Unknown Malicious Executables Detection Based on Run-time Behavior

Yongtao Hu[1], Liang Chen[2], Ming Xu[2], Ning Zheng[2], Yanhua Guo[2]
tyhu@trimps.ac.cn *mecury5460@163.com* *mxu@hdu.edu.cn*
1. The Third Research Institute of the Ministry of Public Security
2. Hangzhou Dianzi University

Abstract

Traditional anti-virus scanner employs static features to detect malicious executables. Unfortunately, this content-based approach can be obfuscated by techniques such as polymorphism and metamorphism. In this paper, we propose a malicious executable detecting method using 35-Dimension feature vector. Each dimension stands for a malicious run-time behavior feature represented by corresponding Win32 API calls and their certain parameters. An automatic executable behavior tracing system (Argus) is also implemented to dynamically capture the features. Experiments are performed on a data set of 8223 malicious and 2821 benign executables. Training set is then used to generate detection model and several testing groups are set up for classification. Experiment result suggests that the method is efficient in detecting previously unknown malicious executables which have more than two behavior features captured.

1. Introduction

Malicious executables are computer programs which do damage or inconvenience to computer users. Typical examples include computer viruses, Trojan horses, worms, spyware and adware [1].

Technologies used to detect malicious executables can be separated into technical component and analytical component [2].

A technical component: it provides data to be analyzed by the analytical component. The data may be file byte code, text strings and action of a malicious executable running within the operating system.

An analytical component: it acts as a decision-making system. It makes a decision whether the analyzed data is malicious according to its rules.

This work is supported by Science and Technology Commission of Shanghai Municipality (No. 06511502), Natural Science Foundation of Zhejiang Province of China (No. Y106176) and Technology Search Planned Projects of Zhejiang Province of China (No. 2007C33058).

Traditional signature-based anti-virus scanner gets segments of file content as the technical component. The analytical component is just a simple comparison between the segments and the signature-pattern database. This method gives birth to a very low false-positive fraction near to zero while it performs poorly when facing with previously unknown malicious executables or variants of existing ones.

Current anti-virus scanner involves static heuristic [3] to alleviate this problem. Instead of looking for specific signature of a virus, it looks for virus behavior [4]. Each signature is a generic code sequence that represents a behavior feature and a complex comparison is invited in the analytical component. However, this method also drives data form the file content as the technical component and can be obfuscated by techniques such as polymorphism and metamorphism [5]. Although wildcard have been added to the code sequence to resolve the obfuscation problem, a high false positive fraction comes along consequently. On the other hand this method depends on aided techniques such as unpacking, decryption and disassembly.

To overcome the limitation of driving data from file, in this paper, the data is driven from the action of malicious executables when running within the operating system. Our main contributions are summarized as follows:

1. We concluded the characteristic behavior often seen with malicious executables and defined each behavior feature with corresponding Win32 API calls and their certain parameters.
2. An automatic executable behavior tracing system is implemented to dynamically capture the features we defined.
3. Experiment result suggests that model based on the features is efficient in detecting previously unknown malicious executables which have more than two behavior features captured.

The rest of this paper is organized as follows: Section 2 describes previous work on malicious executables detection based on malicious behavior. Section 3 tells the malicious behavior feature definition. In section 4, the

978-0-7695-3305-6/08 $25.00 © 2008 IEEE
DOI 10.1109/FSKD.2008.185

Table 2. Definition of five behavior features

Behavior Features	Related APIs	Parameter Description
COPY_PEFILE	CopyFile CopyFileEx	PE files, sensitive directories
TERMINATE_PROCESS	TerminateProcess	Any process
CLICK_KEYBOARD	Sendmessage	Keyboard information
SET_CRISIS_REG	RegSetValue	Sensitive key such as auto run
MODIFY_SERVIC	ChangeServiceConfig	Such as close update service wuauser

architecture of Argus is presented. In section 5, Experimental result is analyzed and a strategy of detecting malicious executables is proposed. Section 6 is the conclusion and outline of future works.

2. Related works

Related works are focused on two aspects most close to our work, the way of getting Win32 API calls and the definition of malicious behavior.

J. Y. Xu, A. H. Sung, P. Chavez, and S. Mukkamala [6] use Win32 API call sequence to reflect the behavior of an executable and a PE binary parser is developed to extract static API call sequence. It extracts the CALL instructions and finds their target APIs. This method depends on decryption and disassembly and research [7] shows that incorrect disassembly output is produced when confronts with control flow obfuscation.

C. Willems, T. Holz, and F. Freiling [8] present CWSandbox, which executes malware samples in a simulated environment, monitors all system calls, and automatically generates a detailed report to simplify and automate the malware analyst's task. It monitors all the executed functionality.

Most similarity to our work is the system implemented by R. Koike [9]. It captures Win32 API calls of executables when running within a closed environment. Relation between monitored behavior and API calls is defined. However, only two malicious behavior features, file creation and registry change are monitored. This may be not enough to issues a verdict whether an executable is malicious.

In this paper, Win32 API calls together with their certain parameters are captured dynamically and more malicious behavior features are defined.

3. Malicious behavior feature definition

In general, the way of representing the malicious run-time behavior can be divided into two levels, the machine instruction level and the operating system interface level. For the first level, a sequence of CPU instructions is used. For the operating system interface level, the Application Programming Interface (API) which allows applications to exploit the service of the operating systems is used.

Taking into account that most executables nowadays target 32-bit Windows platform and exploit the service of the operating system, Win32 API is chosen to represent malicious behavior.

However, the definition of malicious behavior with Win32 API alone could result in many normal Win32 API calls are treated unjustly. The same Win32 API call with different parameters may have various risk ranks. To copy a .dll file or an .exe file to a system-sensitive directory tends to be considered malicious while copying a .txt file to user's document directory is thought to be normal and benign. Table 1 shows the risk rank of CopyFile with different parameters. In the table, Param1 refers to file name to be copied, Param2 refers to the destination. System-sensitive directory represents %Windir%, %System%, %Temp% and so on.

Table 1. Risk evaluation of CopyFile

Param1	Param2	Risk rank
*.dll	system-sensitive directory	harm
*.exe	system-sensitive directory	harm
*.dll	Other directory	less harm
*.exe	Other directory	less harm
*.txt	Any directory	benign

To describe malicious behavior more accurately, the parameters are included with Win32 API to represent the malicious behavior. Six classes of malicious behavior are concluded respectively related to file, process, window, network, register, and windows service. Some are listed below.

- **File-related behavior**

1. Creating PE files under system-sensitive directory.
2. Copying PE files to system-sensitive directory.
3. Writing system-sensitive files such as PE files.

- **Process-related behavior**

1. Writing memory of other processes.
2. Creating remote thread in other processes.
3. Terminating other processes.

- **Window-related behavior**

1. Hooking keyboard.
2. Hiding window.

- **Network-related behavior**

1. Binding and listening to port
2. Initiating http connection.

Table 3. Malicious and benign sample distribution with the number of features

Features / Class	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Malicious (%)	11.95	22.74	17.76	11.58	11.15	9.45	5.90	4.27	2.30	1.37	1.03	0.26	0.21	0.02	0.00	0.01
Benign (%)	2.41	10.92	7.94	69.27	5.42	2.23	0.99	0.21	0.25	0.25	0.07	0.00	0.00	0.00	0.04	0.00

- **Register-related behavior**

1. Creating and Setting register key for automatic running.
2. Setting register to lower security check.

- **Windows service behavior**

1. Terminating windows update service
2. Terminating windows fire-wall.
3. Opening telnet service.

A 35-Dimension feature vector is finally defined with each dimension standing for one behavior feature. Table 2 gives the definition of five malicious behavior features.

4. Auto feature extraction

To dynamically capture the malicious behavior features, an automatic executable behavior tracing system (Argus) is implemented. Figure 1 presents its architecture.

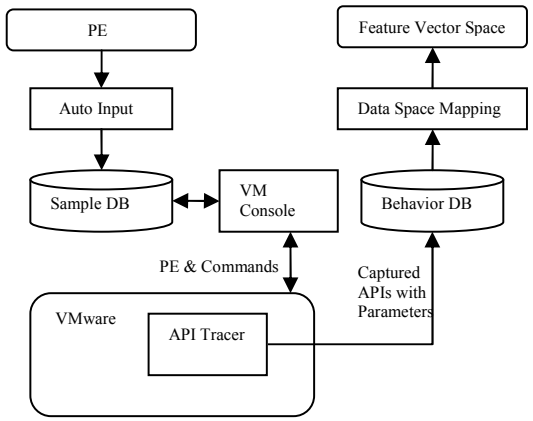

Figure 1. The architecture of the Argus

1. Auto Input: it automatically input executables of PE format into the Sample DB. PE structure rather than simple .exe suffix is used to exclude non-PE executables.

2. VM Console: it controls the running of VMware and API Tracer. It consults to the sample DB to acquire the path information and transferred PE executables into VMware. The transfer is realized by set share privilege onto folder including the executables. To protect the VMware from infection, a clean snapshot of VMware is recovered every time it runs a new executable.

3. API Tracer: it is a tool that can monitor the running Win32 API calls of executables and translate living process stack to API parameters. This tool is base on windows debug technology.

4. Data Space Mapping: The output of API Tracer is tracing records of APIs with their parameter. This module maps this data into the feature vector space for modeling.

5. Evaluation

5.1 Data set description

The data set consists of 11044 samples split into 8223 malicious and 2821 benign executables. Malicious executables are provided by ANTIY laboratory, a member of CNCERT/CC. (National Computer Network Emergency Response Technical Team/Coordination Center of China). Benign executables are collected from the Internet.

Both malicious and benign executables are strictly checked before chosen to construct the data set. Currently, the API Tracer only targets executables of PE format. Non-PE executables will not be included in the data set. For the benign executables, they are firstly scanned by kaspersky to eliminate hidden malicious ones. And then, repeated benign executables are deleted according to their MD5 value.

All the samples are dynamically traced by the Argus to extract malicious behavior features. Each sample is then transformed to a vector of the form V = (FI, F2, ⋯ , F35).

5.2 Data noise filtering

Statistical analysis had been made to the data set before the modeling. Table 3 presents the sample distribution with the number of different features captured. Of the 8223 malicious samples, 11.95% have no feature captured. The reason for this can be complex. One reason is that samples do not run successfully and fully if needed DLL (Dynamic Link Library) can not be found in the virtual environment or they need interaction with human beings. The other reason is that malicious behavior of the samples is beyond our definition or the malicious tend to be benign essentially.

No matter how, model based on the 35-dimension feature vector can not classify these non-feature samples at all.

Another problem must to face is the malicious executable variants. This may result in training set includes too many variants belonging to one family.

To avoid the data noise affecting the model, two data noise filtering rules are presented. As to the feature number filtering rule, malicious samples with no feature captured are not chosen for modeling. For benign samples with no feature captured, we assume that they do not have these features in essence and include them for modeling. As to the variant filtering rule, one of the variants is selected according to their naming regulation.

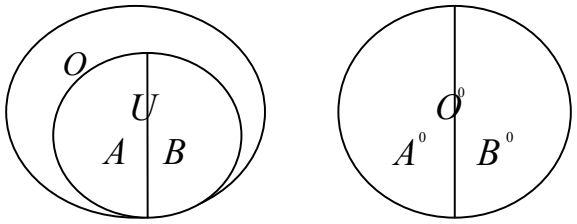

Figure 2. Sets in the experiment

Figure 2 shows the relationship among the sets. Set O derives from the 8223 malicious samples according to the feature number filtering rule. It includes 7240 malicious samples. Set U derives from set O according to the variant filtering rule. It contains 1865 malicious samples. Set O^0 stands for benign samples. It includes 2821 benign samples.

5.3 Evaluation measures

TPF, FPF and ACY are used to evaluate the model based on our features**.**

True Positive (TP): Number of malicious executables correctly classified to be malicious.

True Negative (TN): Number of benign executables correctly classified to be benign.

False Positive (FP): Number of benign executables incorrectly classified to be malicious.

False Negative (FN): Number of malicious executables incorrectly classified to be benign.

True Positive Fraction (TPF): $= TP/(TP+FN)$

False Positive Fraction (FPF): $= FP/(FP+TN)$

Accuracy Rate:

ACY $= (TP+TN)/(TP+TN+FP+FN)$

5.4 Model training and classification

Set U is randomly divided into set A and B at ratio 4:1. The number of set A^0 is set to equal to set A. 1282 malicious samples and equal quantity benign samples are then selected to train Naïve Bayes Classifier.

Classification tests are separately taken to 4 groups. The experiment is repeated five times and TPF, FPF, ACY obtained from the five iterations are averaged to present the experiment results in Table 4.

Table 4. Experiment result

Group	Sets	TPF(%)	FPF(%)	ACY(%)
1	$B \bigcup B^0$	69.26	7.64	88.23
2	$A \bigcup A^0$	67.94	7.70	80.12
3	$U \bigcup O^0$	68.21	7.66	83.46
4	$O \bigcup O^0$	64.98	7.66	72.52

FPF for each testing group is low and steady. TPF of group 1 indicates the ability of the model to detect previously unknown executables. The main reason of the not high TPF is that many malicious samples have few features. This makes them classified as benign ones easily. Further analysis of these malicious samples shows that the number of these samples with one and two features contributes 84.9% of the false negative.

Table 5. False negative distribution

Number of Features	Number of samples	Feature4 captured	Feature16 captured	Feature18 captured
1	1500	1500	0	0
2	677	677	443	174
3	151	141	48	19

The other reason is the similarity of captured features between the malicious samples and benign ones. Table 5 shows the top three contributions of the false negative in one experiment on group 4. Of the 2563 false negative, 1500 have only one feature captured and all of these samples have Feature4 captured. According to the behavior definition, this feature refers to ISDEBUG. By this activity, samples get to know if they are debugged. Another frequently captured feature is Feature16, CREATE_PROCESS. Both features happened 677 and 443 times within the 677 false negative which have two features captured.

This is same to the benign samples with less than three features. Of the 308 benign samples with only one feature captured, 296 have feature4 captured. Of the 224 benign samples with two features captured, 217 have feature4 captured too.

Both the few features and similar features with benign samples make the malicious samples classified as benign ones easily. Although the TPF for the whole 7240 malicious samples is not high, the detecting rate of

malicious samples with more than two features captured is acceptable. Table 6 shows the malicious executable detecting rate distribution with different features in one experiment on group 4.

Table 6. Detecting rate distribution

Number of Features	Samples	Detected	Detecting rate (%)
1	1870	370	19.79
2	1460	783	53.63
3	952	801	84.14
4	917	764	83.32
5	777	714	91.89
6	485	466	96.08
7	351	351	100.00
8	189	189	100.00
9	113	113	100.00
10	85	85	100.00
11	21	21	100.00
12	17	17	100.00
13	2	2	100.00
14	0	0	100.00
15	1	1	100.00
Total	7240	4677	64.60

Further experiment is taken to training set and testing groups with a feature threshold of 3. That is, malicious samples with features less than the feature threshold will not be included in the training set and testing group. Table 7 shows the classification results with a threshold of 3.

Table 7. Experiment result with a threshold of 3

Group	Sets	TPF(%)	FPF(%)	ACY(%)
1	$B \bigcup B^0$	89.27	7.91	91.87
2	$A \bigcup A^0$	89.64	7.15	91.24
3	$U \bigcup O^0$	89.56	7.71	91.63
4	$O \bigcup O^0$	88.17	7.71	89.90

The experiment result improves with a TPF increase of 20% while the FPF keeps steady. Corresponsively, a strategy for malicious executable detection based on the method is proposed. For executables with one or two features, alarms of captured features are reported. For executables with more than two features captured, the classifier makes a decision whether they are malicious or not.

6. Conclusion and future works

In this paper, we propose a malicious executable detecting method using 35-Dimension feature vector and define each feature with corresponding Win32 API calls and their certain parameters. Model based on the features performs well in detecting malicious executables, but it is still limited to detect executables with more than two features captured. Future work will focus on two aspects: concluding more characteristic malicious behavior and perfecting the Argus performance. We intend to increase the dimension of the feature vector so that the executables can have more features. The main problem facing the Argus is that many malicious executables need human interaction when running. If they do not run fully or successfully, few features are still be captured by the Argus. In fact, most of the interaction with human beings is just a button clicking. So, future work aims to let the Argus to click for itself.

References

[1] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables", in 2001 IEEE Symposium on Security and Privacy, 2001, pp. 38-49.

[2] Alisa Shevchenko, "The evolution of technologies used to de tect malicious code", 2007, *http://www.viruslist.com/en/analysis?pubid=204791972*

[3] Jay Munro, "Antivirus Research and Detection Techniques", 2002, http://www.extremetech.com/article2/0,1697,325439,00.asp

[4] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior", in first Conference on India Software Engineering Conference, 2008, ACM press, pp. 5-14.

[5] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns", in Twelfth Security Symposium, 2003, pp. 169-186.

[6] J. Y. Xu, A. H. Sung, P. Chavez, and S. Mukkamala, "Polymorphic Malicious Executable Scanner by API Sequence Analysis", in Fourth International Conference on Hybrid Intelligent Systems, 2004, pp. 378-383.

[7] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection", in Twenty-Third Annual Computer Security Applications Conference, 2007, pp. 421-430.

[8] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox", in Security & Privacy: IEEE Computer Society, 2007, pp. 32-39.

[9] R. Koike, N. Nakaya, and Y. Koi, "Development of System for the Automatic Generation of Unknown Virus Extermination Software", in International Symposium on Applications and the Internet, 2007, pp. 8-8.