Detecting Unknown Anomalous Program Behavior Using API System Calls

Md. Rafiqul Islam, Md. Saiful Islam* and Morshed U. Chowdhury

Deakin University, Melbourne, Australia

*Swinburne University of Technology, Melbourne Australia

rafiqul.islam@deakin.edu.au, mdsaiifulislam@swin.edu.au and
morshed.chowdhury@deakin.edu.au

Abstract. This paper presents the detection techniques of anomalous programs based on the analysis of their system call traces. We collect the API calls for the tested executable programs from Microsoft detour system and extract the features for our classification task using the previously established n -gram technique. We propose three different feature extraction approaches in this paper. These are frequency-based, time-based and a hybrid approach which actually combines the first two approaches. We use the well-known classifier algorithms in our experiments using WEKA interface to classify the malicious programs from the benign programs. Our empirical evidence demonstrates that the proposed feature extraction approaches can detect malicious programs over 88% which is quite promising for the contemporary similar research.

Keywords: Malicious Program, API System Calls, Classification.

1 Introduction

The Internet has become an integral part of our everyday life and its usage is expected to continue growing. However, its rapid adoption has also left it susceptible for misuse and abuse. Along with the growth of the Internet, there has been a dramatic growth of Internet threats in recent years [1]. It is a major concern that despite the increasing development of Internet security services, such as AV (anti-virus), AS (anti-spam) etc., and technologies, the rate of increasing volume of malicious software samples thousands per day with the development of malicious code writing techniques. According to the report from king soft Lab, it is usually larger than 10,000 per day [2]. Therefore, effective and efficient automated malicious code detection technique is most important research concern nowadays.

Recently, there has been an increase in attacks targeting different controlling systems in their critical infrastructures like SCADA (Supervisory Control and Data Acquisition) [1-3]. There has been a recent alarming report of an Internet based attack on a nuclear power plant (Davis-Besse) in Oak Harbor (Ohio, USA), which was attacked by the Slammer SQL server worm and disabled a safety monitoring system of the Nuclear Power plant for nearly five hours [3]. There has been great

concern that application programs running on controlling systems can be compromised due to intrusions and attacks from the Internet. A change in the controlling program behavior can result in the corresponding change of its ability to control field devices and timely respond to legitimate queries from operators.

In this paper, we propose novel techniques for feature extraction to detect malicious behavior of the programs by exploiting variations in frequency and duration of system calls in the programs. We also aggregate the features which can help to identify potential intrusions or attacks. We use three different approaches to identify the anomaly of the programs and our empirical evidence shows that our investigation is significant in particular the hybrid approach to detect the malicious programs.

The rest of the paper is organised as follows: Section 2 describes the background and related work, Section 3 describes the real time identification of anomalous program behavior, Section 4 describes the empirical performances and finally, Section 5 concludes the paper.

2 Background and Related Work

A computer program is a sequence of instructions stored on disk in order to perform some specific tasks. Executing a sequence of instructions in arbitrary ways which differ from intended behavior can have significantly harmful effect on the computer host. One approach to identify a program's abnormal behavior is by profiling its behavior under normal operational conditions and identifying any deviations from this profile.

There have been many proposed approaches [6-9] in intrusion detection systems based on the program behavior. Forrest et al. [7] proposed a novel intrusion detection system based on program behavior that was inspired by the human immune system. Their hypothesis is that patterns of normal program behavior can be generated from short sequences of system calls. Experimentally, they store the system calls of each individual program's normal behaviors in independent database. In order to flag it as normal or abnormal they match the current behavior for each program against this database. Kosoresow et al. [8] use fixed-length frames to the Forrest approach. They analyse the structure of system calls where they establish recurrent patterns in each execution trace for any given program. They model program behavior using a Deterministic Finite Automaton (DFA) by manually selecting the macros that matched the patterns which are believed to represent the normal behavior. Once this is done the anomalies can be detected by applying these macros to the current behavior. Sekar et al. [10] used similar state machine models, however, rather than detecting normal behavior they detect the violations of specified behaviors. In this approach each program requires a specification model for its intended or acceptable behavior. Specification-based approaches have also been used in other intrusion detection systems for example, an auditing specification language [10-12] is used to model the expected/allowed program behavior under certain policies or models. Therefore, any behavior that violates the desired model or policy is deemed to be an attack or compromise.

All of these techniques suffer by the fact that any small change in program behavior will not be matched by the database and therefore, will be flagged as potentially malicious program. Over the last decade, researchers have been trying hard to identify the malicious behavior of the programs by adopting a diversity of solutions. Many research efforts have been conducted on developing automatic malware (virus, Trojan horse, worms etc.) detection technique using different data mining techniques [13 - 16]. All of these techniques are consists of two basic modules: (a) method of feature extraction and (b) method of classification.

Our approach is to create a database of rules for the normal operation or behavior of a program. After creating this database we will design efficient techniques to match runtime program behavior against this database. Any large variation should indicate a possible altered pattern of program execution and should be flagged for security audits.

3 Identification of Anomalous Program Behavior

We propose a three-step process in detecting unknown program behavior. First, we analyse the system calls running under normal conditions and populate a database with these system calls. Second, from these database entries of system calls we create a general model of the normal behavior. This step is important since few specific rules may trigger many false positives when the program is run under varying conditions. Finally, we observe the sequential and temporal changes in the runtime behavior (using system calls) of the program in order to detect any anomalous or abnormal behavior.

3.1 Analysing System Calls

Intrusion Detections Systems (IDS) normally employ two types of profiling systems; user behavior profiling, and program behavior profiling. For example, Lane and Brodley [11] used a technique to profile the normal user behavior in order to distinguish between normal and anomalous user behavior. Endler et al. [12] uses neural networks to learn user behavior based on the Solaris Basic Security Module (BSM). However, a common problem with these approaches is that an intruder can easily get around user behavior profiling by slowly changing his/her behavior from the normal model. On the other hand, since a system is normally compromised by a program, analysis of program behavior reveals attacks or anomalous behavior much more succinctly than user behavior. Furthermore, in many sensors and actuators devices user based login similar to regular Operating Systems (OS) is not available. Consequently, a profiling system based on program behavior is more robust and appropriate in our case than using user based profiling. The first step of analysing the system calls is to collect information on the system calls (e.g., read, write, open, close, etc.) generated by a executable as it is running, and store them in a database along with the timestamps of the events. In view of the above, we will develop a strategy to efficiently collect and store system calls from executables for subsequent analysis and modelling of normal behavior.

3.2 Extracting the Features Using N-gram

Many researchers propose n-gram based approaches of program behavior for anomaly detection [18-21]. In these approaches, n -gram information is extracted from executable files. In the n -gram technique, we collect the contents of a window of width n as it slides along the trace is called the *sliding window* algorithm. For example, consider a trace of training data, where we use letter symbols to stand for system calls:

A B C C A B B C C A A B C

If $n = 4$, the self database consists of the set of all substrings of length four of the training string:

ABCC
BCCA
CCAB
CABB
ABBC
BBCC
CCAA
CAAB
AABC

After the database is constructed, we can check executions of the program against the self database. An *anomaly* is an n -gram that occurs during execution but is not in the database. For example, the string ABCCABBCCABABC contains the anomalies CABA, ABAB, and BABC. A few anomalies may simply mean that the training traces did not cover all normal behavior, but a large cluster of anomalies is a good indicator of an attack.

In our testing, we use a sliding window and generate short sequences of system calls between 1 to n . These short sequences we called as n -grams. Then, all nodes between *root* and a node at level i , represents an n -gram of order i , as shown in Fig. 1. Since the common prefixes are merged, it forms an efficient storage structure.

Fig. 1. N-gram-tree of order 3 built with a training trace.

For extracting the features, we monitor the log of each executable file and count the number of times for a given features of n -gram sequence and create the vector based on the presence and absence of the selected n -grams. We select the value of $n=4$ (as it is considered as standard) and we use information gain (IG) technique, a well known technique to select the most informative features. The IG quantifies the reduction in uncertainty of the class attribute based on the value of other input attributes.

3.3 Creating the Training and Test Datasets

After collecting the system call traces of normal program behavior described in Section 3.2, we create a training dataset by labelling the short sequences (l) of system calls derived from the normal trace files and labelling them *normal*. More specifically, the training dataset contains records with l attributes each referring to a particular system call (read, write, open, close, etc.) in some order derived from the normal database, hence the class attribute has the *normal* label. Since, a rule learning algorithm benefits from a mix of both positive and negative examples, we also add some records whose system calls do not appear in the normal database and label the class as *abnormal*. There are two ways of generating abnormal rules, we can randomly generate it (and test that it is not in the normal database) or use the system calls from a malicious program. We use both of these techniques to generate abnormal test cases. The test dataset is similar to the training dataset (i.e., it contains both normal and abnormal records) except that the records between the two databases are mutually exclusive (so that the model is more robust to unknown incidents).

3.4 Sequential and Temporal Changes in System Calls

One of the most difficult challenges of analysing program behavior is how to detect changes in the system calls which may indicate possible anomaly. Existing approaches [6-13] employ different matching techniques between sequences of system calls of a running program against the database of sequences of system calls for the same program under normal conditions. The main idea is to store the normal program call sequences in a database and count the mismatches during testing. If the number of mismatches exceeds a tolerable *threshold* then a flag is raised. We have identified two important problems of anomaly detection that are not addressed by existing techniques: (a) the effect of system call frequency due to attacks (such as DoS and DDoS), and (b) the effect of elapsed time variation of system calls due to unwanted programs running (such as a worm or a virus). Next, we consider each of these techniques and discuss our proposed solutions.

3.4.1 Frequency Based Detection

In this approach, we look at the frequency of appearance of a particular sequence of calls within a program. If this value is quite high then this might indicate a

potentially malicious attempt to deny legitimate services. Let $S = \{C_1, C_2, \dots, C_n\}$ represent the sequence of system calls. Existing techniques [6-13] detect a problem if the order of calls appears differently during program execution. For example, $S = \{C_1, C_2, \dots, C_n\}$ would indicate a possible change in behavior. However, these approaches ignore the frequency of calls when the order is correct. In our approach, we note the order as well as the frequency of the system calls in the normal database in order to identify anomalies occurring from malicious programs or attacks. For example, $S = \{C_1, C_2, C_2, C_2, \dots, C_n\}$ should be flagged as an anomalous behavior since system call C_2 has occurred three times more than one call in the normal trace. Consequently, our approach will design a mechanism to identify large variations in frequency of system calls from the normal model.

3.4.2 Time Based Detection

Some traditional control system runs specialized applications (for example, Human Machine Interface (HMI) operator console just runs HMI client application, Remote Terminal Unit (RTU) only runs services that communicate with field devices and respond to Master Terminal Unit (MTU), etc.), there is little variation in the execution time of a sequence of calls unlike a traditional PC which is multi-tasking. As a result, system calls of any program may get intertwined with another program's system calls. For example, let $T_s = \{t_1, t_2, \dots, t_n\}$ represent the interval between consecutive system calls from the normal system call trace. If an unwanted program (a virus or a worm) is executed in the same machine, then the runtime behavior will change and can be easily detected. For example, in $T_s = \{t_1, t_r, t_2, \dots, t_n\}$ the presence of an unwanted system call has caused the second legitimate t_2 to happen after t_r seconds, which is clearly an anomaly. Consequently, if we can detect large variation in the elapsed time of a sequence of system calls within the system then this might indicate the presence of a rogue or unwanted program like worms or viruses.

3.4.2 Hybrid System

We combine the above two techniques, described in section 3.4.1 and 3.4.2, which are frequency and time based detection techniques, we call it hybrid system. In this technique, we combine both extracted feature vectors in serialize fashions. We then pass this information to train the classifier. The main focus of this technique is to enhance the detection rate with minimize false positive alarms.

4 Evaluation

In our experiment, we use a small data set which consists of 50 cleanware and 50 malicious executables. The malicious executables contain Backdoors, Flooders, Rootkits, Worms and Viruses, collected from VX Heavens repository (<http://vx.netus.org>) and the collected cleanware are win32 based. We extract system

call sequences by running them on VM in Win32 environment using the Microsoft detour technique. In our testing, we use the following three different tests:

- (a). T-1 : Frequency based detection
- (b). T-2: Time based detection and
- (c). T-3 : Hybrid technique

In our classification process, we use WEKA interface to build the training model of the classifier and evaluate the test data. Fig. 2 summarizes the comparative analysis of the confusion matrix, that is, the output of FP (false positive), FN (false negative), TP(true positive) and TN(true negative), of the three tests. It is clear from the graph presented in Fig. 2 that the test T-3 shows the best performance among the three, followed by T-2 and T-1 is the worst. But none gives zero FP and FN. It is also evident that the percentage of FN is higher than the percentage of FP which affects the accuracy of the classifier.

Fig. 3. Comparison of precision, recall, accuracy and error of the three tests.

Fig. 3 shows the comparative analysis of precision, recall, accuracy and errors of our three experiments. It is evident again that T-3 performs better compared to the other two, but the performance of T-2 and T-1 is closer in terms of recall (which are ~79.8% and ~80.0%), accuracy (which are ~85.8% and 84.7%) and error (which are 14.2% and ~15.3%) but varies in precision (which are ~96.7% and ~92.3%). The difference in performance is mainly due to FPs and FNs that makes a significant gap between precision and recall for all the testing classifiers.

Table 1 shows the average performance of our experiment using Receiver Operating Characteristic (ROC) report. We use four important measurement values in this ROC report, AUC (Area Under Curve) estimation values, AUC estimated standard error (StdErr), a 95% of Confidence Interval (CI) for both lower and upper limit, and 1-sided Probability-values (P-value).

The AUC is a popular measure of the accuracy of an experiment. All things being equal, the larger the AUC, the better the experiment is at predicting by the existence of the classification. The possible values of AUC range from 0.5 (no diagnostic ability) to 1.0 (perfect diagnostic ability). The CI option specifies the value of alpha to be used in all CIs. The quantity (1-Alpha) is the confidence coefficient (or confidence level) of all CIs. The P-value represents the hypotheses tests for each of the criterion variables.

Fig. 3. Comparison of precision, recall, accuracy and error of the three tests.

Table 1. The ROC report of our three tests.

ROC Estimation	T-1	T-2	T-3
AUC	0.83594	0.85156	0.90885
AUC (StdErr)	0.04031	0.03894	0.03245
95% of CI	0.68147 0.91910	0.69835 0.93016	0.76047 0.96704
P-value	(<0.001)	(<0.001)	(<0.001)

Obviously, a useful experiment should have a cut-off value at which the true positive rate is high and the false positive rate is low. In fact, a near-perfect classification would have an ROC curve that is almost vertical from (0, 0) to (0, 1) and then horizontal to (1, 1). The diagonal line serves as a reference line since it is the ROC curve of experiment that is useless in determining the classification.

Table 2 present the cost-benefit analysis for ROC report. The cost-benefit ratio is the ratio of the net cost when the condition is absent to the net cost when it is present. In our experiment, we select the cut-off value for which the computed cost value is

maximized (or minimized) and the prevalence is considered ($=0.00001$), which is the actual probability of the condition in the population.

Fig. 4 shows the ROC curves of our experiment. A ROC curve shows the characteristics of our experiment by graphing the false positive rate (1-specificity) on the horizontal axis and the true positive rate (sensitivity) on the vertical axis for various cut-off values. Each point on the ROC curve represents a different cut-off value. Cut-off values that result in low false positive rates tend to result low true positive rates as well. As the true positive rate increases, so does the false positive rate. It is clear from Fig. 4 that the T-3 technique shows the best ROC curve, as indicated in our previous experiments.

	Cuto ff Valu e	Sens.	Spec.	Cost-Benefit when ratio=0.5	Cost-Benefit when ratio=0.7000	Cost-Benefit when ratio=0.8000	Cost-Benefit when ratio=0.9000
T-1	-1.000	1.0000	0.0000	-49998.5000	-69998.3000	-79998.2000	-89998.1000
	+1.000	0.9219	0.7500	-12498.9531	-17498.9031	-19998.8781	-22498.8531
T-2	-1.000	1.0000	0.0000	-49998.5000	-69998.3000	-79998.2000	-89998.1000
	+1.000	0.9531	0.7500	-12498.9219	-17498.8719	-19998.8469	-22498.8219
T-3	-1.000	1.0000	0.0000	-49998.5000	-69998.3000	-79998.2000	-89998.1000
	+1.000	0.9844	0.8333	-8332.2656	-11665.5656	-13332.2156	-14998.8656

Table 2. Cost benefit analysis with prevalence $=0.00001$ for T-1, T-2 and T-3.

(a)

(b)

(c)

Fig. 4. The average ROC curve for three tests T-1, T-2 and T-3.

5 Conclusion

In this paper, we propose an anomaly detection technique using program behavior based on tracing API system calls. We develop three different feature extraction approaches namely, frequency-based, time-based and a hybrid strategy using n -gram technique. It is demonstrated that the hybrid approach shows better performance compared to the other two proposed approaches. The time based detection outperforms the frequency based detection in terms of both accuracy and false positive alarms. In our test, we use small data set; however, in our future work we plan to use large data set with various malicious programs in real time environment.

References

1. Cárdenas, A. A., Amin, S., Sastry, S.: Research Challenges for the Security of Control Systems. In: Proc. of the 3rd USENIX workshop on Hot Topics in Security, Associated with the 17th USENIX Security Symposium. San Jose, CA, USA (2008)
2. Protecting Critical Infrastructure SCADA Network Security Monitoring. In: Tenable Network Security Inc. (2008)
3. Andy, G.: America's Hackable Backbone. Forbes (2007)
4. Ye, Y., Wang, D., Li, T., T. Ye, D., Jiang, Q.: An Intelligent PE-Malware Detection System Based on Association Mining. J. Comput. Virol., vol. 4, pp. 323–334 (2008)
5. Symantec Internet Security Threat Report Trends for 2008. Vol. xiv (2009)
6. Forrest, S., Hofmeyr, S. A., Somayaji, A.: The Evolution of System-Call Monitoring. In: Proc. of the 24th Annual Computer Security Applications Conference, pp. 418-430 (2008)
7. Forrest, S., Nguyen, T. V., Weimer, W., Goues, C. L.: A Genetic Programming Approach to Automated Software Repair. In: Proc. of the Genetic and Evolutionary Computation Conference, pp. 947-954 (2009)
8. Kosoresow, A. P., Hofmeyr, S. A.: Intrusion Detection via System Call Traces. In: IEEE Software, vol. 14, no. 5, pp. 35-42 (1997)
9. Hofmeyr, S., Forrest, S., Somayaji, A.: Intrusion Detection Using Sequences of System Calls. In: Journal of Computer Security, vol. 6, pp. 151-180 (1998)
10. Sekar, R., Uppuluri, P.: Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications. In: Proc. of the 8th Conference on USENIX Security Symposium (1999)
11. Lane, T., Brodley, C. E.: Data Reduction Techniques for Instance-Based Learning of Human/Computer Interface Data. In: Proc. of the 17th International Conference on Machine Learning, pp.519-526 (2000)
12. Endler, D.: Intrusion Detection Applying Machine Learning to Solaris Audit Data, In: Proc. of the 14th Annual Computer Security Applications Conference, pp.268 (1998)
13. Lee, T., Mody, J. J.: Behavioral Classification. In: Proc. of EICAR Conference (2006)
14. Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., and Nazario, J.: Automated Classification and Analysis of Internet Malware. In: Proc. of 10th International conference on Recent advances in Intrusion Detection, pp. 178–197(2007)
15. Jiang, X., Zhu, X.: vEye: Behavioral Footprinting for Self-Propagating Worm Detection and Profiling. In: Knowl. Inf. Syst., vol. 18, no. 2, pp. 231–262(2009)
16. Sung, A., Xu, J., Chavez, P., Mukkamala, S.: Static Analyzer of Vicious Executables (SAVE). In: Proc. Of the 20th Annual Computer Security Application Conference, pp. 326–334 (2004)

17. Ye, Y., Wang, D., Li, T., Ye, D.: IMDS: Intelligent Malware Detection System. In: Proc. ACM Int. Conf. Knowl. Discovery and Data Mining, pp. 1043–1047(2007)
18. Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.: N-Gram-Based Detection of New Malicious Code. In: Proc. of the 28th Annual International Computer Software and Applications Conference-Workshops and Fast Abstracts, IEEE Computer Society, Washington, DC, USA, vol. 2, pp. 41-42(2004)
19. Ahmed, F., Hameed, H., Shafiq, M. Z., Farooq, M.: Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection. In: Proc. of the 2nd ACM Workshop on Security and Artificial Intelligence, ACM, New York, NY, USA, pp. 55-62 (2009)
20. Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown Malcode Detection via Text Categorization and the Imbalance Problem, In: IEEE International Conference on Intelligence and Security Informatics, pp. 156 -161(2008)
21. Park, Y., Reeves, D., Mulukutla, V., Sundaravel, B.: Fast Malware Classification by Automated Behavioral Graph Matching, In: Proc. of the 6th Annual Workshop on Cyber Security and Information Intelligence Research, ACM, New York, NY, USA, pp. 1-4 (2010)