Journal of Forensic Research - Open Access

www.omicsonline.org

Research Article
OPEN ACCESS Freely available online
doi:10.4172/2157-7145.1000112

NewApproach for Detecting Unknown Malicious Executables

Boris Rozenberg[1]*, Ehud Gudes[1], Yuval Elovici[2] and Yuval Fledel[2]

[1]Deutche Telekom Laboratories at BGU and Department of Computer Science
[2]Deutche Telekom Laboratories at BGU and Department of Information System Engineering, Ben Gurion University, Beer Sheva 84105, Israel

Abstract

We present a method for detecting new malicious executables, which comprise the following steps: (a) in an offline training phase, finding a set of system call sequences that are characteristic only to malicious files, when such malicious files are executed, and storing said sequences in a database; (b) in a real time detection phase, for each running executable, continuously monitoring its issued system calls and comparing with the stored sequences of system calls within the database to determine whether there exists a match between a portion of the sequence of the run-time system calls and one or more of the database sequences, and when such a match is found, declaring said executable as malicious. We have evaluated our method and the preliminary results are promising and justify the use of system calls sequences for the purpose of detection of new malicious executables.

Keywords: Malware Detection; System Calls Sequences

Introduction

Detection of malicious executables that are known beforehand is usually performed using signature-based techniques. These techniques typically rely on the prior explicit knowledge of the malicious executable code, which is in turn is represented by one or more signatures or rules that are stored in a database. The database is frequently updated with new signatures, based on new observations. The main disadvantage of these techniques is the inability to detect totally new, i.e., un-encountered malicious executables.

The goal of this paper is to provide a technique which can detect new malicious executables, whose signatures are unknown yet. The main prior art approach for performing such a task is to employ machine learning and data mining for the purpose of creating a classifier that is able to distinguish between malicious and benign executables statically (without actually running them) [1-3]. The main drawback of the above approach is its inability to deal with obfuscated/encrypted files.

In this paper we introduce a novel technique for the real-time detection of new malicious executables that follows dynamic analysis (or behavior-based) approach (detection during the execution). Traditionally, dynamic analysis approaches have been used in intrusion detection systems (IDS) based on anomaly detection [5-11]. These systems build models of a normal program behavior during a training phase, and then, using the models the systems attempt to detect deviations from said normal behavior during a detection phase. The main drawback of using these techniques is the necessity to perform a complex and frequent retraining in order to separate "noise" and natural changes to programs from malicious codes. Legitimate program updates may result in false alarms, while malicious code actions that seem to be normal may cause missed detections. Furthermore, most applications that are based on anomaly detection techniques identify malicious behavior of specific processes only.

Another using of dynamic analysis approach is for malicious code classification [12,15] and for detection of variations of known malware [18,19]. The techniques proposed in [12,15] can be used to classify a given malicious code instance as belonging to one of the predefined number of classes, but cannot be used for a new malicious code detection in real time. The methods proposed in [18,19] are suitable to detect variations of existing malware, but not completely new malware.

In this paper we try to provide a general, real time detection method that is more reliable than existing methods. Our method comprises of the following steps: (a) in an offline training phase, finding a collection of system call sequences that are characteristic only to malicious files, when such malicious files are executed, and storing said sequences in a database; (b) in runtime, for each running executable, continuously monitoring its issued run-time system calls and comparing with the stored sequences of system calls within the database to determine whether there exists a match between a portion of the sequence of the run-time system calls and one or more of the database sequences, and when such a match is found, declaring said executable as malicious. A major issue in this method is finding an optimal set of such sequences. We employ SPADE [4] and genetic algorithm (GA) to perform the first step - i.e. finding "behavior signatures" (sequences of system calls) that are characteristic to malicious executables and not to benign executables and use said signatures for the purpose of detection in the second step.

In this paper, we make three main contributions:

- We show that there are "behavioral signatures" (sequences of system calls) that can be used for detection of new malicious executables (and not only for classification).
- We present two methods for discovering the signatures above as well as their evaluation.
- We present overall system for real time malware detection that is based on the proposed method.

The rest of the paper is structured as following: in Section 2 we describe related work, in Section 3 we present the overall system architecture for real time detection of new malicious executables as

***Corresponding author:** Boris Rozenberg , Deptartment of Computer Science, Ben-Gurion University, Beer- Sheva, 84105, Israel, Tel: +972-8-6461626; Fax: (972)-8-6472909; E-mail: rozenbu@bgu.ac.il

Received December 15, 2010; **Accepted** December 29, 2010; **Published** December 31, 2010

Citation: Rozenberg B, Gudes E, Elovici Y, Fledel Y (2010) NewApproach for Detecting Unknown Malicious Executables. J Forensic Res 1:112. doi:10.4172/2157-7145.1000112

well as two methods for discovering "behavior signatures". Then, in Section 4 we describe our data collection and present evaluation results and we conclude in Section 5 with results discussion and future work.

Related Work

Static analysis approach

In recent years several approaches for detecting unknown malware based on its binary code have been presented. In [2] the authors introduced the idea of applying machine learning methods (ML) on binary code. They used *program header, string features* and *byte sequence features* to represent instances (malicious and benign programs) and developed three classifiers (rule-based classifier, *Naïve Bayes* and *Multi-Naïve Bayes*) to classify new (unseen) instances. They compared the accuracy of these methods to the accuracy of the signature-based method (antivirus) and show that all of the ML methods are more accurate than the signature-based method. In [3] the authors construct a representative profile for malicious and benign classes, composed of the common n-grams. New files are compared to both profiles and matched to the most similar, using the k nearest neighbor (KNN) method. In [1] the authors used a vector of n-grams to represent malicious and benign files and presented a comprehensive evaluation of IBk, TFIDF, naive Bayes, SVM, decision trees, boosted naive Bayes, boosted SVM, and boosted decision tree classifiers. The authors indicated that the results of their n-gram study were better than those presented by [2] and boosted decision tree classifier out-performed the others.

Behavior-based approach

While static analysis consists of examining the code of programs to determine properties of the dynamic execution of these programs without running them, behavior-based approach considers monitoring the execution of a program to detect malicious behavior.

The use of system calls: One common way to represent program behavior is to use a system calls sequences. Following is an example of the trace produced by sTrace [20] utility for Windows:

Sequences of system calls are used by several malware detection/classification systems and will also be used in our method (see below).

Anomaly detection systems: Typically, the systems belonging to this category build models of normal program behavior and then attempt to detect deviations from the normal model in the observed behavior. Variety of anomaly detection methods utilizing this approach has been proposed [5-11]. Forrest [5], for example, introduced a simple anomaly detection method based on monitoring the system calls issued by privileged processes. During the training phase, the proposed system records short system call sequences that represent a normal process behavior ("self") into Normal Dictionary. During detection phase, actual system call sequences are compared with the Normal Dictionary. An alarm is raised if no match is found.

Another approach, proposed in [6] is based on the idea that rare system calls sequences are suspicious. The authors suggest ranking each system call sequence by comparing how often the sequence occurs in normal instances with how often it is expected to occur during attack. Sequences occurring frequently during attack are declared as suspicious.

Several data mining techniques for studying system call sequences have been proposed. Lee and others [7,8] proposed a method for describing "normal" system call sequences by a (small) set of rules that cover the common elements in those sequences. During detection, sequences violating the rules are considered as anomalies. The main advantage of anomaly detection techniques is their ability to detect new, previously un-encountered malicious codes. The main drawback of using these techniques is the necessity to perform a complex and frequent retraining in order to separate "noise" and natural changes to programs from malicious codes. Legitimate program updates may result in false alarms, while malicious code actions that seem to be normal may cause missed detections. Furthermore, most applications that are based on anomaly detection techniques identify malicious behavior of specific processes only. Another common drawback of the anomaly detection methods presented above is their inability to cope with mimicry attacks [9,10]. A mimicry attack is an attack where the attacker can inject exploit code that imitates the system call sequence of a legitimate program run while performing malicious actions. Mutz and others [11] claims to overcome mimicry attacks by using a method that analyzes the arguments of system calls in order to prevent evasion and improve detection accuracy.

Behavior-based malware classification: Another application of dynamic analysis approach is in a malware classification domain. Lee et al. [12] proposed a malicious code classification technique which is based on clustering of system call sequences. Malicious programs of various classes are represented as sequences of system calls. A K-medoid Clustering algorithm, as described in [13], is applied to the sequences in order to map the input into a predefined number of different classes. The distance between sequences is defined by the minimum "cost" required in order to transform one sequence of system calls to another sequence of system calls, by applying a set of predefined operations. The process results in a classifier, which includes plurality of medoids, wherein each medoid is a best representative of each cluster. The classification of new objects is performed using the nearest neighbor classification method as described in [14]. A new object is compared to all medoids, and receives a class label of the closest one.

In [15] Bayer et al. proposed to generalize the malware' execution traces "into behavioral profiles, which characterize the activity of a program in more abstract terms". After the profiles have been created, the analyzed samples are clustered according to their behavioral profile. This technique produces more precise results than previous approaches.

The techniques above can be used to classify a given malicious code instance as belonging to one of the predefined number of classes, but cannot be used for a new malicious code detection in real time.

Behavior-based malware detection: In [18], the authors propose a method for automatic creation of specification of malware behavior. They introduce the concept of malware specification called Malspec. Malspec is a directed acyclic graph (DAG) where each node corresponds to a (relevant) system call invocation and edges represent dependences between arguments of different system calls. Malspec is extracted by contrasting the execution behavior of a known malware against the execution behaviors of a set of benign programs. The authors proposed the algorithm that creates the system-call graphs from execution traces, and derives a Malspec by computing the minimal differences between the system-call graphs of a malware sample and of multiple benign programs. The authors show that Malspecs can be converted to templates/signatures used by malware detectors to detect variations of a certain malware.

In [19], authors pointed out that Malspecs do not encode data flow dependencies between system call parameters, and that using Malspecs for detection without verifying these dependencies would lead to a large number of false alarms. They propose an approach that builds a behavior graph for analyzed malicious program where nodes are (interesting) system calls and edges represent a data dependency between the system calls. Then, they extract the program slices responsible for such dependences. For detection, they execute the extracted program slices to match them against the runtime behavior of an unknown program.

Both methods above are suitable to detect variations of existing malware, but not completely new malware.

Our Method

Our method determines and assigns sequences of system calls as representing the behavior of malicious programs. This is performed during a learning/training phase. During a detection phase, which is performed in real time, the method identifies malicious executables by comparing their own run time sequences of system calls with said stored (in the database) sequences of system calls that are characteristic to only malicious executables. Figure 1 is a flow diagram illustrating the process for detecting malicious executables. During the training phase (101), which is performed off-line, an "M determining module" (102) operates to determine M-sequences of system calls that are characteristic only to malicious executables, and not to any benign program. This module produces an "M database" (103) which includes the collection of M-sequences, as determined. The M database (103) forms an input data to comparator (104). During the runtime monitoring phase (105), the comparator continuously receives inputs relating to the system calls that are issued by the currently running executables, compares them separately with each of the sequences stored in the M database. If with respect to a specific running program a match is found with one or more of the M-sequences, this specific executable is declared as malicious and can be terminated. Otherwise, as long as no such an alert signal is issued, this running file is considered as benign.

We developed our system for the MS-Windows operating system but the same ideas can work for other OSs. Presently there are about 1100 different system calls for Windows operating systems. We define an M-sequence as a sequence of system calls occurring one after another in time, but not necessarily consecutive. This is the common definition for a sequence that is used generally in Data mining.

Figure 2 describes the training phase process for determining the database of M-sequences. The process comprises accumulation of *n* executables that are known to be malicious, and *m* executables that are known to be benign. In steps 201 and 202, each of said benign and malicious executables is executed, and runtime sequence of system calls is recorded for each of said executables. The result is two datasets, *Mr* dataset which contains *n* records of "raw" sequences relating to malicious executables, and *Br* dataset which contains m records of "raw" sequences relating to benign executables. The length of each of said *n* and *m* sequence records (within *Mr* and *Br*) is relatively long (we discuss this issue in the Evaluation section). It should be noted that there is no necessity for having a same sequence length for all the various "raw" recorded sequences within either *Mr* and/or *Br* datasets. In step 203, a set *S* of all frequent sequences in *Mr* which do not appear within any of the sequences in *Br* is determined. To perform this task we use the SPADE algorithm [4] and genetic algorithm (GA) discussed in the sections 3.1 and 3.2, but other approaches can be applied as well. In step 204 we reduce the set *S* to minimal set *Smin* that matches all the sequences (files) in *Mr*. This reduced set forms the database *M*. The result of the completion of the procedure of Figure 2 is the database *M*, which contains a collection of sequences of system calls that are characteristic to only malicious executables. The database *M* is used in runtime for detecting malicious executables, in a manner as was described above with respect to Figure 1.

SPADE

SPADE [4] is an algorithm for fast mining of sequential patterns in large databases. Given a database and a minimal support value, SPADE efficiently generates all sequences that repeat (i.e., frequent) in the database with a support equal to or grater than a minimal support value. In our implementation we enumerate frequent sequences in *Mr* for the 1%≤minSupp≤100% and take a minimal set of sequences that matches all the sequences in *Mr* (malicious files' traces) and no sequence in *Br* (benign files' traces).

SPADE uses lattice approach [16] to enumerate all frequent sequences. When we run our implementation of SPADE against a dataset we have created, we found that the lattice for our domain is too large to search. One of the examples we ran had 30 frequent 1-sequences and lattice depth of around 50, and the process was stopped after running three weeks and still searching in the first sub-lattice, the one that has the first frequent 1-sequence as its root. We had no choice but to limit the size of the lattice. This was done via limiting the number of system calls between first and last system call in the sequence. The idea can be thought of as a sliding window, where the user defines the window size. This strategy proved to reduce the processing time drastically. The disadvantage is that we are able to enumerate frequent sequences of limited length. In our experiments we get reasonable runtime for signature length less or equal to 15. We observed that as the signatures length increases the performance of our approach increases as well (we discuss the evaluation results in details in the section 4.1). This observation motivated us to use another approach to enumerate frequent sequences of arbitrary length.

In order to improve the performance of our approach, we employ Genetic Algorithm (GA), presented in the section 3.2. It allows us to

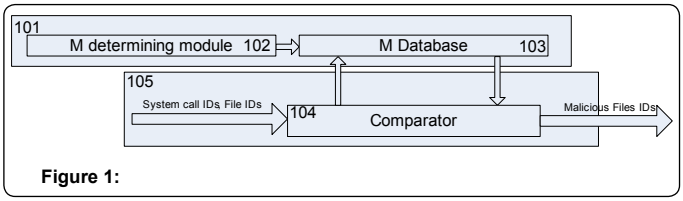

Figure 1:

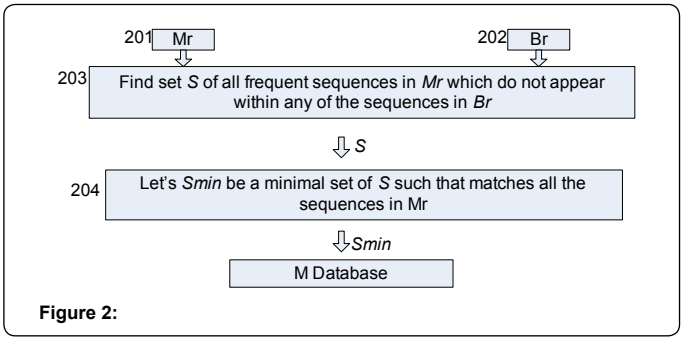

Figure 2:

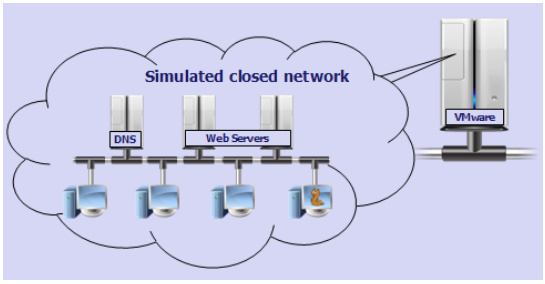

Figure 3:

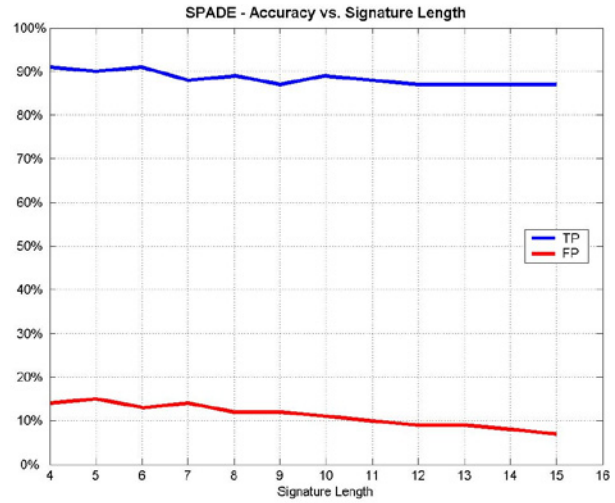

Figure 4:

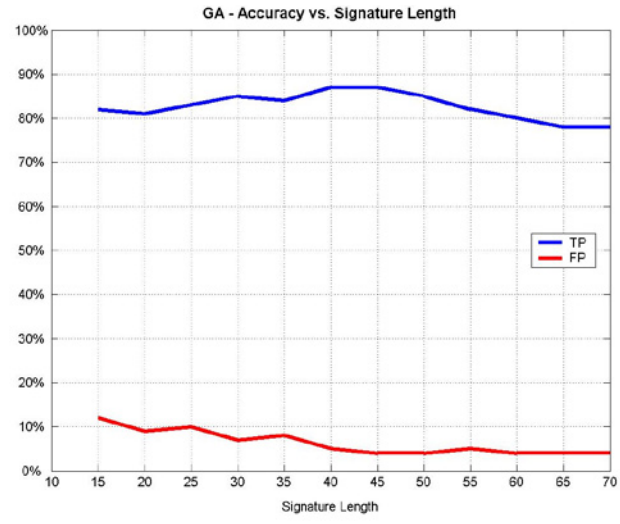

Figure 5:

search for larger signatures that, as we'll show later, produce better results.

Genetic Algorithm: Genetic Algorithms (GA) are based on the Darwinian principle of evolution i.e. the natural selection, also known as "Survival of the fittest" and can be used to solve computational problems. The mechanism of evolution is perfectly suited to be applied on the problems that require searching after an optimal solution among a lot of potential solutions in the searching space. Genetic algorithms typically used to provide a well enough approximate solution to the problems that can't be solved accurately in reasonable time.

- We have developed GA for mining frequent sequences of system calls. The input to the algorithm is two datasets (*Mr* and *Br*), signatures obtained by the SPADE as described in the Section 3.1, and the desirable signature length (N). The output is a set of signatures of size *N* that matches all malicious files in *Mr* and any benign file in *Br*. We start from a random population of signatures of length *N*, conducted from the signatures obtained by SPADE and populated randomly with system calls to get the desirable length *N*. Next step is to calculate a score for each signature. We define score of individual to be a number of malicious files matched by this individual (signature). We assign the score "0" to those individuals that match one or more benign files. Next, we perform the cross over and the mutation operations and calculate score for new individuals. We use the following cross over methodsFirst half of new signature is taken from signature 1, second half from signature 2
- All even indexes from signature 1, all odd indexes from signature 2

Mutation is defined as replacing (with a certain probability) of system call in the signature by another system call.

Finally, we sort all original individuals as well as the obtained off springs by the score value. The best individuals together with some new random signatures (another parameter to algorithm; we need this step in order to prevent a converge to a local maximum) comprise the new generation, which is given a score. The steps above are performed repeatedly for specified number of times (generations). At the end of the process we choose the minimal number of individuals (signatures) from the final population which matches all the malicious files in *Mr* and any of benign files in *Br*.

Evaluation

Figure 3 describes an evaluation environment we used to create datasets (Mr and Br). We use a virtual machine (VMware) that is booted with a kernel mode monitor agent, prior to the execution of each target file. Each executable is executed for 5 minutes. During this exemplary 5 minutes period, a running file typically invokes between 500 to 10,000 system calls. We take only those files that invoke at least 1000 system calls. The monitor agent intercepts and monitors all system calls in kernel mode and produces the datasets Mr (640 files) and Br (570 files). Each file is represented by first 1000 invoked system calls.

We acquired the malicious files from the VX Heaven website [17]. The files in the benign set, including executable and DLL (Dynamic Linked Library) files, were gathered from machines running Windows XP operating system in our campus.

To evaluate our method, we ran four-fold cross-validation (three folds for the training and one for the testing). We measure the true positives (TP) and the false positive (FP) rates as a function of signature length and average the results. Following subsections present the obtained results for SPADE and GA separately.

SPADE Evaluation

Figure 4 presents the evaluation results for signature length increasing from 4 till 15. From the figure we can see that short signatures tend to be more common, matching both malicious and

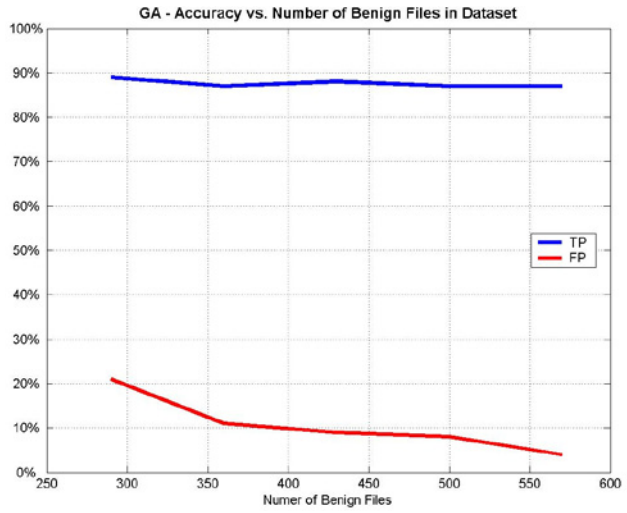

Figure 6:

benign files. As signature length increases the TP rate almost do not changes but the FP rate decreases. Signatures of length 15 produce the best results. We achieved a training accuracy of 100%. For given dataset we have discovered 28 characteristic sequences of length 15 that match 87% of malicious executables (from the testing set), with 7% false alarms.

GA Evaluation

We omit here the evaluation of GA parameters - we got the best results for number of generations equal to 15000, population size equal to 500, mutation probability equal to 0.1, and percentage of random individuals added to each generation equal to 10%. The following figure presents the evaluation results for signature length increasing from 15 till 70. Each next run takes the previously obtained signatures as input and populate them randomly to get the desirable signature length Figure 5.

First of all we can see the decrease of FP rate as the signature length increases. Secondly we can see the increase (or at least not decrease) in TP rate in the range 15<=SignatureLength<=45 and the decrease in TP rate for SignatureLength>45. This is because long signatures tend to be more file specific (over fitting).

A Signature length of 45 gave the best results – the highest TP rate and the lowest FP rate. However there is a tradeoff between the signature length and the detection time – as the signature length increases the detection time increases as well.

Our approach is based on the assumption that there is a similarity between certain groups of malware in terms of sequences of invoked system calls and these sequences do not present in any benign file. We can expect that the more benign files we have in the dataset, the better the FP rate we will receive. In order to justify this assumption we made several runs, all with the same parameters, but having different datasets. We took the original dataset which had 570 benign files and 640 malicious files, and formed four more datasets from the original dataset, each has 70 benign files less than its predecessor (meaning 500,430,360,290 benign files), but the same number of malicious files (640). In the graph below, we can see that the more benign files we have in the dataset, the FP rate decreases. This suggests that if we have more and variable benign files we'll get a better accuracy Figure 6.

In order to increase the TP rate we plan to improve the performance by finding correlations between the found characteristic sequences using additional data mining techniques (and not use each feature separately).

Conclusion

In this paper we show that there are "behavioral signatures" (sequences of system calls) that can be used for real time detection of new malicious executables. We present two methods for discovering the signatures above as well as their evaluation. Finally, we present overall system for real time malware detection that based on proposed method.

We used relatively small datasets for our evaluation. We plan to enlarge it up to 10000 malicious files and 10000 benign files. Also each file in the dataset was represented by first 1000 system calls only (due to the computation complexity). We believe that for the larger dataset with more system calls for each file we'll receive the better results. Also we plan to improve the performance by finding correlations between the found characteristic sequences using various data mining techniques.

Our goal was to show the reliability of our method. We show that the larger dataset of benign files will reduce the false positive rate. Large enterprises, such as antivirus providers could apply our method on large datasets, thus producing more practical results.

References

1. Kolter JZ, Maloof MA (2004) Learning to detect malicious executables in the wild. In: 10th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press 470-478.
2. Schultz M, Eskin E, Zadok E, Stolfo S (2001) Data mining methods for detection of new malicious executables. In: IEEE Symposium on Security and Privacy 38-49.
3. Abou-Assaleh T, Cercone N, Keselj V, Sweidan R (2004) N-gram Based Detection of New Malicious Code. In: 28th Annual International Computer Software and Applications Conference (COMPSAC'04) 2: 41-42.
4. Zaki MJ (2001) SPADE: Efficient Algorithm for Mining Frequent Sequences. Machine Learning 42: 31-60
5. Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A Sense of Self for UNIX Processes. In: IEEE Symposium on Security and Privacy 120-128.
6. Helman P, Bhangoo J (1997) A statistically based system for prioritizing information exploration under uncertainty. In: IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 27: 449-466.
7. Lee W, Stolfo SJ, Chan PK (1997) Learning patterns from UNIX process execution traces for intrusion detection. In: AAAI Workshop on AI Approaches to Fraud Detection and Risk Management 50-56.
8. Lee W, Stolfo SJ (1998) Data mining approaches for intrusion detection. In: 7th USENIX Security Symposium.
9. Tan K, Maxion R (2002) Defining the Operational Limits of Stide, an Anomaly-Based Intrusion Detector. In: IEEE Symposium on Security and Privacy 188-201.
10. Tan K, Killourhy K, Maxion R (2002) Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits. In: RAID.
11. Mutz D, Valeur F, Vigna G (2006) Anomalous System Call Detection. In: ACM Transactions on Information and System Security. 9: 61-93.
12. Lee T, Mody JJ (2006) Behavioral Classification. In: EICAR.
13. Kaufman L, Rousseeuw PJ (1990) Finding groups in data: An introduction to cluster analysis. New York: John Wiley & Sons.
14. Beyer K, Goldstein J, Ramakhrisnan R, Shaft U (1999) When is 'nearest neighbor' meaningful? In: 7th Int Conf on Database Theory (ICDT'99).

15. Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Kirda E (2009) Scalable, Behavior-Based Malware Clustering. In: Network and Distributed System Security Symposium (NDSS).

16. Davey BA, Priestley HA (1990) Introduction to lattices and order. Cambridge: Cambridge University Press.

17. http://vx.netlux.org/

18. Christodorescu M, Jha S, Kruegel C (2007) Mining Specifications of Malicious Behavior. In Proceedings of the ESEC/FSE'07 Dubrovnik Croatia.

19. Kolbitsch C, Comparetti PM, Kruegel C, Kirda E, Zhou X, et al. (2009) Effective and efficient malware detection at the end host. In USENIX Security Symposium, Montr´eal, Canada.

20. Strace for NT – support and services.