2015 IEEE 39th Annual International Computers, Software & Applications Conference

Malware Detection System Based on API Log Data Mining

Chun-I Fan
Department of
Computer Science and
Engineering
National Sun Yat-sen University
Kaohsiung 804, Taiwan
Email: cifan@faculty.nsysu.edu.tw

Han-Wei Hsiao
Department of
Information Management
National University of Kaohsiung
Kaohsiung, Taiwan
Email: hanwei@nuk.edu.tw

Chun-Han Chou and Yi-Fan Tseng
Department of
Computer Science and
Engineering
National Sun Yat-sen University
Kaohsiung 804, Taiwan
Email: gx1394@gmail.com
Email: yftseng1989@gmail.com

***Abstract*—As information technology improves, the Internet is involved in every area in our daily life. When the mobile devices and cloud computing technology start to play important parts of our life, they have become more susceptible to attacks. In recent years, phishing and malicious websites have increasingly become serious problems in the field of network security. Attackers use many approaches to implant malware into target hosts in order to steal significant data and cause substantial damage. The growth of malware has been very rapid, and the purpose has changed from destruction to penetration. The signatures of malware have become more difficult to detect. In addition to static signatures, malware also tries to conceal dynamic signatures from anti-virus inspection. In this research, we use hooking techniques to trace the dynamic signatures that malware tries to hide. We then compare the behavioural differences between malware and benign programs by using data mining techniques in order to identify the malware. The experimental results show that our detection rate reaches 95% with only 80 attributes. This means that our method can achieve a high detection rate with low complexity.**

I. Introduction

In modern society, the rapid development of information technology is closely combined with everyday life. Due to the popularity of mobile devices and cloud computing, there are numerous resources available on the Internet including computing power and sensitive data. These resources have become targets of attackers, and malware is the primary tool used in the attacks. Attackers adopt multiple approaches to phishing and penetration, and detecting these customized attacks is difficult for users. A security report [1] from Trend Micro: TrendLabs2012 ANNUAL SECURITY ROUNDUP Evolved Threats in a ”Post-PC” World points out that targeted attacks have become more popular and that attackers are experts in customizing such attacks. The Spear-phishing mails are still the main approach in the delivery of targeted attacks.

Malware no longer aims to only destroy systems but also focuses on penetration and stealth. Owing to the growing amount of malware and counter-analysis techniques, the traditional signature-based detection is inadequate for the analysis of such malware. Statistics from AV-TEST [2] show that the total number of malwares exceeded 130 million in 2013 and continues growing.

There are two types of malware analysis: static analysis and dynamic analysis. Static analysis disassembles malware binary files and analyzes the assembly codes. Dynamic analysis executes the malware binary files and analyzes the behaviors that malware performs. Static analysis can achieve high analysis speed, but automatic static analysis is easy to be thwarted by packing and encryption techniques. Although there are also many stealth techniques for countering dynamic analysis, we believe that dynamic analysis can quarry more features than static analysis. We think that we can distinguish the differences between malwares and benign programs by analyzing features of dynamic behaviors.

Some reports [3] [4] point out that APT attacks widely use RATs (Remote Access Tool or Remote Access Trojan) to remotely control victims after breaching. An example of this kind of tools is Poison IVY. In our analysis of Poison IVY, it performs remote control and also tries to conceal its behaviors in order to become stealth in victims. Malwares not only perform malicious behaviors but also attempt to conceal behaviors. We believe that dynamic feature analysis and behavior tracing of malwares is an important issue, and also an effective method in malware detection.

In this research, we take API call (System Call) as dynamic features for analysis. First we build a virtual environment to monitor behaviors. In addition to the API monitoring agent, we also develop a tracing module in virtual environment in order to trace the behaviors that malware attempts to conceal. For these behavior records, we adopt data mining tools to analyze and build a description model. This description model can be used to identify malwares and benign programs. Since there are thousands of features in the analysis procedure, we apply an attribute selection technique to reduce the number of features and also reduce the complexity. In our experiment, we collect 773 malwares and 253 benign programs as the dataset. The detection module can identify 95% of malwares and benign programs by using only 80 attributes. Our method can achieve high detection rate with high efficiency.

II. Related Works

There are many kinds of researches for detecting malwares. Some systems use network-based detection to detect bot or spyware by analyzing the network traffic. In this research, we utilize the system behavior for analysis. Therefore, we will focus on the host-based detection.

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.241

Host Based Detection
The host based detection has three ways to detect malware: malware signature, malware binary code, and malware dynamic behavior. Malware signature detection is employed by the tradition anti-virus systems but it is easy to be evaded.

Some researches analyzed the malware binary codes by static analysis [5] [6] [7] [8] [9]. They extracted the API names and strings from malicious PE executables and then analyzed these information with algorithms in order to find out the differences between malicious and benign programs. Analyzing malware by static analysis can achieve very high detection speed but the packing technique makes automatic static analysis more difficult.

In Wang et al. [10], they employed static analysis to extract the API names from the PE executables and build an API call sequence. They compared the API call sequence with a suspicious behavior database which they have defined and the Bayesian algorithm was applied to compare the differences and then identify malwares. Except for the API information, Islam et al. [11] also utilized the printable string as a feature. They analyzed the function length frequency and printable string information by classification engines and their solution achieves very high detection rate.

The dynamic behavior detection [12] [13] [14] [15] [16] analyzes the behavior differences between malware and benign programs. This method needs to construct a secure environment for malware execution. After execution, these behavior records of each malware will be analyzed so as to compare benign and malicious programs. Behavior analysis can extract more features about malwares than static analysis, but it spends more time than automatic static analysis because static analysis does not need to execute malwares.

Kolbitsch et al. [17] proposed a method to monitor API and find the relationship of API calls by analyzing the memory addresses of API parameters. They constructed a behavior graph of API records for malwares and detect malwares by comparing the behavior graph. Tian et al. [18] and Salehi et al. [19] also monitored the API calls, but they applied data mining tools to classify API records in order to compare malicious and benign programs. Trinius et al. [20] proposed the concept of MIST. They divided API functions into different categories and encoded each function with a number. Each code represents an API function. Then they calculated the similarity of malwares and benign programs by applying data mining engine to analyze these functions and parameters.

III. The Proposed Method

In this research, we identify benign and malicious programs by tracing and analyzing their behaviors. We believe that API calls (System Calls) reflect the behavior of a program the most, so we regard them as features for analysis. However, malwares would attempt to obscure from inspection. Therefore, we also trace some stealth techniques that malwares regularly adopt in order to trace the behaviors that malwares attempt to conceal.

Figure 1 shows the architecture of our detection system. In our system, sample programs will be launched in virtual machines and their behaviors (API calls) will be recorded.

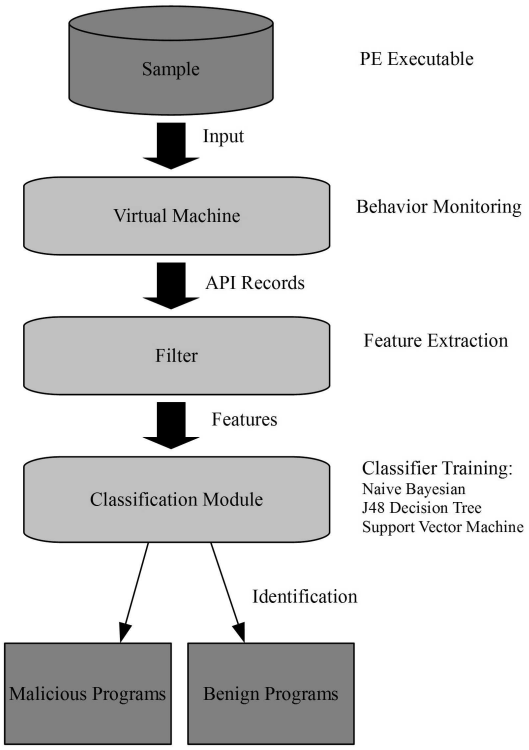

Fig. 1. System Architecture

After acquiring records, we extract important features from these records so as to reduce the analysis complexity and improve accuracy. Then these features will be analyzed by classification tools in order to compare the differences between malware and benign programs.

A. API Record Collection

We construct a virtual environment for monitoring API and deploy API Monitor v2 and Malpimp as our monitoring agents. These two monitoring tools are able to monitor not only regular APIs but also Native APIs i.e. undocumented APIs or low-level APIs. Instead of using regular APIs, many malwares utilize native API so as to evade from inspection.

There are thousands of APIs that programs might use, but we do not monitor all the APIs due to the accuracy and complexity. Most of APIs are less important, so we only select 6 DLLs to monitor. They are user32.dll, kernel32.dll, advapi32.dll, ntdll.dll, ws2_32.dll, and wininet.dll. These DLLs provide many important functionalities such as process management, network access, and graphic interface.

Stealth Techniques
Recently, malwares often employ stealth techniques to obscure from users and antivirus. According to our observation, malawres often hijack system processes to make itself undetectable. Here are two approaches we want to trace in the monitoring procedure: 1) New process creation. Malwares often create a new process to perform malicious behaviors. It

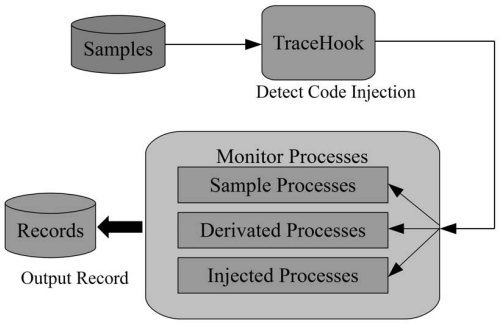

Fig. 2. Monitor Procedure

is hard to find malicious behaviors from the original process because malicious behaviors are performed by the new created process. 2) Code injection. Malwares apply this technique to inject malicious codes into existing processes (usually system processes) and force these (injected) processes to perform malicious behaviors.

Monitor and Trace Module
We have to monitor not only sample processes but also derivative processes and injected processes because malwares might behave with stealth techniques. For each sample program, we will test if it is on code injection behavior before monitoring its APIs. We develop a tracing program **TraceHook** to detect code injection behaviors by hooking techniques.

First of all, we run TraceHook to launch a sample and detect the code injection. TraceHook intercepts the CreateThread function and identifies the injected process by the parameters of this function. The injection information will be returned to the monitoring procedure. Then the procedure begins to monitor API with three aspects simultaneously: 1) The API monitor launches the sample and monitors its process. 2) Any new process which is created by the sample process will also be monitored. 3) According to the previous test, the monitoring procedure will monitor the processes that the sample process injects codes into. Figure 2 shows the monitoring procedure.

B. Feature Extraction

For each sample, we extract the calling frequency of each API as features. The following displays our extraction procedure. If we have three sample records:

- $sample_1 = \{API_1, API_2, API_3, API_2, API_3\}$
- $sample_2 = \{API_4, API_3, API_6, API_3\}$
- $sample_3 = \{API_2, API_4, API_5, API_4, API_4\}$

An array $APlist$ stores APIs that all malicious/benign programs have called:

$$APlist = \{API_1, API_2, API_3, API_4, API_5, API_6\}$$

We count the frequency of each API that each sample has called:

- $count_1 = \{API_1\text{: }1, API_2\text{: }2, API_3\text{: }2\}$ for $sample_1$
- $count_2 = \{API_3\text{: }2, API_4\text{: }1, API_6\text{: }1\}$ for $sample_2$
- $count_3 = \{API_2\text{: }1, API_4\text{: }3, API_5\text{: }1\}$ for $sample_3$

Then we cross-refer the $APlist$ and API frequency to construct a new table. This table will be used for classification:

	API_1	API_2	API_3	API_4	API_5	API_6
$sample_1$	1	2	2	0	0	0
$sample_2$	0	0	2	1	0	1
$sample_3$	0	1	0	3	1	0

C. Classification Module

Data mining is a technique for analyzing numerous data and digging out some useful but unknown information. Classification is one of the techniques in data mining and it uses a set of correct data for training and then builds a description model for the training data. Once new data come, the description model can be applied to identify if the new data are similar to the training data. Our research collects API records of malicious and benign programs and then takes the records as training data to build a description model. We can make use of this model to identify malicious and benign programs. We select **Naive Bayesian**, **Decision Tree (J48)**, and **Support Vector Machine** algorithms as our classification module. These three algorithms all belong to supervised learning. In the supervised learning, the training data must be labelled before training.

IV. Experiment and Evaluation

In this section we evaluate our detection module. We discuss the detection accuracy under different numbers of attributes and different algorithms. There are three algorithms selected for comparing malicious and benign programs: Naive Bayesian, J48 (Decision Tree), and Support Vector Machine.

A. Sample Program Collection

For the experiment, we collect about 7000 malicious programs from the Internet. These malicious samples are all PE executables and have been confirmed by VirusTotal. We also collect 251 benign programs as our benign samples. These benign samples are composed by Windows System programs and other widely-used applications.

We perform two experiments with different ratios of benign and malicious samples because the proportion of samples might affect the result. In the first experiment, we take the equal amount of malicious and benign samples for classification since this proportion is most reasonable. Therefore, we randomly select 263 malicious samples from the sample set and use all 251 benign programs for Experiment 1. In order to compare with other researches, we also choose the similar proportion of samples for Experiment 2. We use all 251 benign samples and randomly choose 773 malicious samples from the sample set. All of these benign and malicious samples have been verified to be launched successfully in our virtual environment. Table 1 lists the datasets of the two experiments.

TABLE I. TWO DATASETS FOR CLASSIFICATION

	Malicious samples	Benign samples
Experiment 1	263	251
Experiment 2	773	251

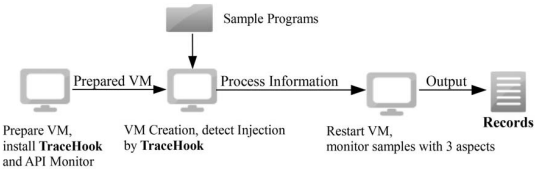

Fig. 3. Detailed Monitor Steps

B. API Monitoring

In monitoring samples, we construct a virtual environment by VirtualBox. We install Windows XP operating system and the tools that are used in monitoring. Then we save the status of the virtual machine with the snapshot function after construction.

For each sample, we first utilize our tracing tool TraceHook to launch it and detect the injection. The TraceHook returns the information of the injected process for later monitoring. Then the virtual machine will be restarted to monitor the API calling of the sample with three aspects: the sample process itself, derivative processes, and injected processes. Each sample will be executed for 2 minutes. Figure 3 displays the detailed steps.

C. Evaluation

We extract the features from the API records and form the features in the ARFF format for WEKA classification. WEKA is a data-mining tool and contains many algorithms. It provides functionalities of classification, clustering, and attribute selection. We choose Naive Bayesian, Support Vector Machine, and J48 (Decision Tree) as our classification algorithms.

We adopt K-fold Cross-Validation to evaluate the results. This validation randomly splits the input data into k parts, and then chooses one part of data as the experiment data and the other $(k-1)$ parts are the training data. These k parts of data would take turns becoming the experiment data and training data. As a result, there are k calculations and these k results will be averaged. We can get more reasonable result by cross-validation and we choose the standard 10-fold cross-validation for our experiment. The following measurements evaluate the experimental results and Table 2 is the Confusion Matrix:

- **True Positive:** The number of positive files that are identified as positive files.
- **False Positive:** The number of negative files that are identified as positive files.
- **True Negative:** The number of negative files that are identified as negative files.
- **False Negative:** The number of positive files that are identified as negative files.
- **True Positive Rate:** The probability that a positive file is identified as a positive file in an identification.
$$TPrate = \frac{TP}{TP+FN}$$

TABLE II. CONFUSION MATRIX

		Predict	
		Positive	Negative
Actual	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

TABLE III. RESULT OF EXPERIMENT 1

Algorithm	Type	TP rate	FP rate	Precision	Recall	F-Measure	Accuracy
J48	Malicious	0.962	0.055	0.948	0.962	0.955	
	Benign	0.962	0.055	0.948	0.962	0.955	
	Average	**0.954**	**0.047**	**0.954**	**0.954**	**0.954**	**0.954**
NaiveBayes	Malicious	0.962	0.043	0.958	0.962	0.96	
	Benign	0.957	0.038	0.961	0.957	0.959	
	Average	**0.959**	**0.041**	**0.959**	**0.959**	**0.959**	**0.959**
SVM	Malicious	0.76	0.125	0.862	0.76	0.808	
	Benign	0.875	0.24	0.78	0.875	0.824	
	Average	**0.817**	**0.182**	**0.822**	**0.817**	**0.816**	**0.816**

- **False Positive Rate:** The probability that a negative file is identified as a positive file in an identification.
$$FPrate = \frac{FP}{FP+TN}$$
- **Precision:** The probability that an identified positive file is really a positive file
$$Precision = \frac{TP}{TP+FP}$$
- **Recall:** This is the same as the TP rate
$$Recall = \frac{TP}{TP+FN}$$
- **F-Measure:** The harmonic mean of $Precision$ and $Recall$. This measure weights $Precision$ and $Recall$ performance evenly.
$$F-Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$
- **Accuracy:** The probability that a file is correctly identified.
$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

D. Experimental Result

Experiment 1

In Experiment 1, we use 251 benign and 263 malicious samples for training. The experimental result is listed in Table 3. This table shows that J48 and Naive Bayesian perform better. The accuracy of these two algorithms is up to 95% and the FP rate is about 4%. There are 10 malicious and 14 benign samples are misidentified in the J48 classification. Also, there are 10 malicious and 11 benign samples are misidentified in Naive Bayesian classification. Support Vector Machine performs worse: the accuracy has only 81% and it misidentifies 95 samples.

Experiment 2

Experiment 2 uses 251 benign and 773 malicious samples for classification. The performance is listed in Table 4. In Experiment 2, the performance is slightly increased in J48 and reduced in Naive Bayesian. There are 42 and 60 samples that are misidentified in two classifications. Also, SVM can identify 89% of samples with 109 misidentified samples.

TABLE IV. RESULT OF EXPERIMENT 2

Algorithm	Type	TP rate	FP rate	Precision	Recall	F-Measure	Accuracy
J48	Malicious	0.975	0.09	0.97	0.975	0.973	
	Benign	0.91	0.025	0.924	0.91	0.917	
	Average	**0.959**	**0.074**	**0.959**	**0.959**	**0.959**	**0.959**
NaiveBayes	Malicious	0.937	0.043	0.985	0.937	0.96	
	Benign	0.957	0.063	0.833	0.957	0.891	
	Average	**0.942**	**0.048**	**0.947**	**0.942**	**0.943**	**0.941**
SVM	Malicious	0.99	0.396	0.883	0.99	0.933	
	Benign	0.604	0.01	0.951	0.604	0.739	
	Average	**0.894**	**0.3**	**0.9**	**0.894**	**0.885**	**0.893**

E. Attribute Reduction

The amount of attributes reaches 1800 in the training procedure. Most of the attributes are less important so we attempt to minimize the attributes in order to increase performance. The characteristic of J48 is suitable for this experiment because J48 usually needs only tens of attributes for tree construction. We apply InfoGainAttributeEval to calculate the InfoGain value of each attribute and select different numbers of the attributes for J48 training.

TABLE V. ATTRIBUTE REDUCTION FOR EXPERIMENT 1

Attributes	TP rate	FP rate	Precision	Recall	F-Measure	Accuracy
1769	0.954	0.047	0.954	0.954	0.954	0.954
800	0.954	0.047	0.954	0.954	0.954	0.954
250	0.952	0.048	0.952	0.952	0.952	0.952
120	0.932	0.068	0.933	0.932	0.932	0.932
80	0.938	0.062	0.938	0.938	0.938	0.938
40	0.907	0.091	0.91	0.907	0.907	0.907
20	0.89	0.109	0.892	0.89	0.89	0.889

According to the InfoGain value, we choose the top 20, 40, 80, 250, and 800 attributes for training and then compare the results with those of the original 1800 attributes. Table 5 and Table 6 list the performance under different numbers of attributes. The results of the two experiments show that our detection system needs only about one hundred attributes to detect malwares and achieve high accuracy. Also, Table 7 lists the most important attributes in the training of Experiment 1.

TABLE VI. ATTRIBUTE REDUCTION FOR EXPERIMENT 2

Attributes	TP rate	FP rate	Precision	Recall	F-Measure	Accuracy
1889	0.959	0.074	0.959	0.959	0.959	0.959
800	0.958	0.072	0.958	0.958	0.958	0.958
250	0.952	0.048	0.952	0.952	0.952	0.952
120	0.95	0.085	0.95	0.95	0.95	0.95
80	0.953	0.073	0.954	0.953	0.953	0.953
40	0.923	0.018	0.922	0.923	0.921	0.923
20	0.912	0.231	0.914	0.912	0.908	0.912

F. Comparison

In this section we compare our method with other works. The comparison is summarized in Table 8. Wang et al. [10] built and analyzed API sequences with static analysis. They used 461 benign and 451 malicious programs as the dataset and obtained 93.9% detection rate. Islam et al. [11] utilized the function length and printable information as features for classification. They used 3996 malwares in experiments and they can identify 83.3% malwares. Tian et al. [18] monitored the behaviors of malwares and then analyzed the API call records by classification tools. They extracted features by the existence of each API call. They tested 454 benign and 1369 malicious programs but they executed benign samples three times. Hence, they have equal amount of benign and malicious records. Their detection rate reaches 97.3%. Salehi et al. [19] also analyzed the API records, but they analyzed not only the API names but also the API parameters. Their detection rate reaches 98.1% with 410 attributes in total and 385 benign and 826 malicious programs. Our method can achieve high accuracy but more efficiency according to the number of attributes. Some researches did not specify how much attributes they used in the experiments, so we assume that they took all attributes for training.

TABLE VII. IMPORTANT APIS IN EXPERIMENT 1

API Function Name	Explanation
LdrLockLoaderLock	Enter to a load lock.
GetModuleHandleA	Get the handle of specific module.
NtProtectVirutalMemory	Modify the protection of blocks of memory.
RtlNtStatusToDosError	Convert NTSTATUS code to system error.
RtlAnsiStringToUnicodeString	Convert ANSI string to Unicode string.
NtAllocateVirtualMemory	Allocate virtual memory in a process.
RtlAcquirePebLock	
wcsrchr	Locate the last occurrence of wide character.
RtlFreeUnicodeString	Clear a string buffer.
RtlReleasePebLock	

TABLE VIII. COMPARISON

	Experiment Data	Analysis	Attributes	Accuracy
[10]	461(B)/451(M)	Static	not provided	93.9%
[11]	3396(M)	Static	not provided	83.3%
[18]	454(B)/1369(M)	Dynamic	not provided	97.3%
[19]	385(B)/826(M)	Dynamic	410/166	98.1%/93%
Ours	251(B)/263(M)	Dynamic	80	93.8%
	251(B)/773(M)	Dynamic	80	95.3%

B: Benign

M: Malicious

V. CONCLUSION AND FUTURE WORKS

Techniques for network attacks and anti-detection are upgraded continuously. Malwares have widely taken encryption techniques to counter static analysis and signature-based detection. Except static signatures, malware can also conceal its dynamic signatures in order to escape inspection. Malwares attempt to keep stealth in the target hosts because network attacks have gradually changed into penetration attacks. Hence, we believe that new malwares will continue to adopt these anti-detection techniques (stealth techniques).

In this research, we utilize hooking techniques to trace and monitor the behaviors that samples perform. Behavior records

are used for classification training and building a description model. After that, our system can identify malwares by this model. We select Naive Bayesian, J48 (Decision Tree), and Support Vector Machine as our classification algorithms. We also apply attribute selection technique to reduce the amount of attributes for training so as to improve efficiency.

Our experimental result shows that, the detection rates are up to 95% for the J48 and Naive Bayesian algorithms. The Support Vector Machine algorithm has the worse performance, with a detection rate of up to 89%. Since the J48 algorithm usually needs only tens of attributes for building a tree, we use J48 for further experiment. We reduce the amount of attributes with InfoGainAttributeEval selection and the result shows that we can achieve a high detection rate with fewer attributes and significantly reduce complexity.

We propose an integral fashion in monitoring behaviors but our method has some limits. We believe that there are some ways to improve our work in the future: 1) All of our tracing techniques are under Userspace. The monitoring overhead and stability can be improved by Kernelspace hooking. 2) In our method, we monitor the injected process by monitoring the entire process. The tracing can be more accurate if we only trace the tainted threads, not the entire process.

ACKNOWLEDGMENT

This work was supported in part by the Taiwanese Ministry of Science and Technology under Grant MOST 103-2221-E-110-057 and by the NSYSU and the Taiwanese Ministry of Education through the Aim for the Top University Plan.

REFERENCES

[1] TrendMicro, "Trendlabs2012 annual security roundup evolved threats in a post-pc world," TrendMicro, http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-evolved-threats-in-a-post-pc-world.pdf, Tech. Rep., 2012.

[2] AV-TEST, "2013 malware statics from av-test institute," AV-TEST Institute, http://www.av-test.org/en/statics/malware/, Tech. Rep., 2013.

[3] TrendMicro, "Detecting apt activity with network traffic analysis," TrendMicro, http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-detecting-apt-activity-with-network-traffic-analysis.pdf, Tech. Rep., 2012.

[4] ——, "The heartbeat apt campaign," TrendMicro, http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the-heartbeat-apt-campaign.pdf, Tech. Rep., 2012.

[5] J. Lee, K. Jeong, and H. Lee, "Detecting metamorphic malwares using code graphs," in *Proceedings of the 2010 ACM symposium on applied computing*, 2010, pp. 1970–1977.

[6] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1020–1025.

[7] B. Kang, T. Kim, H. Kwon, Y. Choi, and E. G. Im, "Malware classification method via binary content comparison," in *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, 2012, pp. 316–321.

[8] P. Faruki, V. Laxmi, M. Gaur, and P. Vinod, "Mining control flow graph as api call-grams to detect portable executable malware," in *Proceedings of the Fifth International Conference on Security of Information and Networks*, 2012, pp. 130–137.

[9] K. Iwamoto and K. Wasaki, "Malware classification based on extracted api sequences using static analysis," in *Proceedings of the Asian Internet Engineeering Conference*, 2012, pp. 31–38.

[10] C. Wang, J. Pang, R. Zhao, W. Fu, and X. Liu, "Malware detection based on suspicious behavior identification," in *Proceedins of First International Workshop on Education Technology and Computer Science, 2009.*, 2009, pp. 198–202.

[11] R. Islam, R. Tian, L. Batten, and S. Versteeg, "Classification of malware based on string and function feature selection," in *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second*, 2010, pp. 9–17.

[12] H. Zhao, M. Xu, N. Zheng, J. Yao, and Q. Ho, "Malicious executables classification based on behavioral factor analysis," in *Proceedings of International Conference on e-Education, e-Business, e-Management, and e-Learning, 2010.*, 2010, pp. 502–506.

[13] S. B. Mehdi, A. K. Tanwani, and M. Farooq, "Imad: in-execution malware analysis and detection," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 1553–1560.

[14] V. P. Nair, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi, "Medusa: Metamorphic malware dynamic analysis usingsignature from api," in *Proceedings of the 3rd international conference on Security of information and networks*, 2010, pp. 263–269.

[15] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of api call signatures," in *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, 2011, pp. 171–182.

[16] K. Tsyganok, E. Tumoyan, L. Babenko, and M. Anikeev, "Classification of polymorphic and metamorphic malware samples based on their behavior," in *Proceedings of the Fifth International Conference on Security of Information and Networks*, 2012, pp. 111–116.

[17] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host." in *USENIX Security Symposium*, 2009, pp. 351–366.

[18] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, 2010, pp. 23–30.

[19] Z. Salehi, M. Ghiasi, and A. Sami, "A miner for malware detection based on api function calls and their arguments," in *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on*, 2012, pp. 563–568.

[20] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," University of Mannheim, Tech. Rep., 2011.