel worms whose information was not used in the model structure process.

Keywords
Data Mining, Worm Detection, Binary Classification, Static Analysis, Disassembly, Tuition Sequences

I. Introduction

Computer virus detection has evolved into malware detection since Cohen first formalized the term computer virus in 1983 [13]. Malicious programs, often termed as malware, can be confidential into virus, worms, Trojans, spywares, adware's and a variety of other classes and subclasses that sometimes overlap and blur the boundaries among these groups [24]. The most general detection method is the signature based detection that makes the core of every commercial antivirus program. To avoid detection by the traditional signature based algorithms, a number of stealth method has been developed by the malware writers. The incapability of traditional signature based detection approaches to catch these new breed of malware has shifted the focus of malware 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 approximately be separated 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 surroundings. The motionless analysis, while free from the execution overhead, has its limit when there is a dynamic decision point in the program control flow. Dynamic analysis checks the execution of the program to identify behavior that might be considered malicious. These two approaches are joint also [23] 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 behavior from worms and clean programs. We set up the idea of using a sequence of instructions extracted from the disassembly of worms and clean programs as the primary classification feature. Unlike stable length instructions or programs, the variable length instructions naturally capture the program control flow information as every sequence reflects a control flow block.

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

First, the proposed approach applied 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 method mentioned in the related research section did not discuss this step in feature except [25]. Second, all features were sequences of instructions extracted by the disassembly instead of using fixed length of bytes such as n-gram. The compensation are:

1. The instruction sequences include program control flow information, not present in programs.
2. The instruction sequences capture information from the program at a semantic level rather than syntactic level.
3. These instruction sequences can be traced back to their original location in the program for further analysis of their associated operations.
4. These features can be grouped together to form additional derived features to increase classification accuracy.
5. A significant number of sequences that appeared in only clean program or worms can be eliminated to speed up the modeling process.
6. The classifier obtained can achieve 95% detection rate for new and unseen worms.

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

II. Related Research

Divided worm detection into three main categories; Traffic monitoring, honeypots and signature detection. Traffic analysis includes monitor network traffic for anomalies like a sudden increase in traffic volume or change in the traffic pattern for some hosts etc. Honey pots are devoted systems installed in the network to collect data that are passively analyzed for potential malicious activities. Signature detection is the mainly common method of worm detection where network traffic logs, system logs or files are searched for worm signatures.

Data mining has been the focus of many malware researchers in the current years to detect unknown malware. A number of classifiers have been built and exposed to have very high accuracy rates. Data mining provides the earnings for analysis and detection of malware for the categories distinct above. The majority of these classifiers use the program or API calls as their main feature. A program is a sequence of bytes of a given length extracted from the hexadecimal dump of the file. Besides file dumps, network traffic data and honey pot data are mined for malicious activities.

Introduced the idea of using telltale signs to use general program patterns instead of specific signatures. The telltale signs replicate specific program behaviors and actions that identify a malicious activity. Though a telltale sign like a sequence of specific function calls seems a promising identifier, yet they did not provide any experimental results for unknown malicious programs.

The idea of telltale signs was furthered by [10] and they included program control and data flow graphs in the analysis. Based upon the telltale sign idea, they define a safety 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 new results were reported to describe algorithm competence, particularly on unseen data.

In another data mining approach, [20] used three dissimilar types

of features and a variety of classifiers detect malicious programs. Their primary data set together with this 3265 malicious and 1001 clean programs. They applied RIPPER (a rule based system) to the DLL data set. String data were used to well a Naive Bayes classifier while programs were used to train a Multi-Naive Bayes classifier with a voting strategy. No program reduction algorithm was reported to be used. In its place data set partitioning was used and 6 Naive-Bayes classifiers were trained on each partition of the data. They used different features to build different classifiers that do not pose a fair comparison among the classifiers. Naive-Bayes using strings gave the finest accuracy in their model.
A similar approach was used by [15], 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 data set consisted of 1971 clean and 1651 malicious programs. Information increase was used to choose top 500 programs as features. Best capability was reported using the boosted decision tree J48 algorithm.
Used program to build class profiles using the KNN algorithm. Their data set was small with 25 malicious and 40 benign programs. As the data set is relatively little, non-gram reduction was reported. They reported 98% correctness rate on a threefold cross corroboration experiment. It would be interesting to see how the algorithm scale as a bigger data set is used.
Proposed a signature based method called SAVE (Static Analysis of Vicious Executables) that used behavioral signatures indicating malicious activity. The signatures were positioned for in the form of API calls and Euclidean distance was used to evaluate these signatures with a sequence of API calls from programs under inspection. Besides data mining, other accepted methods include activity monitoring and file scanning. [19] proposed a system to detect scanning worms using the premises that scanning worms tend to reside in hosts with low successful connections rates. Every unsuccessful or successful connection attempt was assigned a score that signals a host to be infected if past a threshold. [14] proposed behavioral signatures to detect worms in network traffic data. [16] industrial Honeycomb, that used honeypots to generate network signatures to detect worms. Honeycomb used irregularity in the traffic data to generate signatures.
All of this work stated above, that does not include data mining as a process, used extremely a small number of samples to validate their method. The security policies required human experts to devise general characteristics of malicious programs.
Data preparation is a very significant step in a data mining process. Except [25], none of the authors presented above have discussed their datasets in detail. Malicious programs used by these researchers are extremely eclectic in nature exhibiting different program structures and applying the similar classifier to each program does not guarantee similar results.

III. Data Processing

Our collection of worms and clean programs consisted of 2775 Windows PE files, in which 1444 were worms and the 1330 were clean programs. The clean programs were getting from a PC running Windows XP. These include small Windows applications such as Calc, notepad, etc. And extra application programs running on the machine. The worms were downloaded from [8]. The data set was thus consisted of a wide variety of programs, created using dissimilar compilers and resulting in a sample set of uniform representation. Fig. 3 shows the data processing steps.

A. Malware Analysis

We ran PEiD [5] and Ex Einfo PE [2] on our data collection to notice compilers, common packers and cryptors, used to compile and/or change the programs. Table 1 displays

Table 1: Packers/Compilers Analysis of Worms

Packer/Compiler	Number of Worms
ASPack	77
Borland	110
FSG	31
Microsoft	336
Other Not Packed	234
Other Packed	83
PECompact	26
Unidentified	140
UPX	67

Table 2: Packers/Compilers Analysis of Worms and Clean Programs

Type of Program	Not Packed	Packed	Unidentified
Clean	1002	0	49
Worm	624	340	140
Total	1626	340	189

The distribution of dissimilar packers and compilers of the worm collection.
The clean programs in our collection were also subjected to PEiD and ExeInfo PE to gather potential packers/crytpors information. No packed programs were detected in the clean collection. Table 2 displays the number of packet, not packed and unidentified worms and clean programs.
Before additional processing, packed worms were unpacked using specific unpacks such as UPX (with -d switch) [6], and generic unpacks such as Generic Unpacker Win32 [3] and VMUnpacker [7].

B. File Size Analysis

Before disassembling the programs to extract instruction sequences, a file size analysis was performed to make sure that the number of instructions extracted from clean programs and worms is roughly equal. Table 3, displays the file size statistics for worms and clean programs.
Table 3, indicates the that the average size of the clean programs is twice as big as the average worm size. These large programs were removed from the collection to get an equal file size distribution for worms and clean programs.

Table 3: File Size Analysis of the Program Collection

Statistic	Worms Size (KB)	Cleans Size (KB)
Average	67	147
Median	33	43
Minimum	1	1
Maximum	762	1968

C. Disassembly

Binaries were transformed to a disassembly representation that is parsed to extract features. The disassembly was obtained using Data rescues' IDA Pro [4]. From these disassembled files we extracted sequences of instructions that served as the primary source for the features in our data set. A sequence is definite as instructions in succession until a conditional or unconditional branch instruction and/or a functional boundary is accomplished. Instruction sequences thus obtained are of a variety of lengths. We only considered the opcode and the operands were discarded from the analysis. Fig. 1 shows a part of the disassembly of the Netsky. A worm.

D. Parsing

A parser written in PHP translates the disassembly in fig. 1, to instruction sequences. Fig. 2, displays the output of the parser. Every row in the parsed output represented a single instruction sequence. The raw disassembly of the worm and clean programs resulted in 1972920 instruction sequences. 47% of these sequences belonged to worms while 53% belonged to clean programs.

E. Feature Extraction

The parsed output was processed through our Feature Extraction Mechanism. Among them 1972920 instruction sequences, 213330 unique sequences were identified with different frequencies of occurrence. We separate the sequences that were found in one class only as they will reduce the classifier to a signature detection technique. This removed 94% of the sequences and only 23738 sequences were found common to both worms and clean programs. Every sequence was considered as a potential feature.

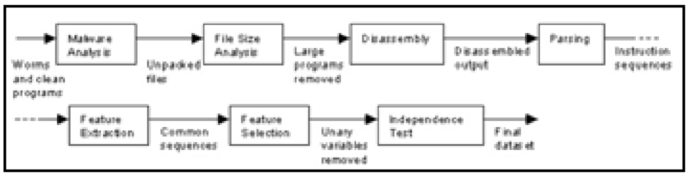

Fig. 1: Data Preprocessing Steps

F. Feature Selection

The Feature Selection Mechanism considered the frequency of occurrence of every sequence in the whole data to be the primary selection criteria. Sequences with below 10% frequency of occurrence were identified as rare items are were not included in the data set. This detached 97% of the sequences and only 679 sequences were selected. The data set consisted of frequency of occurrence of each of these sequences in every file. A binary target variable identified every file as a worm or cleaner.
Using the incidence frequency as the primary data item in the data set enabled us to consider the features as count variables.

G. Independence Test

A Chi-Square test of independence was performed for every feature to determine if a relationship exists among the feature and the target variable. The variables were distorted to their binary representation on a found/not found basis to get a 2-way opportunity table. Using a p-value of 0.01 for the test resulted in the removal of about half of the features that did not demonstrate any statistically significant relationship with the goal. The resulting number of variables after this step was 268.

IV. Experiments

The data were partitioned into 70% training and 30% test data. Parallel experiments showed the best results with a tree based models for the count data [21]. We construct decision tree, bagging and casual forest models using R [1].

A. 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 model a tree is built. The learning system takes on a top-down approach that searches for a solution in a part of the search space. Traversing the ensuing tree gives a set of rules that finally classified each observation into the given class. We used the decision tree model to obtain a set of rules that can classify every sample into either malicious or benign class.
The decision tree model we build in R used the Gini as a split criterion with a maximum depth of 15.

B. Bagging

Bagging or Bootstrap Aggregating is a meta-algorithm to recover classification and regression models in terms of accuracy and constancy. Bagging generates multiple versions of a classifier and the uses plurality vote to decide for the final class outcome between the versions. The many versions are created using bootstrap imitations of the unique data set. Bagging can give a substantial increase in accuracy by improving on the unsteadiness of individual classifiers [11].
We used categorization trees with 100 bootstrap replications in the Bagging model.

C. Random Forest

Random forest provides a degree of development over Bagging by minimizing correlation between classifiers in the band. This is able by using bootstrapping to generate multiple versions of a classifier as in Bagging but employing only a random subset of the variables to split at every node, instead of all the variables as in Bagging. Using a casual selection of features to split every node yields error rates that evaluate positively to Adaboost, but are more robust with respect to noise. [12]
We grew 100 classification trees in the Random forest model. The number of variables sampled at every split was 22.

V. Results

We tested the models using the test data. Confusion matrices were created for every classifier using the actual and predicted responses. The following four approximations define the members of the matrix.
True Positive (TP): Number of correctly documented 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 recognized malicious programs.
The performance of every classifier was evaluated using the detection rate, false alarm rate and overall correctness that can be defined as follows:
Detection Rate: Percentage of properly identified malicious programs.

Table 4: Experimental Results

Classifier	Detection Rate	False Alarm Rate	Overall Accuracy
Random Forest	95.6%	3.8%	96%
Bagging	94.3%	6.7%	93.8%
Decision Tree	93.4%	13.4%	90%

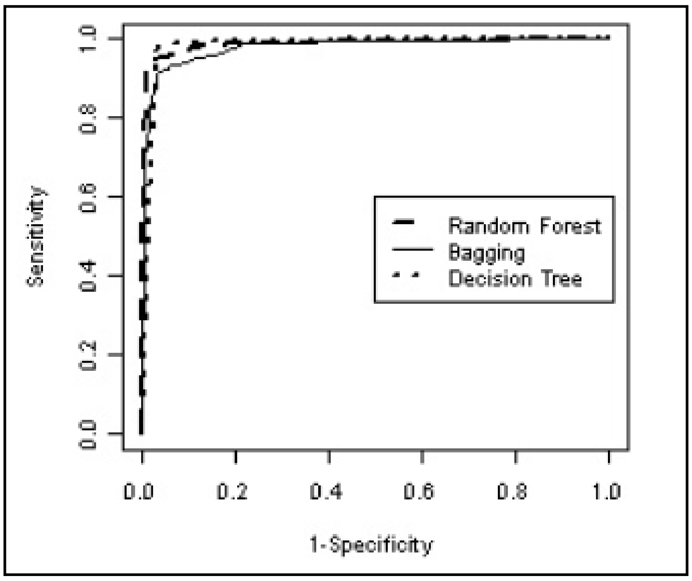

Fig. 2: ROC Curve Comparing Decision Tree, Bagging and Random Forest Test Results

Detection Rate= $TP/_{TP+FN}$

False Alarm Rate: Percentage of incorrectly identified benign programs.

FalseAlarmRate= $FP/_{TN+FP}$

Table 4, displays the experimental results for each classifier. Fig. 4, displays the ROC curves for test data for each model. The meta algorithms execute better than a single decision tree as probable. Random forest performed a small better than Bagging which is an endorsement of its advantage over Bagging as claimed in [12]

VI. Conclusions

In this paper we obtainable a data mining framework to detect worms. The primary characteristic used in the process was the frequency of occurrence of variable length instruction

Table 5: Area Under the ROC Curve for Each Classifier

Classifier	AUC
Decision Tree	0.9060
Bagging	0.9775
Random Forest	0.9871

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 generally to both worms and clean programs to remove any biases caused by the features that have all their incidence in one class only. We illustrate 95.6% detection rate with a 3.8% fake positive rate.

VII. Future Work

The information included in this analysis was extracted from the executable section of the PE file. To achieve an enhanced detection rate this information will be appended from information from other sections of the file. This will include bringing in Address Table and the PE header. API calls analysis has completed to be an competent tool in malware detection [22]. In addition header information has been used in heuristic detection [24]. Our next step is to include this information in our feature set.

References

[1] "The r project for statistical computing", [Online] Available: http://www.rproject.org/.
[2] ExEinfo PE. [Online] Available: http://www.exeinfo.go.pl/.
[3] "Generic Unpacker Win32". [Online] Available: http://www.exetools.com/unpackers.htm.
[4] "IDA Pro Disassembler", [Online] Available: http://www.datarescue.com/idabase/index.htm.
[5] PEiD, [Online] Available: http://peid.has.it/.
[6] "UPX the Ultimate Packer for eXecutables", [Online] Available: http://www.exeinfo.go.pl/.
[7] "VMUnpacker", [Online] Available: http://dswlab.com/d3.html.
[8] "VX Heavens", [Online] Available: http://vx.netlux.org.
[9] T. Abou-Assaleh, N. Cercone, V. Keselj, R. Sweidan, "N-gram-based detection of new maliciouscode", In Proceedings of the 28th Annual InternationalComputer Software and Applications Conference- Workshops and Fast Abstracts - (COMPSAC'04) - Vol. 02, pp. 41–42, 2004.
[10] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui,Y. Lavoie, N. Tawbi,"Static detection ofmalicious code in executable programs", Symposiumon Requirements Engineering for Information Security(SREIS'01), 2001.
[11] L. Breiman,"Bagging predictors", Machine Learning, 24(2), pp. 123–140, 1996.
[12] L. Breiman,"Random forests", Machine Learning, 45(1): pp. 5–32, 2001.
[13] F. Cohen. Computer Viruses. PhD thesis, Universityof Southern California, 1985.
[14] D. Ellis, J. Aiken, K. Attwood, S. Tenaglia,"Abehavioral approach to worm detection", In Proceedingsof the 2004 ACM Workshop on Rapid Malcode, pp. 43–53, 2004.
[15] J. Z. Kolter, M. A. Maloof.,"Learning to detectmalicious executables in the wild", In Proceedings of the 2004 ACM SIGKDD International Conference onKnowledge Discovery and Data Mining, 2004.
[16] C. Kreibich, J. Crowcroft,"Honeycomb creatingintrustion detection signatures using honeypots", In 2nd Workshop on Hot Topics in Network, 2003.
[17] R. W. Lo, K. N. Levitt, R. A. Olsson,"Mcf:A malicious code filter. Computers and Security", 14(6), pp. 541–566, 1995.
[18] J. Nazario,"Defense, and Detection Strategies against Internet Worms", Van Nostrand Reinhold, 2004.
[19] S. E. Schechter, J. Jung, B. A. W.,"Fast detectionof scanning worms infections", In Proceedings of SeventhInternational Symposium on Recent Advances inIntrusion Detection (RAID), 2004.
[20] M. G. Schultz, E. Eskin, E. Zadok, S. J. Stolfo,"Data mining methods for detection of new maliciousexecutables", In Proceedings of the IEEE Symposiumon Security and Privacy, pp. 38–49, 2001.

[21] M. Siddiqui, M. C. Wang, J. Lee.,"Data miningmethods for malware detection using instruction sequences", In Proceedings of Artificial Intelligence andApplications, AIA 2008. ACTA Press, 2008.
[22] A. H. Sung, J. Xu, P. Chavez, S. Mukkamala,"Static analyzer of vicious executables", In 20th Annual Computer Security Applications Conference, pp. 326–334, 2004.
[23] Symantec,"Understanding heuristics: Symantec's bloodhound technology", Technical report, Symantec Corporation, 1997.
[24] P. Szor,"The Art of Computer Virus Research and Defense", Addison Wesley for Symantec Press, NewJersey, 2005.
[25] M. Weber, M. Schmid, M. Schatz, D. Geyer.,"A toolkit for detecting and analyzing malicious software", In Proceedings of the 18th Annual Computer Security Applications Conference, pp. 423, 2002.

Mr. Madhu Purnachandra Rao received B.Tech degree in Computer Science and Engineering from JNTU University, Kakinada, in 2010, pursuing M.Tech. degree in Computer Science and Engineering in St. Ann's College Of Engineering and Technology, Chirala, affiliated to JNTU Kakinada, AP, India

Mr. Y. Chittibabu is an Associate Professor of Computer Science & Engineering at St. Ann's College of Engineering & Technology; He completed M. Tech. in Computer Science from JNTU Kakinada. He is a Life Member in CSI & ISTE. He has 08 Years of Experience. He published more than 10 journals and conference in researches interests include Data Mining, Mobile Computing

Dr. P. Harini received B.E. degree in Electronics and Communications Engineering from University of Madras, Chennai, in 1993, received M.Tech. degree in Remote Sensing from JNTU, Hyderabad, in 1997, received M.Tech. degree in Computer Science and Engineering from JNTU, Hyderabad, in 2003 and received Ph.D. in Computer Science and Engineering from JNTU, Anantapur, in 2011. She has 16 Years of Experience in which 1 year of Industrial, 1 year of Research & over 14 years of rich Teaching Experience in reputed Engineering Colleges & She is currently working as Professor & HOD in Computer Science & Engineering department in St.Ann's College of Engineering & Technology, Chirala. She Published 21 Research papers in various International Journals & Conferences. Guided many UG & PG students for projects & Life time Member of ISTE & CSI. Conducted successfully many Workshops, Seminars, conferences, FDPs and many National Level Technical Symposiums.