Forensic Intrusion Detection on System Logs

Karen A. García-Gamboa[1], Eduardo Aguirre-Bermudez[1], Raúl Monroy[1], and Carlos Mex-Perera[2]

[1] Tecnológico de Monterrey, Campus Estado de México
Carretera al Lago de Guadalupe Km. 3-5, Atizapán, Estado de México, 52926, Mexico
{kazurim,A00467871,raulm}@itesm.mx
[2] Tecnológico de Monterrey, Campus Monterrey
Av. Eugenio garza Sada 2501 Sur, Col. Tecnologico, Monterrey, N.L., 64849, Mexico
carlosmex@itesm.mx

Abstract. Computer forensics is often used to analyse an IT system after an intrusion in order to determine how the attacker gained access to a resource and what he did afterwards. Usually, it reveals that the attacker often runs a exploit to take advantage of a vulnerability in order to cause unintended system behaviour. Given a log file, we are interested in pinpointing the execution of an exploit, if any. Solving this problem is rather complex, given both the overwhelming length of a standard log file and the difficulty of identifying exactly where the intrusion has occurred. We introduce a novel approach on forensic intrusion detection using our approach together with a collection of methods. We shall show that forensic intrusion detection based on simple statistical control limits offers a reasonable detection rate.

1 Introduction

Computer forensics is concerned with explaining the current state of a computer or a digital storage medium by means of legal evidence.[3] It is used, for example, to analyse an IT system after an intrusion in order to determine how the attacker gained access to a resource and what he did afterwards. Forensic intrusion detection reveals that, often, the attacker runs a small program, called an *exploit*, which takes advantage of a system vulnerability in order to cause unintended system behaviour.

We are interested in pinpointing the execution of an exploit, if any, in a given log file. Solving this problem is crucial to a successful subsequent forensic analysis, because it enables the computer forensic scientist to focus only on system activity related to an intrusion. However, forensic intrusion detection is rather a complex goal, given both the overwhelming length of a standard log file and the difficulty of identifying exactly where the intrusion has occurred; it thus requires tool support.

In this paper, we take a few steps towards automating forensic intrusion detection. We introduce a novel approach on forensic intrusion detection. Our

[3] http://en.wikipedia.org/wiki/Computer_forensics

investments involved the application of several standard but well-recognised classifiers, namely: on-line k-means, non-negative matrix factorisation and hidden Markov models, and a few basic statistics, which we used to specify control limits.

Paper Overview The rest of this paper is organised as follows. §2 describes our general approach to forensic intrusion detection. Central to our approach is the use of SEQUITUR [1], which is used both for making a log file more manageable, by eliminating redundant information, and for improving detection, by capturing temporal aspects of log information [2]. §3 describes our experimental setting: how and to what extend we collected example system call log files, how we use them in the construction of classification models for intrusion detection, how we tested such models, and how we collected experimentation results. §4 gives a brief overview of the classification methods used in our experiments and details about the way we used them, in §5 we show a comparative table with the results obtained in each one of the applied methods. Finally, §6 concludes the paper and gives directions for further work.

2 Forensic Intrusion Detection: Overall Approach

We approach forensic intrusion detection using an anomaly,[4] host-based[5] intrusion detection model. We assume that an attacker has already bypassed the intrusion detection system, if any, and that there is a set of logs, where evidence of the intrusion has (hopefully) been recorded. Each log contains a record of the system calls that have been executed by the system due to a process request. Our aim is to develop a method capable of pinpointing in one such a log the execution of an exploit, if any, whereby an intruder has lifted his priviledges to those of a system administrator.

Building a successful forensic intrusion detection system (IDS) of this kind involves addressing two key issues. First, accuracy: the performance of an anomaly detection method highly depends on the accuracy of the profile that it uses to capture ordinary system behaviour. Building one such a profile is difficult in general, and worse in our case, because normal and abnormal system behaviour differ quite subtly and hence are difficult to put apart. Second, scalability: computer activity, even in a moderate computer, very quickly yields gigabytes of

[4] Based on the detection scheme, an intrusion detection system (IDS) is either *misuse* or *anomaly*. A *misuse IDS* aims to detect the appearance of a known attack. While accurate, misuse IDSs are often unable to detect variants of known attacks, commonly referred to as *mimicry attacks*, or attacks that exploit vulnerabilities recently discovered. To get around this problem, an *anomaly IDS* counts on a characterisation of ordinary activity and uses it to distinguish an abnormal one [3].

[5] Depending on the activity it observes, an IDS is either of three types: host, network or application. A *host IDS* is usually set to audit the functionality of the underlying operating system, as it executes system calls, but can also be set to watch critical resources.

information [4, 5]; hence, model construction and log analysis for forensic intrusion detection are at best very time consuming but usually overwhelming.

Our approach to address both issues relies on two key observations. First, although huge, log files contain a number of repetitive, spurious information, which gets on the way to a successfull forensic intrusion detection. Second, although forensic intrusion detection involves analysing a complete log file, the construction of a profile for ordinary behaviour might be achieved considering only some parts of it. We consider these two observations in turn below.

Figure 1 graphically explains the process followed in our experiments. First, we generated log files with ordinary and abnormal behavior, divided them into three streams of 30 000 system calls, reduced those streams with sequitur and then divide them into windows of length 100. After that data preprocessing, we obtained the values of 9 attributes for each window and then used those values as input of Online K means, NMF, Box Whiskers and Substractive Clustering along with Online K means, followed by HMM. After the application of the method we obtained a set of possible abnormal windows, we eliminated stand alone possible abnormal windows and possible abnormal set of windows without any dangerous system call. The output are the abnormal windows in a log file.

2.1 Factoring Out Repetitive Behaviour

Driven by our first observation, we shrink a log file identifying sequences of system calls of frequent occurrence and then replacing them all, each for a fresh meta-symbol. To this aim, we apply SEQUITUR [1], a method that infers compositional hierarchies in the structure of an input string. SEQUITUR detects repetition and factors it out of the input string by forming rules in a grammar. SEQUITUR rules out grammatical rules that share one or more digrams or that are used once only in the production of the string. These two constraints enable SEQUITUR to yield the shortest grammatical rule that can produce the input string.

SEQUITUR significantly reduces a system call log file. Throughout our experiments, we have obtained an average reduction factor of over 20 in attack-free log files; this implies an average log file reduction of over 95%. Compact log files account for a much faster construction of the detection model and for making forensic intrusion detection more tractable. This is because compacting log files alleviates the burden of analysing redundant information. Moreover, literature reports that it slightly increases the detection ratio but, more importantly, at the virtually no cost of increasing the false positive ratio only 1% higher [2]. Our experiments confirm this: we have found that the meta-symbols associated with ordinary behaviour are unique and highly repetitive.

These benefits do not come for free though. First, introducing meta-symbols to replace system call sequences increases the size of the alphabet, thus diluting the effectiveness of a classification technique. Second, when reducing information, crucial intrusion tracks might be taken away and thus they will not be observed by the forensic intrusion detection method.

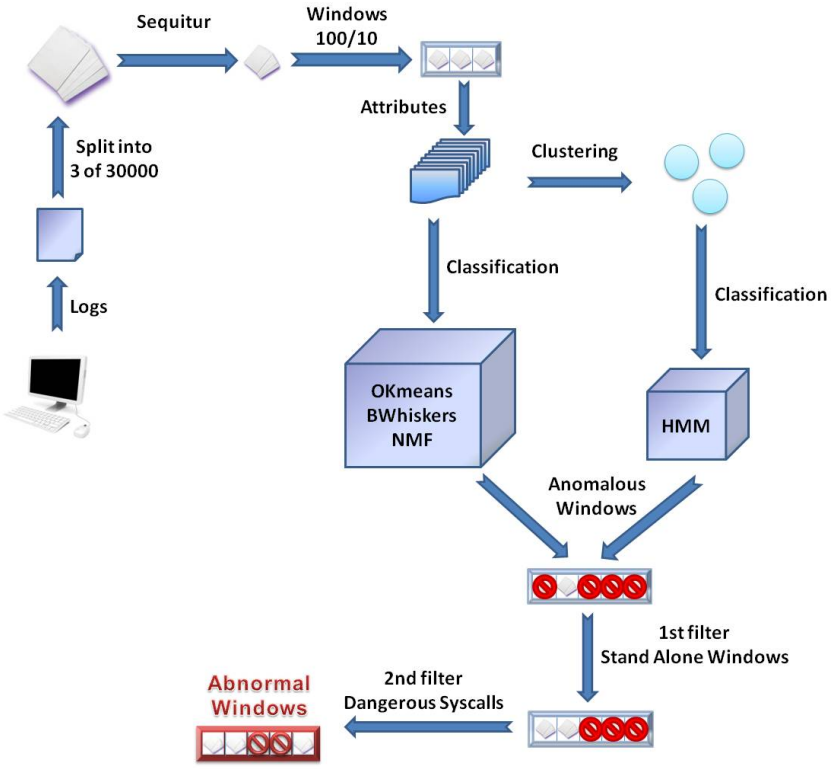

Fig. 1. Methodology

2.2 Good Enough is Enough

Repetition in ordinary behaviour significantly alleviates the burden of forensic intrusion detection. However, we have shortened this burden even further, using only part of a log in the construction of a profile for ordinary system behaviour. To establish that this approach yields a profile that is as good as another one built using the log fully, we applied the following, two-step procedure.

The first step aims at proving that the information entropy of both a log and any randomly selected block of it are very similar. To this end, we picked up a few attack-free logs at random. Then, for each selected log, we randomly selected several blocks of it, each of a different size. We computed the entropy of both the log and all its associated blocks. Throughout our experiments, we have found that these entropies do not differ significantly.

The second step aims at proving that the quality of the profile to be constructed would not be severely affected by taking a log portion only. We established this inputting both the log and their associated blocks, one at a time, to SEQUITUR. Again, throughout our experiments, we have found that the gram-

matical rules ouput by either of these inputs coincide largely, in average more than 90%.

3 Experimental Setting

For the purpose of our work, every one of our experiments were performed in a Linux distribution running on a virtual machine VMWare. We traced 10 log files divided into 2 categories; the first one is composed of 10 ordinary behavior logs from different users, the second are 3 log files with ordinary and attack behavior. An ordinary behavior log file is an attack-free log file, an attack behavior is produced when an attack is done. To capture our traces, we used a special Linux command: strace.

Ordinary log files were captured in the following linux distributions: RedHat Enterprise 4.0, Fedora 8.0, and Ubuntu 9.04. To capture attack behavior, we collected 3 user to root (U2R) attacks, two that exploit Fedora 8.0 vulnerabilities on vmsplice system call, and one exploits a RedHat 7.0 vulnerability on ptrace system call. Each log file traces a user session, or many user sessions in a computer. Each session has an average length of 8 hours.

Of each ordinary log file, we decided to use only a part of them which length is 90 000 system calls divided in three streams of 30 000 system calls. The streams were randomly extracted from the first, second and third part of the entire log file. To analise a log file, each stream of 30 000 system calls was reduced using Sequitur. The sequitur algorithm constructs a context-free grammar which extracts hierarquical structures that generate the stream of system calls. Furthermore, every reduced stream was divided on windows of 100 elements, and as step we slide a window of 10 elements. An element is composed by system calls and metasymbols, where a metasymbol is a stream subsequence which appears more than twice. Our approach depends on 9 attributes from each window in a reduced stream, those attributes are:

1. Maximum number of consecutive system calls.
2. Number of system calls.
3. Mean of succesfull system calls.
4. Mean of consecutive system calls.
5. Number of distinct system calls.
6. Maximum number of consecutive metasymbols.
7. Number of metasymbols.
8. Number of distinct metasymbols.
9. Window lenght without reduction.

To use these attributes we evaluated its usefulness trough an experiment. The results showed us that the attributes are useful for us. Therefore the attribute values were calcuated for every window and then used as input of 3 classification techniques to forensic intrusion detection, and 1 preprocessing set of techniques which will create the input of the last applied classification technique, classification techiques identify possible abnormal windows.

Because of the nature of our experiments an attack can not be in a stand alone window, this allow us to eliminate those windows of the set. In [6], dangerous system calls on a system were defined: *chmod*, *fchmod*, *chown*, *fchown*, *lchown*, *execve*, *mount*, *rename*, *open*, *link*, *symlink*, *unlink*, *setuid*, *setresuid*, *setfsuid*, *setreuid*, *setgropus*, *setgid*, *setfsgid*, *setresgid*, *setregid*, *create_module*. An attack must have at least one dangerous system call, therefore if there is not any dangerous system call in one of the resulting streams, it is eliminated from the set.

As a result of the experiment comparing the nine attributes in some of the ordinary and abnormal log files, we calculated measures like mean and standard deviation. These measures were useful in our analysis to identify abnormal windows on a log file. Therefore we classify the behavior of a reduced log file based on two limits, mean minus standard deviation and mean plus standard deviation of every one of the attributes, at that point we use two criteria: consider that a window is abnormal if it is marked as abnormal by one or more attributes and consider that a window is abnormal if it is marked as abnormal by half or more than the half of the attributes.

Finally one or more sequences of abnormal behavior windows are obtained for each one of the criteria applied, and are processed as explained at the end of the last section.In the following section we will briefly describe the main aspects of each classification method of our approach.

4 Overview of Classification Techniques

In this section we briefly describe the applied classification methods to the data. Each method has as output a set of possible abnormal windows. Because of the length of an attack, the window, and the step, it is impossible that an attack be contained just in a single window, that allows us to consider that the possibly abnormal stand alone windows are ordinary windows. As a result of the previous process every set of possibly abnormal windows has a length of at least two windows.

4.1 Hidden Markov Models

An HMM is a stochastic model of discrete events and a variation of the Markov chain. An HMM consists of a set of discrete states and a matrix $A = a_{ij}$ of state transition probabilities. Every state has a vector of observed symbol probabilities, $B = b_j(v)$ that corresponds to the probability that the system will produce a symbol of type v when it is in state j. The states of the HMM are inferred from the observed symbols.

On our experiments we used HMM as a classfication method to forensic intrusion detection. To use this method we first did a little preprocessing to the data: at this point we have 9 attributes that represent the behaviour of our normal logs, we used *subtractive clustering* to obtain the number of clusters that we used as a parameter to Online K means, as a result of applying the

last technique we obtained a log made of numbers that represent the cluster to which the data that used to be there belongs. Substractive clustering uses a distance measure to calculate the ideal number of clusters of some data based on euclidean distance. Online K means classify data in clusters using euclidean distance.

We used different inputs for substractive clustering, so we obtained different number of clusters for each input. Each output log considers a different number of clusters. These logs are the input for our HMM's.

After this preprocessing, we construct an ordinary behavior model by learning the probability of each element of ordinary behavior emerging from each node and the probability of each transition between nodes. There are some tools wich implement a Hidden Markov Models, we used HTK [7].

Transitions are made successively from a starting to a finishing node, and the transition and elements of ordinary behavior probabilities can be multiplied at each transition to calculate the overall likelihood of all the output elements produced in the transition path until that position. When all transitions are finished, the Hidden Markov Model generates an element sequence according to the likelihood of a sequence being made along each path.

We choose a number of states corresponding to the number of clusters generated. To train our model we used ordinary behavior cluster streams which were the base for the model construction. The testing phase was divided into two steps, we first randomly chose some cluster streams used in the testing phase to calculate a threshold that belongs to ordinary behavior. In the second step ordinary and abnormal cluster streams were tested using the HMM model. The output of the second step is a probability of belonging to the HMM model, if this probability is less than the threshold of ordinary behavior it is labeled as abnormal, otherwise as ordinary.

Notice that we used streams; which are smaller than a log, this is done to make easier the localization of an attack. we divided a log into streams, if a stream doesn't belong to the initial model, it is considered abnormal and we assume that it contains an attack execution.

4.2 Non-Negative Matrix Factorization

Non-Negative Matrix Factorization or NMF is a method developed by Lee et. al in [8], where a matrix X is factorized into two matrices W and H, this method assures that the factors W and H must be non negative. The algorithm is defined formally like, give a database represented by an $n \times m$ matrix V, where each column is an n-dimentional non-negative local vector belonging to the original database (m local vectors), n is the number of samples and m is the number of attributes of each sample. It finds two matrices W and H, we obtain an approximation of the whole database V by:

$$V_{i\mu} \approx (WH)_{i\mu} = \sum_{a=1}^{r} W_{i\alpha} H_{\alpha\mu} \tag{1}$$

Where the dimensions of the matrix W and H are n x r and r x m, respectively. Normally, r is chosen so that $(n + m)r < nm$. Each column of matrix W contains a basis vector while each column of H contains encoding coefficients needed to approximate the corresponding column in V.

In order to use the NMF algorithm, we introduced training ordinary log files as the input and the frecuency of individual elements in each window of length 100 is counted. The NMF method uses the values of elements frequencies in each window as entries for matrix V, where $V_{i\mu}$ is the number of times the i-th element appears in the μ-th window. Matrix V can be factorized into W and H. The columns of W represent the basis profiles and the columns of H are the encoding coeficients of ordinary behavior.

If the features contained in any window deviate significantly from those of the ordinary behaviors learned in the training datasets, the delta value will be bigger, meaning a big difference between the behaviors. We used each one of the test windows as the second input, obtaining deltas whose values were the similarity between the test windows and normal behavior. Using the normal behavior deltas, we calculated the median and everything above it was considered an abnormal behavior.

4.3 Box Wiskers

This is a very graphic method helpful in finding outliers (limits) [9], to calculate them we obtained the first, second and third quartile values, and the inter quartile range (IQR). Once we calculated those data, the outliers are:

$$Lowerlimit = Q2 - IQR * 1.5 \tag{2}$$

$$Upperlimit = Q2 + IQR * 1.5 \tag{3}$$

The constant value 1.5 means that the whiskers extend to at most 1.5 times the box width (IQR) from either the line inside the box that is the median of the data. The data beyond those whiskers is atypical, which means that if we use this method to represent ordinary behavior it is possible to detect abnormal behavior that must be atypical.

For this method we calculated the outliers of each attribute of ordinary behavior and then used those limits to detect abnormal behavior that are all the atypical windows, as a result of this experiment we obtained a set of possible abnormal windows by each analyzed attibute, these windows were processed as written in the las part of section 3 to obtain as a final output of this method a set of abnormal windows.

4.4 Online Kmeans

The *K-Means* algorithm was introduced in [10]. It is a clustering method which dispatches K centroids $w(k)$ in order to find clusters in a set of points $x_1, ..., x_L$. K-means use the distance to the center of each one of the clusters, the nearest

center is the cluster where the data is classified into. This distance is calculated by the similarity between the data. This algorithm can be derived by performing the online gradient descent with the following loss function.

$$Q_{kmeans}(x, w) \triangleq min_{k=1}^{K}(x - w(k))^2 \tag{4}$$

This loss function measures the quantification error, that is the error on the position of point x when we replace it by the closest centroid. The corresponding cost function measures the average quantification error. The algorithm is basically composed of the following steps:

1. Place K points into the space represented by the objects that are being clustered. These points represent the initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

There are a lot of k means mutations, because of this we define the characteristics of the algorithm we used: euclidean distance, two clusters and initialization with data of the corresponding cluster. We trained the algorithm with all logs traced wich contain ordinary and abnormal behavior and we select two random centroids, each one for a different behavior. At the end of the trainning phase, we obtained two new centroids. We tested with the same input of our training but we used the new centroids and this method shows that each window bellongs to a cluster that contains a specific behavior. The input data were the 9 selected attributes per window. The result of the algorithm is the classification of the data by its 9 attributes.

5 Experimental Results

Table 1 compares the results of the proposed methods and STIDE, STIDE is an IDS [11] proposed by Forrest et al that uses n-grams and the system calls generated by an exploit to detect an attack in an specific application. STIDE is one of the most recognized IDS's in the scientific field because of its good detection rate, and it is also the state of the art in system calls based intrusion detection. However in our experiments it does not have a good detection rate, probably to the fact that we are trying to detect an attack in a whole system, not just in an application.

As we can se in the table, the method that had the better results is BW and Mean S.D., but some other methods also obtained good results, however STIDE had a poor performance, this table allow us to say that our methods are a good approach to achieve a good detection.

Table 1. Methods Results

	False Positive (%)	False Negative (%)	Detection Rate (%)
BW (Any Att.)	40	0	100
BW (Half or More Att.)	100	100	0
Mean and S. D. (Any Att.)	52.57	3.33	96.66
Mean and S. D. (Half or More Att.)	9.87	35.37	64.62
BW & Mean S.D. (Any Att.)	63.33	0	100
BW & Mean S.D. (Half or More Att.)	10.68	23.7	76.29
NMF	50	75	25
HMM	57.4	12.6	87.4
Online KMeans (Any Att.)	74.44	13	87
Stide (Any Att.)	42.33	71.29	28.7

6 Conclusions and Future Work

The methods we used for classifying behaviors in a log file are a good approach in the isolation and automate model creation of an attack, which is a future work to do.

Most of the presented approaches had a better performance than STIDE and looking at the results table the detection rate is very acceptable, the errors on the detection are just a few and this can be improved in further versions.

The further work is to experiment with the proposed methodologies, completely isolate the attack and the automatic generation of its model, this model will be the input of an IDS capable to detect the attack and any mimicry modification of it.

References

1. Nevill-Manning, C.G., Witten, I.H.: Identifying hierarchical structure in sequences: a linear-time algorithm. Journal of Artificial Intelligence Research, JAIR **7** (1997) 67–82
2. Godínez, F., Hutter, D., Monroy, R.: Audit File Reduction Using N-Gram Models (Work in Progress). In Patrick, A., Yung, M., eds.: Proceedings of the Ninth International Conference on Financial Cryptography and Data Security, FC'05. Volume 3570 of Lecture Notes in Computer Science., Roseau, The Commonwealth of Dominica, Springer-Verlag (2005) 336–340
3. Kim, J., Bentley, P.: The Human Immune System and Network Intrusion Detection. In: Proceedings of the 7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany, ELITE Foundation (1999)
4. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1999) 133–145
5. Yeung, D., Ding, Y.: Host-Based Intrusion Detection Using Dynamic and Static Behavioral Models. Pattern Recognition **Vol. 36**(No. 1) (2003) pp. 229–243

6. Xu, H., Du, W., Chapin, S.J.: Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. In: In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID, Springer (2004) 21–38
7. Young, S., Evermann, G., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P.: The HTK Book for HTK Version 3.2. Cambridge University Engineering Department (December 2002)
8. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755) (October 1999) 788–791
9. Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers, 4th Edition, and JustAsk! Set. John Wiley & Sons (May 2006)
10. Bottou, L.: Stochastic learning. In: Advanced Lectures on Machine Learning. (2003) 146–168
11. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: In IEEE Symposium on Security and Privacy, IEEE Computer Society (1999) 133–145