A Survey on Data Mining approaches for Dynamic Analysis of Malwares

Kshitij Shah[1], Dushyant Kumar Singh[2],
[1]Student, [2]Assistant Professor, Department of CSE,
MNNIT Allahabad, Allahabad (UP)

Abstract--The number of samples being analyzed by the security vendors is continuously increasing on daily basis. Therefore generic automated malware detection tools are needed, to detect zero day threats. Using machine learning techniques, the exploitation of behavioral patterns obtained, can be done for classifying malwares (unknown samples) to their families. Variable length instructions of Intel x86 placed at any arbitrary addresses makes it affected by obfuscation techniques. Padding bytes insertion at locations that are unreachable during runtime tends static analyzers being confused to misinterpret binaries of program. Often the code that is actually running may not necessarily be the code which static analyzer analyzed. Such programs use polymorphism, metamorphism techniques and are self modifying. In this paper, using dynamic analysis of executable and based on mining techniques. Application Programming Interface (API) calls invoked by samples during execution are used as parameter of experimentation.

Keywords: Dynamic Analysis, API Calls, Classifiers, AdaBoost

1. INTRODUCTION

Malwares are referred as the software that attackers use deliberately to fulfill their harmful intentions. Their intent is to have control of resources as system CPU, network etc, and collecting personal data without taking consent of systems owner, and interrupting computer operations thus creating havoc to the privacy of its users and the availability of internet. Malwares are of different types including Worm's, Viruses, Backdoor, Trojan-horse, Spyware, Adware and Root-kits etc. These classes of malwares overlap in their characteristics, meaning that a particular malware could show behavior of multiple classes at any time i.e. they have something in common. The malware are increasing rapidly in quantity (growing great landscape), variety (new malice techniques) and velocity (rate of threats arrival). These are utilizing new methods to target computing devices, and are evolving, becoming more and more sophisticated.

Around 100,000 new malware samples are cataloged everyday by McAfee which means about sixty-nine novel threats per minute and approx 1 per second [1].

Various obfuscation techniques like insertion of dead code, register re-assignment, subroutine rearrangement, instruction replacement, code substitution, and code integration are used by malware writers to bypass the defense methods like firewalls, antivirus. Antivirus, firewalls generally use signature based techniques which makes them unable to detect the new kind of malwares [2]. Commercial vendors of anti-malwares tools needs to analyze zero day malwares to create their signatures and therefore are unable to provide instant security.

Various soft computing techniques for malware analysis are being followed to get over the constraints of signature based methods, whether static or dynamic, as different malware families typically have same behavioral patterns [1, 2]. The analysts can then well understand the associated risks with any malicious code sample, with the help of above techniques. To overcome the threats that could arise in future, some preventive measures in-line with new trends of malware creation can be exploited. The features which are derived from malware can be used for classifying unknown samples of malwares into their tagged families.

This paper investigates the problem of detecting malwares using soft computing techniques and section II is discussion on related work done in the area by various researchers. Section III gives a review of dynamic analysis techniques that can be used for analyzing and classifying the malware executables. It also specifies the feature selection technique (mRMR) used, and the boosting technique for classification. Section IV shows the results and performance characteristics recorded during the experiments. Finally, section V concludes the paper and gives direction to work that can be carried out in future.

2. LITERATURE SURVEY

The authors in [1] discuss the CWSandbox tool as a malware analyzer that fulfills the design criteria's of automation, effectiveness, and correctness. Tool better supports the Win32 families of operating systems. Zolkipliet. al. proposed a framework based on dynamic approach for behavior modeling and classification of malwares [3]. The authors in [2] presented a binary obfuscation technique and then using that proved that malware analyzers based on advanced semantics can be evaded. Therefore they demonstrated that static malware analysis couldn't be used as longer time solution to malware detection. [4] Gives a new classification approach called binary texture analysis which uses some new type of feature extraction. They demonstrated that image texture analysis based static classification can better complement the classification techniques based on dynamic behavioral analysis.The authors in

978-1-4673-7910-6/15/$31.00 ©2015 IEEE

[5] constructed a classification model incorporating both static as well as dynamic views in single unified framework. In [6] the authors proposed a hybrid detection technique for unknown malwares, which combines the frequency of occurrence of unknown code (statically obtained) with the information of the execution trace of an executable (dynamically obtained). The authors in [7] introduced a feature selection technique mRMR (minimal-redundancy-maximum-relevance) based on mutual information. Authors in [9] gave an overview of dynamic analysis techniques. Bayer et. al. [10] gave an efficient method for dynamic malware analysis which reduced the overall analysis time.

3. MALWARE ANALYSIS

We need to develop systems capable of analyzing malwares automatically in order to deal with the huge volume of malwares. Either by checking the code of program executable or by running that in an isolated environment, the malicious program and its associated risks and intentions can be observed. Analyses done for malicious software is static if analyzed without executing the executable. Static analysis make use of some patterns that are to be detected in unknown samples, as signature string, n-grams, operational code frequency distribution, mnemonic n-grams etc. Malware code dis-assembled by dis-assemblers as OllyDbg / IDAPro, are sequence of assembly instructions (Intel x86 for win32). The mnemonics of these instructions in some patterns gives characteristics to identify malwares. The code obfuscation techniques transform the malwares in such a way that it resists reverse engineering and makes static analysis expensive and unreliable.

3.1 Dynamic Analysis:

Dynamic analysis as reverse of static analysis involve running of executable in a controlled environment, for analyzing the behavior of the malicious code. Several online automated tools are available for dynamic analysis of malwares, e.g. Norman Sandbox, CWSandbox, Anubis, Hook Analyser, ThreatExpert, TTAnalyzer. The report of analysis given by these tools provides detailed conception and deep insight of the malware behavior and actions being performed. Behavior based dynamic analysis suffers less from evasion and obfuscation techniques, in contrast to code analysis. However, dynamic analysis takes more time with more resource for, to be done.

3.1.1 Approaches and Techniques to perform Dynamic analysis

1. **Function Call Monitoring**
   The APIs in many are provided by operating systems that are actually being used by applications to perform task. On windows system, the term Windows API means an APIs which grant access to different functionalities as networking, system services, security and management. Malware executes in user space while code of kernel mode has direct access of system state. Therefore by invoking the respective system callsmalware needs to communicate with its environment. Since the interaction of user processes with the environment, is only possible with system calls, so, API's are of special interest for dynamic malware analysis.

2. **Function Parameter Analysis**
   The correlation of individual function calls that operate on the same object is enabled by tracing of function parameters and return values. Logically coherent sets of grouped function calls provide detailed insight into the programs behavior.

3. **Information Flow Tracking**
   It basically involves tracking data flow throughout the system during program execution. This propagation information of data of interest is an attribute for analysis. Taint source introduces new labels (taint) into the system, this means tainting the data that is found legible to the analysis. A component of the system which gives warning when stimulated with tainted data is the taint sink. Eg.
   (a). Direct Data Dependencies (b). Address Dependencies (c). Control Flow Dependencies

4. **Instruction Trace**
   The instruction sequence of the program executable during execution used while analyzing can contain important information that can be used to classify malwares.

3.2 Machine Learning for Malwares Detection

Techniques based machine learning is found to be playing a needed role in almost all classification problems so are in malware analysis [12]. A feature set is extracted from malwares and then some supervised learning is applied to label new unknown malwares. In a sandbox environment, the behavior of each malware is analyzed automatically and corresponding behavioral reports are generated in case of dynamic analysis. Reports generated are then processed for further machine classification. The executable is disassembled and then the code is analyzed and features like mnemonic n-grams, opcode sequences, etc are extracted, using machine learning techniques, vector models are formed for further classification in case of static analysis. The classifiers used are like Naïve Bayes, Decision Tree, Multilayer Perceptron Neural Network (MLP), J48, Random Forest (RF), AdaBoost, Support Vector Machine (SVM), Sequential Minimal Optimization (SMO) etc.

3.3 Boosted Classifiers

Boosting is a technique to improvise classification by combining a number of classifiers in some topology. Researchers have proved that the performance is often improved by using ensemble methods over single classifier. Boosting consists of learning weak classifiers iteratively with respect to a distribution and adding them to a final strong classifier. They are weighted according to the weak classifier's accuracy when they are added. After that the data is reweighted such that the instances that were misclassified gain weight and those that were correctly classified lose weight. This is done so that the future weak classifiers do put more focus on the instances which the previous weak classifier misclassified. In our experiments we used AdaBoost algorithm to boost Naïve Bayes, Instance based Learner, and Linear Support Vector Machines.

Naïve Bayes – is a probabilistic method. Describing the concept, $P(C_i)$ denotes prior probability of ith class, and $P(v_j|C_i)$ is the conditional probability of jth attribute in the class. The estimates are done by counting in training data, the class frequency and the attribute values frequency for every class. Attributes assumed as conditionally independent, Bayes' rule computes posterior probability for an unknown instance for its every class, given as:

$$C = \operatorname{argmax} P(C_i) \prod P(v_j \mid C_i)$$

Instance Based Learner – an unknown instance is classified as the performance element. Find example in the set with highest similarity with the unknown example and return the class label found as its estimation for unknown example. In our experiments we have used distance as the similarity measure. The three instances found most similar to the unknown instance, where returns the class label with maximum as predicted label for unknown.

Linear Support Vector Machines – It's a linear classifier, with capacity of classifying even a high dimensionality data. It is described by a vector of 3 components weights (w), an interceptor (a), and a threshold (b). Class prediction for classification is done as positive if $(w.x) - b > 0$ while negative other case.

Algorithm

1. Select support vectors to start with. eg S1, S2, S3….Sn
2. Augment the vectors with a "1" as a bias unit to find the threshold.
3. Find parameters a1, a2, … based on following equations

(S1.S1)a1 + (S1.S2)a2 +(S1.S3)a3+…..(S1.Sn)an = class to which S1 belongs (1)

(S2.S1)a1 + (S2.S2)a2 +(S2.S3)a3+…..(S2.Sn)an = class to which S2 belongs (2)

And so on till

(Sn.S1)a1 + (Sn.S2)a2 +(Sn.S3)a3+…..(Sn.Sn)an = class to which Sn belongs (n)

4. The hyperplane which separates the malwares from benign is given by

$$W=\sum a_i S_i$$

AdaBoost Algorithm

1. Select the best classifier i.e. one which has least error.
   $E_c = \sum w_i$, where i belongs to those samples that are misclassified by the classifier "c" and 'w' is initial weights assigned to each example
2. Determine a_c.
   $a_c= 0.5 * (\ln ((1-E_c)/E_c))$
3. Update weights.
   $w = \{0.5 * (w_i / (1-E_c))\}$, where "i" belongs to those instances which are correctly classified by the classifier "c".
   $w = \{0.5 * (w_i / (E_c)) \}$, where "i" belongs to those instances which are incorrectly classified by the classifier "c".
4. Continue steps 1, 2, 3 till we are left with no classifiers or we achieve 100 % accurate results.
5. Now prediction for an unknown instance x made by adaboost is given by
   $$H(x) = \operatorname{sign}(a_1 H_1(x) + a_2 H_2(x) +\ldots+a_n H_n(x))$$

Where a_i is the parameter corresponding to i^{th} selected classifier and $H_i(x)$ is the prediction made by the i^{th} selected classifier.

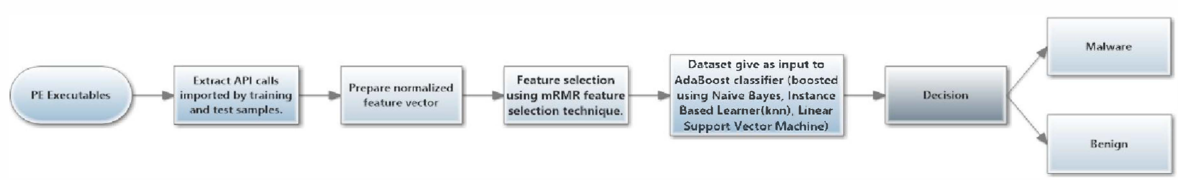

Fig 1: Flow of the analysis procedure

3.4 Steps of Methodology

Step 1: Malware samples can be downloaded from VX Heavens and other agencies. Benign executables can be found in System32 folder of our operating system (Windows7, Windows XP etc).

API calls: A PE contain, an array of data structures, one per imported DLL and points to an array of function pointers. The function pointers array is called as import address table (IAT). Each imported API occupies a reserved place in IAT where the imported function address is written by the windows loader. In unreachable code, attackers may use irrelevant API calls. One has to go for runtime API call detection i.e. executable is to be run in an virtual environment, then all calls made by executable's are captured, and then these executable patterns are mine to extract important information about executables behavior, to overcome this problem. We extracted API calls invoked by executable samples.

Step 2: Represent each sample as a binary vector of 37 API calls prominent in benign executables such that attribute A_i = 1 if API call 'i' is imported by the executable, else A_i = 0.

Step 3: Then the next step is to select the prominent and relevant features among the extracted features. Those which play an important role in classifying the executable as benign or malicious, are prominent features. Minimum Redundancy and Maximum Relevance (mRMR) [7] is the feature algorithm that is used. Using mRMR, by identifying highly correlated features with target class and selecting features dissimilar to other features but strong enough to identify a target class, prominent and effective features are selected. Mutual information (MI) is the measure used for measuring the degree of correlation amongst different variables, in case of mRMR. Maximum Relevance is to search features with large dependency on target class. It is likely that features selected through maximum relevance could have high redundancy i.e. the dependency among them is large.

Step 4: Feature vectors obtained after step 3 form our training set. Steps 1, 2 3 are done for the supplied test set also. Now as input to classification algorithms Naïve Bayes, Instance Based Learner (IBK), and Linear Support Vector Machine, the training and supplied test set are given. The prediction given by above classifiers is used as inputs to AdaBoost classifier which predict the final class of the executable.Experiment results are evaluated using TPR, FPR, TNR, FNR, Accuracy (ACC), Precision, F-Measure, and ROC Area. TPR is true positive rate, and so are the true negative, false positive, and false negative rates.

4. EXPERIMENT RESULTS

Experiment: Using Prominent API calls imported by Benign Files, as Features

The training dataset included 635 samples (400 malicious and 235 benign) and the test set included 90 malicious samples and 89 benign. Prominent 37 api calls present in benign files were taken as features and API calls were extracted from the samples dynamically. Using samples in training set and recorded in one file and those using samples in test set in another, feature vectors were formed. To reduce dimensions, mRMR feature selection technique was used. As input to naïve bayes, knn, the resultant feature vector files were given and linear svm classifiers which prepared a model based on the training set and then predicted the class for the test set. Then as input to AdaBoost classifier which predicted the class for the executable, the results obtained by the above three classifiers were given. A better classification accuracy of 94.9% was obtained using AdaBoost and Linear SVM (refer Fig. 1). By AdaBoost and Linear SVM, a maximum recall of 0.977 was obtained (refer Figure 3).

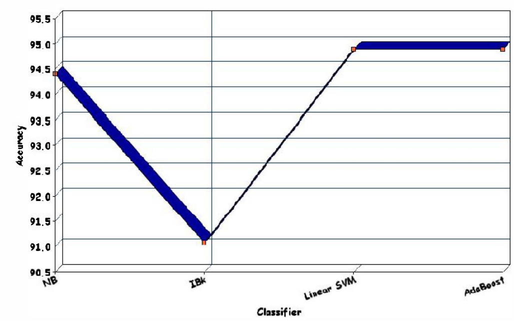

Fig 2: Accuracy obtained by different classifiers

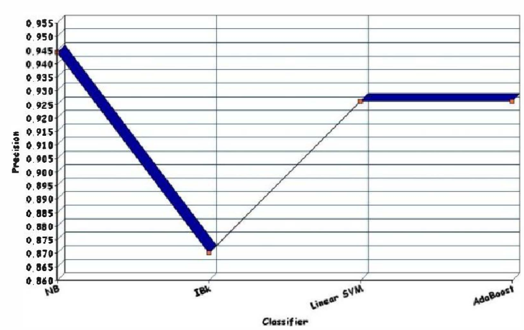

Fig 3: Precision obtained for different classifiers

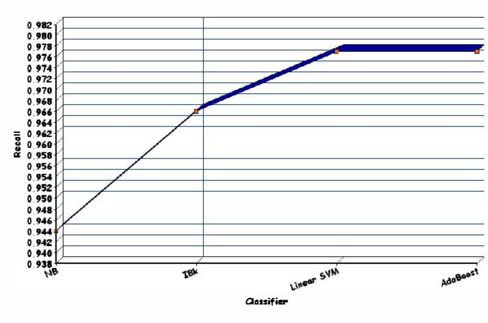

Fig 4: Recall obtained for different classifiers

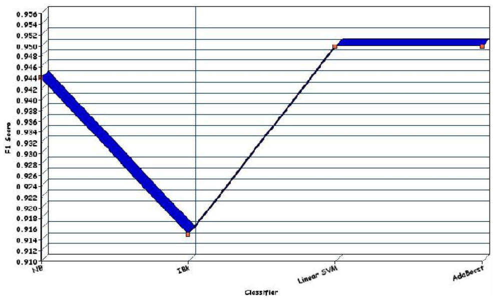

Fig 5: F1 Score obtained for different classifiers

5. CONCLUSION

We see that, to classify program executables into malware and benign, API calls imported by executables while running can be used. Using AdaBoost and Linear SVM classifiers, a classification accuracy of 94.9% was achieved. Also it is seen that boosting technique provides classification accuracy better than individual classifiers and give comparable results with Linear SVM.

REFERENCES

[1] Willems, Carsten, Thorsten Holz, and Felix Freiling. "Toward automated dynamic malware analysis using cwsandbox." *IEEE Security & Privacy* 2 (2007): 32-39.

[2] Moser, Andreas, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection." *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007.

[3] Zolkipli, Mohamad Fadli, and Aman Jantan. "An approach for malware behavior identification and classification." *Computer Research and Development (ICCRD), 2011 3rd International Conference on*. Vol. 1. IEEE, 2011.

[4] Nataraj, Lakshmanan, et al. "A comparative assessment of malware classification using binary texture analysis and dynamic analysis." *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011.

[5] Anderson, Blake, Curtis Storlie, and Terran Lane. "Improving malware classification: bridging the static/dynamic gap." *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012.

[6] Santos, Igor, et al. "Opem: A static-dynamic approach for machine-learning-based malware detection." *International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*. Springer Berlin Heidelberg, 2013.

[7] Peng, Hanchuan, Fuhui Long, and Chris Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*27.8 (2005): 1226-1238.

[8] Kang, Brent Byung Hoon, and Anurag Srivastava. "Dynamic Malware Analysis."*Encyclopedia of Cryptography and Security*. Springer US, 2011. 367-368.

[9] Egele, Manuel, et al. "A survey on automated dynamic malware-analysis techniques and tools." *ACM Computing Surveys (CSUR)* 44.2 (2012): 6.

[10] Bayer, Ulrich, Engin Kirda, and Christopher Kruegel. "Improving the efficiency of dynamic malware analysis." *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010.

[11] Bayer, Ulrich, et al. "Dynamic analysis of malicious code." *Journal in Computer Virology* 2.1 (2006): 67-77.

[12] Firdausi, Ivan, et al. "Analysis of machine learning techniques used in behavior-based malware detection." *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*. IEEE, 2010.