IOSR IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. II (Mar-Apr. 2014), PP 40-46
www.iosrjournals.org

Efficient Detection of Internet Worms Using Data Mining Techniques

B. Sujatha, G.Rajitha devi
Asst.professor, lecturer

Abstract: *Internet worms pose a serious threat to computer security.Traditional approaches using signatures to detect worms pose little danger to the zero day attacks. The focus of malware research is shifting from using signature patterns to identifying the malicious behavior displayed by the malwaresThis paper presents a novel idea of extracting variable length instruction sequences that can identify worms from clean programs using data mining techniques.The analysis is facilitated by the program control flow information contained in the instruction sequences. Based upon general statistics gathered from these instruction sequences we formulated the problem as a binary classification problem and built tree based classifiers including C5.0, boosting and random forest. Our approach showed 99.5% detection rate on novel worms whose data was not used in the model building process.*

Keywords

- *Data Mining,*
- *Worm Detection,*
- *C5.0,Boosting,*
- *Feture selection using stepwise forward selection,*
- *static analysis,*
- *Diassembly,*
- *Insruction sequences.*

I. Introduction

Computer virus detection has evolved into malware detection since Cohen first formalized the term computer virus in 1983 [13]. Malicious programs, commonly termed as malwares, can be classified into virus, worms, trojans, spywares, adwares and a variety of other classes and subclasses that sometimes overlap and blur the boundaries among these groups [24]. 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 malware writers. The inability of traditional signature based detection approaches to catch these new breed of malwares 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 roughly be divided into static and dynamicanalysis. 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. The static analysis, while free from the execution overhead, has its limitation when there is a dynamic decision point in the programs control flow. Dynamic analysis monitors the execution of program to identify behavior that might be deemed malicious. These two approaches are combined 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 introduce the idea of using sequence of instructions extracted from the disassembly of worms and clean programs as the primary classification feature. Unlike fixed length instructions or n-grams, the variable length instructions inherently capture the programs control flow information as each sequence reflects a control flow block. The difference 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 techniques mentioned in the related research section did not discuss this step in detail 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 advantages are:

- The instruction sequences include program control flow information, not present in n-grams.
- The instruction sequences capture information from the program at a semantic level rather than syntactic level.
- These instruction sequences can be traced back to their original location in the program for further analysis

of their associated operations.

- These features can be grouped together to form additional derived features to increase classification accuracy.
- A significant number of sequences that appeared in only clean program or worms can be eliminated to speed up the modeling process.
- The classifier obtained can achieve 95% detection rate for new and unseen worms.
- 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.
-

II. Related Research

[18] Divided worm detection into three main categories; Traffic monitoring, honey pots and signature detection. Traffic analysis includes monitoring network traffic for anomalies like sudden increase in traffic volume or change in traffic pattern for some hosts etc. Honeypots are dedicated systems installed in the network to collect data that is passively analyzed for potential malicious activities. Signature detection is the most 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 recent years to detect unknown malwares.

A number of classifiers have been built and shown to have very high accuracy rates. Data mining provides the means for analysis and detection of malwares for the categories defined above. Most of these classifiers use n-gram or API calls as their primary feature. An n-gram 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 is mined for malicious activities. [17] introduced the idea of using tell-tale signs to use general program patterns instead of specific signatures. The tell-tale signs reflect 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 tell-tale signs was furthered by [10] 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. In another data mining approach, [20] 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 [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 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. [9] 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 ngram 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. [22] proposed a signature based method called SAVE (Static Analysis of Vicious Executables) that used behavioral signatures indicating malicious activity. The signatures were represented in the form of API calls and Euclidean distance was used to compare these signatures with sequence of API calls from programs under inspection. Besides data mining, other popular methods includes activity

Monitoring and file scanning. [19] proposed a system to detect scanning worms using the premises that scanning worms tend to reside on hosts with low successful connections rates. Each 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] developed Honeycomb, that used honeypots to generate network signatures to detect worms. oneycomb used anomalies in the traffic data to generate signatures. 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 policies needed human experts to devise general characteristics of malicious programs.

Data preparation is a very important step in a data mining process. Except [25], none of the authors presented above have 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.

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 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 worms were downloaded from [8]. The dataset was thus
consisted of a wide range of programs, created using different compilers and resulting in a sample set of uniform representation. Figure 3 displays the data processing steps.

❖ Malware Analysis

We ran PEiD [5] and ExEinfo PE [2] on our data collection to detect compilers, common packers and cryptors, used to compile and/or modify 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 different packers and compilers on 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 packed, not packed and unidentified worms and clean
programs. Before further processing, packed worms were unpacked using specific unpackers such as UPX (with -d switch) [6], and generic unpackers such as Generic Unpacker Win32
[3] and VMUnpacker [7].

❖ File Size Analysis

Before disassembling the programs to extract instruction sequences, a file size analysis was performed to ensure that the number of instructions extracted from clean programs and worms is approximately 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 large as 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

```
inc     si
jb      short near ptr loc_171+1
ins     word ptr es:[di], dx
cmp     ah, [bx+si]
inc     di
popa
jz      short near ptr loc_16E+1
dec     sp
outsw
arpl    [bp+di+58h], bp
xor     dh, [bx+si]
xor     [bx+si+74h], al
jb      short near ptr loc_178+3
```

Figure 1. Portion of the output of disassembled Netsky.A worm.

```
inc jb
ins cmp inc popa jz
dec arpl xor xor jb
```

Figure 2. Instruction sequences extracted from the disassembled Netsky.A worm.

❖ **Disassembly**

Binaries were transformed to a disassembly representation that is parsed to extract features. The disassembly was obtained using Datarescues' IDA Pro [4]. From these disassembled files we extracted sequences of instructions that served as the primary source for the features in our dataset. A sequence is defined as instructions in succession until a conditional or unconditional branch instruction and/or a function boundary is reached. Instruction sequences thus obtained are of various lengths. We only considered the opcode and the operands were discarded from the analysis. Figure 1 shows a portion of the disassembly of the Netsky.A worm.

❖ **Parsing**

A parser written in PHP translates the disassembly in figure 1 to instruction sequences. Figure 2 displays the output of the parser. Each 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.

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 removed 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.

Each sequence was considered as a potential feature.

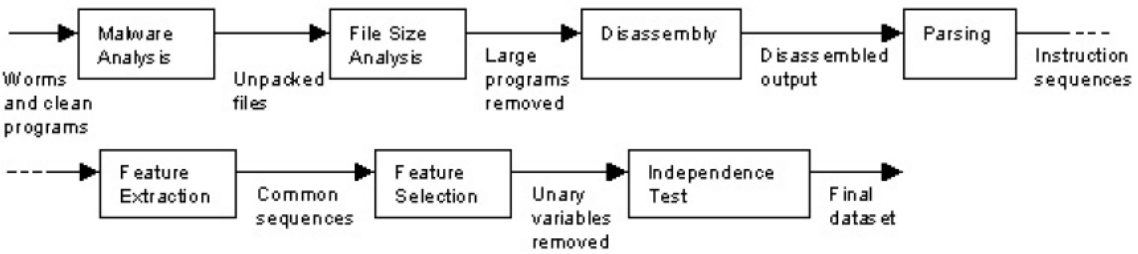


Figure 3. Data preprocessing steps.

❖ **Feature Selection**

Feature selection is the problem of choosing a small subset of features that ideally is necessary and sufficient to describe the target concept (Kiraand Rendell, 1992). The terms features, variables, measurements, and attributes are used interchangeably in the literature. Selecting the appropriate set of features is extremely important since the feature set selected is the only source of information for any learning algorithm using the data of interest. A goal of feature selection is to avoid selecting too many or too few features than is necessary. If too few features are selected, there is a good chance that the information content in this set of features is low. On the other hand, if too many (irrelevant) features are selected, the effects due to noise present in (most real-world) data may overshadow the information present. Hence, this is a tradeoff which must be addressed by any feature selection method. In statistical analyses, forward and backward stepwise multiple regressions (SMR) are widely used to select features, with forward SMR being used more often due to the lesser magnitude of calculations involved. The output here is the smallest subset of features resulting in an R2 (correlation coefficient) value that explains a significantly large amount of the variance. In forward SMR, the analyses proceeds by adding features to a subset until the addition of a new feature no longer results in a significant (usually at the 0.05 level) increment in explained variance (R2 value). In backward SMR, the full set of features are used to start with, while seeking to eliminate features with the smallest contribution to R2.

❖ **Independence Test**

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. The resulting number of variables after this step was 268.

IV. Experiments

The data was partitioned into 70% training and 30% testdata. Similar experiments showed best results with tree based models for the count data [21]. We built decision tree, bagging and Random forest models using R [1].

❖ **C5.0 CLASSIFIER**

The C5.0 algorithm is a new generation of Machine Learning Algorithms (MLAs) based on decision trees [16]. It means that the decision trees are built from list of possible attributes and set of training cases, and then the trees can be used to classify subsequent sets of test cases. C5.0 was developed as an improved version of well-known and widely used C4.5 classifier and it has several important advantages over its ancestor [17]. The generated rules are more accurate and the time used to generate them is lower (even around 360 times on some data sets). In C5.0 several new techniques were introduced: _ boosting: several decision trees are generated and combined to improve the predictions. _ Variable misclassification costs: it makes it possible to avoid errors which can result in a harm._ new attributes: dates, times, timestamps, ordered discrete attributes. _ values can be marked as missing or not applicable for particular cases. _ supports sampling and cross-validation. Here, the threshold was lowered and the fractional usage is shown.
C5.0 algorithm by embedding “boosting” technology in cost matrix and cost-sensitive tree to establish a new model C5.0 Better Than C4.5 The major differences are the tree sizes and computation times; C5.0's trees are noticeably smaller and C5.0 is faster

The error rate on unseen cases is reduced for all three datasets, substantially so in the case of forest for which the error rate of boosted classifiers is about half that of the corresponding C4.5 classifier. Unfortunately, boosting doesn't always help -- when the training cases are noisy, boosting can actually reduce classification accuracy. C5.0 uses a novel variant of boosting that is less affected by noise, thereby partly overcoming this limitation.

❖ **New functionality**

C5.0 incorporates several new facilities such as variable misclassification costs. In C4.5, all errors are treated as equal, but in practical applications some classification errors are more serious than others. C5.0 allows a separate cost to be defined for each predicted/actual class pair; if this option is used, C5.0 then constructs classifiers to minimize expected misclassification costs rather than error rates.

The cases themselves may also be of unequal importance. In an application that classifies individuals as likely or not likely to "churn," for example, the importance of each case may vary with the size of the account. C5.0 has provision for a case weight attribute that quantifies the importance of each case; if this appears, C5.0 attempts to minimize the weighted predictive error rate.

C5.0 has several new data types in addition to those available in C4.5, including dates, times,

timestamps, ordered discrete attributes, and case labels. In addition to missing values, C5.0 allows values to be noted as not applicable. Further, C5.0 provides facilities for defining new attributes as functions of other attributes.

Some recent data mining applications are characterized by very high dimensionality, with hundreds or even thousands of attributes. C5.0 can automatically winnow the attributes before a classifier is constructed, discarding those that appear to be only marginally relevant. For high-dimensional applications, winnowing can lead to smaller classifiers and higher predictive accuracy, and can often reduce the time required to generate rulesets.

C5.0 is also easier to use. Options have been simplified and extended -- to support sampling and cross-validation, for instance -- and C4.5's programs for generating decision trees and rulesets have been merged into a single program.

❖ **Boosting**

Boosting – Combines models of the same type (e.g., decision tree) and it is iterative, i.e., a new model is influenced by the performance of the previously built model

- Boosting – Uses voting or averaging (similar to bagging)
- Different boosting algorithms exist Assumption: Learning algorithm can handle weighted instances (usually handled by randomization schemes for selection of training data subsets)
- By weighting instances, the learning algorithm can concentrate on instances with high weights (called "hard" instances), i.e., incorrectly classified All instances are equally weighted
- A learning algorithm is applied
- The weight of incorrectly classified examples is increased ("hard" instances), correctly – decreased ("easy" instances) (easy.
- The algorithm concentrates on incorrectly classified "hard" instances
- Some "had" instances become "harder" some "softer"
- A series of diverse experts (classifiers) is generated based on the reweighed data

The concept of boosting applies to the area of predictive data mining, to generate multiple models or classifiers (for prediction or classification.

Boosting will generate a sequence of classifiers, where each consecutive classifier in the sequence is an "expert" in classifying observations that were not well classified by those preceding it. During deployment (for prediction or classification of new cases), the predictions from the different classifiers can then be combined (e.g., via voting, or some weighted voting procedure) to derive a single best prediction or classification.

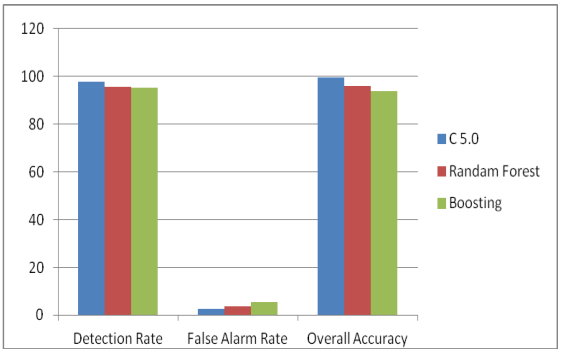
❖ **Random Forest**

Random forest provides a degree of improvement over Bagging by minimizing correlation between classifiers in the ensemble. This is achieved by using bootstraping to generate multiple versions of a classifier as in Bagging but employing only a random subset of the variables to split at each node, instead of all the variables as in Bagging. Using a random selection of features to split each node yields error rates that compare favorably 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 each split was 22.

V. Results

Classifier	Detection Rate	False Alarm Rate	Overall Accuracy
C 5.0	97.6	2.8	99.5
Randam Forest	95.6	3.8	96
Boosting	95.3	5.6	93.8



VI. Conclusion

Classifier	AUC
C 5.0	99.5
Randam Forest	96.0
Boosting	93.8

In this paper we presented a data mining framework to detect worms. The primary feature used for the process was the frequency of occurrence of variable length instruction sequences. The effect of using such a feature set is twofold 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 worms and clean programs to remove any biases caused by the features that have all their occurrences in one class only. We showed 99.5% detection rate with a 2.8% false positive rate.

VII. Future Work

The information included for this analysis was extracted from the executable section of the PE file. To achieve a better detection rate this information will be appended from information from other sections of the file. This will include Import Address Table and the PE header. API calls analysis has proven to be an effective tool in malware detection [22]. Moreover 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 http://www.rproject.org/.
[2] ExEinfo PE. http://www.exeinfo.go.pl/.
[3] Generic Unpacker Win32. http://www.exetools.com/unpackers.htm.
[4] IDA Pro Disassembler. http://www.datarescue.com/idabase/index.htm.
[5] PEiD. http://peid.has.it/.
[6] UPX the Ultimate Packer for eXecutables. http://www.exeinfo.go.pl/.
[7] VMUnpacker. http://dswlab.com/d3.html.
[8] VX Heavens. http://vx.netlux.org.
[9] 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.
[10] 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.
[11] L. Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996.
[12] L. Breiman. Random forests. Machine Learning, 45(1):5–32, 2001.
[13] F. Cohen. Computer Viruses. PhD thesis, University of Southern California, 1985.
[14] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In Proceedings of the 2004 ACM Workshop on Rapid Malcode, pages 43–53, 2004.
[15] 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.
[16] C. Kreibich and J. Crowcroft. Honeycomb creating intrustion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network, 2003.
[17] R. W. Lo, K. N. Levitt, and R. A. Olsson. Mcf: A malicious code filter. Computers and Security,14(6):541–566, 1995.
[18] J. Nazario. Defense and Detection Strategies against Internet Worms. Van Nostrand Reinhold, 2004.
[19] S. E. Schechter, J. Jung, , and B. A. W. fast detection of scanning worms infections. In Proceedings of Seventh International Symposium on Recent Advances in Intrusion Detection (RAID), 2004.
[20] 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.
[21] M. Siddiqui, M. C. Wang, and J. Lee. Data mining methods for malware detection using instruction sequences. In Proceedings of Artificial Intelligence and Applications, AIA 2008. ACTA Press, 2008.
[22] 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.
[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, New Jersey, 2005.
[25] 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.