Detecting malware using process tree and process activity data

Krijn Wijnands
Faculty of Technology, Policy and Management
Delft University of Technology,
the Netherlands
Email: k.j.wijnands@student.tudelft.nl

***Abstract*—In the last few years malware is incurring more damage and has become more sophisticated. Current security solutions are still based on signature and know behavior based detection. This renders them incapable of detecting new malware. In this paper we will present an anomaly detection method based on the combined data from process activities and process trees. We assume that processes from the same application show comparable process activities and different applications show differences in the process activities. Using a distance measure on the process characteristics, depth and cluster from process a and process b, we can show which processes deviate from the known processes. The detection algorithm tries to minimize the distance for every process in the datasets. Evaluation showed that the presented algorithm could detect processes from two of the three malware samples used. The highest TPR gained was 0.917. For future research we would recommend using a data collection set-up in which all data is collected on one machine.**

***Index Terms*—malware detection, process tree, process activities, anomaly detection.**

I. Introduction

MALWARE is a huge problem in today's IT environment. And the predictions are these will incur more damage and become more sophisticated [1], [2]. Headlines as *"Enterprise bank accounts targeted in new malware attack"* [3], *"Hackers attack the energy industry with malware designed for snooping"* [4], *"Hackers exploit Flash in one of the largest malware attacks in recent history"* [5] are not uncommon and are all from the first eight months of 2015. According to [6] the number of new malware samples discovered each year is rising significantly, from around 80 million samples in 2013 up to 143 in 2014.

With these huge numbers of new malware samples released, it is difficult for anti-virus vendors to keep up to speed with their protection against malware. The reason for this is that most security solutions are still based on a combination of Signature-based Detection and Sandboxing. In signature-based detection hashes of known malware files are used to detect it on a computer. Sandboxing runs the executable with strict policies on the host, such that the executable has thinks it can execute all its commands. The behavior of the executable will be compared to know malicious behavior.

From the above information it can be concluded that current security solutions are still based on detecting known malicious behavior. This creates a head start for the malware developers and the damage is done before the security vendors can update their list of known malicious behavior.

To solve this problem, a detection method should be used that does not rely on known behavior and signatures of known malware. In current scientific literature a lot is written on detecting malicious behavior on computers or networks. The main distinction in detection is made between misuse detection and anomaly detection. Misuse detection is still based upon known malicious behavior and is not sufficient for detection 0-day malware or exploits [7]–[9]. In the contrary anomaly detection is more suitable for detecting 0-day malware and exploits. The anomaly detection model is based on know normal behavior and has the ability to detect deviations from this known normal behavior [7]–[10]. However a disadvantage of anomaly detection is the higher number of false positives it generates [11] in comparison to misuse detection.

In this paper we will present a novel anomaly detection method for malware based on combined data from process activities and process trees.

In the next section related work will be discussed. After the related work we will introduce the assumption on which our presented algorithm is based. Section IV we will explain what data is collected. The following section will provide an overview on how the data is collected, after which in section VI the data processing is explained. In section VII will explain the novel detection algorithm, which shall be evaluated in following section. This paper will end with the conclusion in section IX and recommendations for future research in section X.

II. Related Work

IN [12] anomaly detection on Linux is done by using process related information, which includes the relationships among processes. This information is used to create a graph showing the relations between: processes and processes, processes and programs and processes and system calls. Each node in the tree consists of two parameters $name_{proc}$ and $stad_d$. To be able to detect malicious behavior the distance between the $stad_d$ of the two nodes is calculated. Then the model is trained using a supervised SVM on randomly selected 75% of the dataset and evaluated on the remaining 25%. This was repeated nine times, rendering an accuracy between 0.71 and 0.87.

The concept of process trees for malware detection on Linux machines is also used in [13]. However instead of system calls the command line options are recorded.

In [14] a anomaly detection is proposed by using process properties from Windows systems. The process properties used are: changes to Windows registry, changes to filesystem, infection of running processes, network activity and the starting and stopping of Windows services.

In this paper we will extend the above presented work by presenting a concept of anomaly detection for a single Windows host based on the use of process trees and process activity characteristics. The information of the process trees created, will be combined with the activity characteristics of

the processes. This will be done for malware free datasets, as well datasets containing malware infections. Then we will construct three comparing methods to compare the constructed malware datasets against the clean datasets. The outcome of these comparisons will be used for detection malicious behavior.

III. Assumption

The main assumption on which the presented method is based, is the that processes from the same application show comparable process activities and different applications show differences in the process activities. If we compare the process activities of the processes from different applications against each other it will generate a higher distance than when comparing processes from the same application.

IV. Dataset Description

The data used, is collected by an endpoint security application which can log low level process information on Windows machines and contains the following eight type of events that can be triggered by a process:

- filesystem
- registry
- process create
- process exit
- thread create
- thread exit
- module load
- object callback

All the event types have the following common data: an unique process id, an unique id assigned by the endpoint security application, and a timestamp. The rest of the data contains event specific data. For example the filesystem event contains information on what kind of filesystem action is performed, e.g. a write or read action. The registry event contains information on what registry key action was performed and on which registry key. Of the data collected, about 85 to 90% are filesystem events, the registry takes another 8 to 10% of the collected events.

V. Data collection

For our research we have collected four clean datasets containing each a full boot cycle. Two were collected during a normal working day and have a time span of around 7 hour and 50 minutes. The other two clean datasets are of a duration of less than an hour. Collecting of the data was done on a employee's workstation.

Due to security limitations the collection of the malware data had to be done in a virtual machine. We tried to create an identical environment as possible. For the creation of the malware datasets three different types of malware were used. Namely a banking malware (Dridex), a Remote Access Trojan and a variant of Zeus malware. Each of these malware samples was run in the VM whilst working behavior was simulated on the machine. Again due to security limitations, we were not able to do normal work on the machine for the risk of leaking personal or company information.

Therefor the collected malware datasets are much shorter, ranging from 20 to 40 minutes, in comparison to the clean datasets. This might be of influence on the outcome of the evaluation.

The malware datasets will be compared against all the clean datasets. More on the evaluation set-up in section VIII

VI. Data preparation

The collected data will be aggregated such that each row of the dataframe corresponds to a unique process id and contains node and edge information for the process tree. For every process we will count how often which event type is triggered and divide this by the total running time of the process in seconds. This will provide us with events triggered per second per process for each event type. This was done to eliminate the fact that processes running for a long time will show high event counts. To be able to compare the columns within a dataframe and between dataframes we normalize the data between 0 and 10, see equation 1. In which x is the value to be normalized, A and B are the minimum respectively maximum value of the variable to be normalized and a and b provide the range for the normalization. For our data a would be zero and b would be ten.

$$\frac{(x - A) * (b - a)}{(B - A)} \tag{1}$$

To get the maximum and minimum possible values of the dataset, all collected data was combined together to normalize each column. The reason for normalizing the data is the fact that filesystem and registry events, making up about 95% of the data, occur way more than a process create or thread create event. By normalizing the data on each process activity column we can easily identify high and low values.

A row of the aggregated dataframe contains the following variables: *unique process id, filesystem, registry, process create, thread create, module load, ob, unique parent process id, process executable path, parent process executable path*. Where the second till seventh variable, filesystem till ob, represent the number of times this event is triggered per second by the corresponding process. For example table I shows the normalized number of times such an event type is triggered per second by the unique process 9999.

TABLE I: Example of events per second

unique process id	filesystem	registry	process create	thread create	module load	ob
9999	0.008845	0.00092	$2.06e-05$	0.00669	0	0.00469

The process executable path is an tokenized string of the location of the executable. This information will not be used in in the proposed detection method, however it provides valuable information to check if a process belongs to the same executable.

As stated in section III we expect that processes from the same application tend to show the same process activity. If we cluster the processes based on the six activity types these processes will be in the same cluster. If a process shows deviating process activities it will be assigned to another

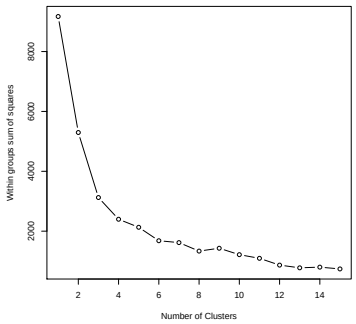

Fig. 1: K-means plot

cluster. A k-means clustering algorithm will be used for clustering as this is a widely used clustering algorithm in anomaly and misuse detection [15]. To cluster the data, the K-means "Hartigan-Wong" algorithm is used on the clean datasets by minimizing the within-cluster sum of squares, see equation 2 [16]. In which $i = 1, 2, ...n$, with n defining the number of events, so the number of processes in the dataset. j is defined as $j = 1, 2, ..., p$, in which p is the number of variables, so in our case six. $\overline{x(k, j)}$ is the mean of the variable j of all elements in a cluster k. The k used will be eight as the within group sum of squares does not decline that much more when selecting a greater k, see figure 1

$$Sum(k) = \sum_{i=0}^{n} \sum_{j=0}^{p} (x(i, j) - \overline{x(k, j)})^2 \tag{2}$$

The found cluster centers will be used to assign the processes of the malware datasets to their appropriate cluster by selecting the cluster center with the lowest distance. For calculating the distance the Euclidean distance will be used, see equation 3. The distance is calculated between two vectors x and y with the dimension i [17, pp.509]. In this case the dimensions are the eight variables mentioned above. A distance matrix contains the distance of every combination of processes between both datasets.

$$\sqrt{\sum (x_i - y_i)^2} \tag{3}$$

The data is now prepared to be tested by our detection algorithm, which will be discussed in the next section.

VII. Detection algorithm

This section will explain the algorithm used to compare the malware datasets against the clean datasets.

The algorithm can be described as follow: For every malware dataset these steps will be done:

1) select a clean dataset
2) For every depth present in the malware dataset we select the nodes in the malware dataset and clean dataset at the selected depth, starting from depth 0.
3) At every depth a distance matrix will be calculated using the Euclidean distance, equation 3, on the following variables:
   - filesystem
   - registry
   - process create
   - thread create
   - module load
   - ob
   - depth
   - fit cluster
4) From the calculated distance matrix we select the minimum distance present and assign the distance between the process from the clean dataset and malware dataset to the process from the malware dataset and set the distance to NA in the distance matrix.
5) repeat step 4 until all processes from the selected depth have an distance assigned
6) repeat steps 2 to 5 until all depths are done
7) Repeat steps 1 to 6 until the malware dataset is compared to all clean datasets.

The outcome of running this algorithm will create for every malware dataset four new dataframes containing a distance to a process in the matching clean dataset.

To mark a process as malicious we will use a threshold value for the distance. If a process has a distance higher than the threshold value it will be marked malicious. The used threshold values will be discussed in the next section.

As the usage of a programs can differ every day, comparing a dataset in which program A is used to a dataset where program A is not used, will result in a high distance and therefor might be marked as malicious. However when comparing to a dataset in which program A is used, the processes will have a low distance. However if a malicious process is present, it will have a high distance to every dataset.

Therefor a process in the malware dataset will only be marked malicious, if it is above the set threshold in all four comparison datasets. For example process i in the banking malware dataset has a distance above the threshold value for the comparison with all the four clean dataset it will be marked malicious.

VIII. Evalution

To evaluate the present algorithm we will test if the malicious process are marked as malicious by using six different threshold values. The values used are the mean, 75%, 80%, 85%, 90% and 95% quantile of the distances found in the compared malware dataframe.

As we know which processes are malicious we can calculate the True Positive Rate, the False Positive Rate and Accuracy, see the equations 4, 5 and 6. Where TP is True positive, malicious processes marked as malicious and FN is False Negative, malicious processes marked as benign. FP are the benign processes marked as malicious, and True Negative are the correctly marked benign processes.

$$TPR = TP/(TP + FN) \tag{4}$$

$$FPR = FP/(FP + TN) \tag{5}$$

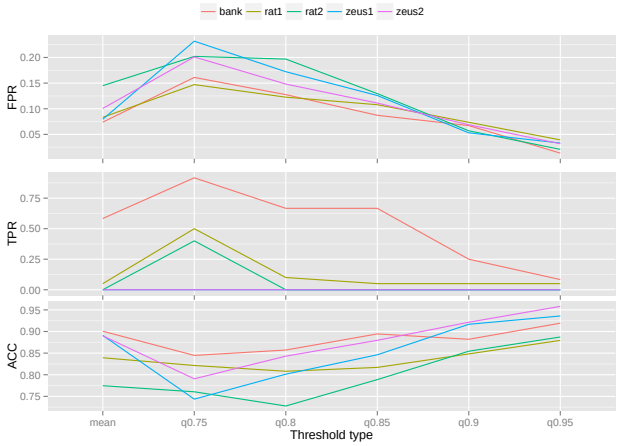

Fig. 2: The FPR, TPR and ACC of the algorithm for all six threshold types

$$ACC = (TP + TN)/(TP + TN + FP + FN) \quad (6)$$

In figure 2 the TPR, FPR and ACC is shown for all the datasets on every threshold type. The values for TPR range from 0 tot 0.917 (on the banking malware), for FPR is between 0.013 and 0.232 and the ACC range from 0.728 up to 0.958.

The threshold type given the best TPR is the 75% quantile, however together with a rising the TPR the FPR will rise as well and the ACC will go down.

The presented algorithm was capable of detecting at least some of malicious processes from the banking and RAT malware, however it was incapable of detecting any of the malicious processes from the Zeus malware.

We analyzed the malicious processes from the Zeus malware to find out why it was not detected. The processes from the Zeus malware showed low values on the process activities. This might imply that the Zeus malware was only installed and started listing for a command from command and control center, but not receive any. The not receiving of any command might have to do with the fact that the collection of the malware datasets was a very short period.

IX. Conclusion

In this paper we presented a novel anomaly detection method for malware based on combined data from process activities and process trees. We explained what kind of data was collected and which processing steps are taken. The evaluation of the detection algorithm showed that it was capable of detecting malicious processes from two of the three malware types. However a higher TPR give a higher FPR as well.

X. Recommendations

The set-up for our data collection was, due to security limitations, not ideal. For future research we would advise to perform the same experiment with the data, clean and malware, collected on the same machine. Hereby eliminating any inconsistencies in the programs installed and used.

In the conducted research only one k value was tested, in future research testing the impact of other numbers of k might render a higher success rate.

The data was normalized between zero and ten. However other normalization methods, such as Z-score, might render different results.

In addition during analyzing the data we concluded that some process perform a set number of actions, however different running times due to using different machines, will change the number of events per second. This will create other characteristics for a process whilst it is performing exact the same actions. Therefore further research should be conducted on converting the number of events into a value that can be comparable.

The number of malware samples tested was low, to get a better insight in the performance of the presented algorithm it should be tested on a larger amount of malware samples. The problem hereby is that generating the data for the malware samples is quite time consuming.

References

[1] "Security threat report 2014: Smarter, shadier, stealthier malware," Report, Sophos, 2013. [Online]. Available: http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf

[2] "Five predictions for information security and cybercrime in 2014," http://www.theguardian.com/media-network/media-network-blog/2013/dec/10/predictions-information-security-cybercrime-2014, Dec. 2013, [Online; accessed 30-June-2014].

[3] "Enterprise bank accounts targeted in new malware attack," www.pcworld.com/article/2906056/enterpise-bank-accounts-targeted-in-new-malware-attack.html, April 2015, [Online; accessed 1-September-2015].

[4] "Hackers attack the energy industry with malware designed for snooping," http://fortune.com/2015/03/31/spies-malware-energy-email, March 2015, [Online; accessed 1-September-2015].

[5] "Hackers exploit flash in one of the largest malware attacks in recent history," https://bgr.com/2015/08/04/hackers-flash-yahoo-malware-attack/, August 2015, [Online; accessed 1-September-2015].

[6] "Number of new malware per year," http://www.av-test.org/en/statistics/malware/, [Online; accessed 15-january-2015].

[7] J. Song, H. Takakura, Y. Okabe, and K. Nakao, "Toward a more practical unsupervised anomaly detection system," *Information Sciences*, vol. 231, no. 0, pp. 4 – 14, 2013, data Mining for Information Security. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025511004245

[8] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772 – 783, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366412000266

[9] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*, May 2010, pp. 305–316.

[10] J. M. Harjinder Kaur, Gurpreet Singh, "A review of machine learning based anomaly detection techniques," *International Journal of Computer Applications Technology and Research*, vol. 2, no. 2, pp. 185 – 187, 2013.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[12] C. Wagner, G. Wagener, R. State, and T. Engel, "Malware analysis with graph kernels and support vector machines," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 2009, pp. 63–68.

[13] G. Wagener, A. Dulaunoy, T. Engel *et al.*, "Self adaptive high interaction honeypots driven by game theory," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2009, pp. 741–755.

[14] K. Rieck, T. Holz, C. Willems, P. Dssel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, D. Zamboni, Ed. Springer Berlin Heidelberg, 2008, vol. 5137, pp. 108–125.

[15] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, June 2005, pp. 118–125.

[16] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. pp. 100–108, 1979. [Online]. Available: http://www.jstor.org/stable/2346830

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009.