A Pattern Matching Based Filter for Audit Reduction and Fast Detection of Potential Intrusions

Josué KURI[1], Gonzalo NAVARRO[2], Ludovic MÉ[3], and Laurent HEYE[3]

[1] ENST, Department of Computer Science and Networks, Paris, France,
kuri@enst.fr
[2] University of Chile, Department of Computer Science, Santiago, Chile,
gnavarro@dcc.uchile.cl
[3] Supélec, Campus de Rennes, France,
Ludovic.Me@supelec.fr, Laurent.Heye@supelec.fr

Abstract. We present a pattern matching approach to the problem of misuse detection in a computer system, which is formalized as the problem of multiple approximate pattern matching. This permits very fast searching of *potential* attacks. We study the probability of matching of the model and its relation to the filtering efficiency of potential attacks within large audit trails. Experimental results show that in a worst case, up to 85 % of an audit trail may be filtered out when searching a set of attacks without probability of false negatives. Moreover, by filtering 98 % of the audit trail, up to 50 % of the attacks may be detected.

Introduction

Research in intrusion detection has emerged in recent years as a major subject in the computer security field because of the difficulty of ensuring that information systems are free from security flaws. Computer systems suffer from security vulnerabilities regardless of their purpose, manufacturer or origin. It is both technically hard and economically costly to ensure that systems are not susceptible to attacks.

Two approaches have been proposed to address the problem: *anomaly detection* (see for example [1, 2]) and *misuse detection* (see for example [3]). The former suggests that user's activity in the system can be characterized so that a profile of "normal utilization" of the system is established and excursions from this profile are flagged as potential intrusions, or attacks in a more general sense. The latter assumes that attacks are well-known sequences of actions, called *scenarios* or *attack signatures*, and that the activity of the system (in the form of logs, network traffic, etc.) may be audited in order to determine the presence of such scenarios in the system.

Anomaly detection leads to some difficulties: a flow of alarms is generated in the case of a noticeable systems environment modification and a user can slowly change his behavior in order to cheat the IDS. On the other hand, misuse detection becomes an increasingly demanding task in terms of semantics

and processing, as more sophisticated attacks are discovered every day (which implies an increasing number of sophisticated scenarios to search for in audit trails). These challenges have lead to a research trend aimed to a simplified representation of the problem in order to improve performance and efficiency of detection. In the short term, effective intrusion detection systems will incorporate a number of techniques rather than a "one-strategy-fits-all" approach. The greater the variety of available tools is, the better the IDS is.

In this spirit, we introduce an original intrusion detection model inspired by the misuse detection approach. Its main goal is to provide an intrusion detection system for *fast* detection of *potential* attacks rather that accurate (i.e., exhaustive) detection of actual attacks. The results of such a detection (i.e., filtered audit trails, in which attacks may be present) would be used in turn as input for a more accurate detection algorithm. This idea was already at the root of the $G^A{}_S SA_T A$ IDS, which use a genetic algorithm with this aim in view [4].

We formalize a concrete instance of the misuse detection problem as a pattern matching problem which permits very fast searching of potential attacks. We then study the statistics of this model and their relation to filtering efficiency of potential attacks in the resulting system.

Section 1 explains our proposed intrusion detection model and the constraints of the problem. Section 2 gives analytical and experimental results on the probability of matching. Section 3 presents our testing system and experimental results. Finally, conclusions and future works are presented.

1 Intrusion Detection as a Pattern Matching Problem

In general terms, the misuse detection problem is to detect the existence of *a priori* known series of events within the traces of activity of a system to protect.

Traces widely differ in their origin, form and content, depending on the type of potential attacks that they attempt to cover. For example, traces in the form of network traffic collected by a firewall or a sniffer may be used to detect well-known attacks to implementations of a TCP/IP protocol stack. Another example are the logs of commands typed by users of a multi-user computer. In both cases, traces may be collected at a single place (e.g., an ethernet segment, a host computer) or at multiple locations simultaneously. We consider the detection of attacks using logs (audit trails) of commands typed by users of a distributed computer system. In this context, attacks appear to be typically short sequences of no more than 8 commands.

We propose to model the misuse detection problem as a pattern matching problem in the following way: auditable commands in the system can be seen as letters of an alphabet Σ and the audit trail as a large *string* of letters in Σ (i.e., the text). The sequences of events representing attacks to be detected are then *substrings* (i.e., patterns) to be located in the main *string*. Since attackers may introduce spurious commands among those that represent an actual attack in order to disperse their evidence, a limited number of spurious letters must be allowed when searching the pattern. We are interested in simultaneously

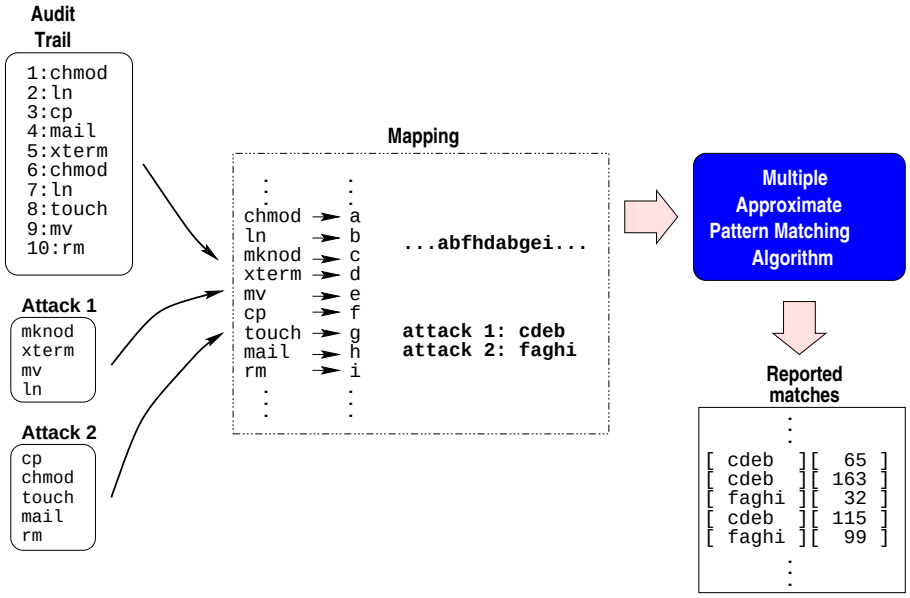

Fig. 1. Attack signature searching as a multiple approximate pattern matching problem.

searching a *set* of patterns. Thus, the misuse detection problem can be regarded as a particular case of the *multiple approximate pattern matching problem*, where *insertion* in the pattern is the only allowed edit operation. Figure 1 illustrates our proposed model to map the misuse detection problem as a multiple approximate pattern matching problem. The set of commands in the audit trail and attack signatures is translated into letters of Σ. The resulting string and patterns are passed to a multiple approximate pattern matching algorithm which in turn searches for the occurrences of substrings in the main string. Matches represent potential attacks in the audit trail.

We formalize the above problem as follows: Our text, $T_{1..n}$, is a sequence of n characters from an alphabet Σ of size σ. Our patterns, $P^1 \dots P^r$, are (short) sequences of characters from Σ. Let us consider such a pattern $P_{1..m}$ of length m. We want to report all the text positions that match P, where at most k insertions between characters of P are allowed in its occurrence in T. We call $\alpha = k/m$ the "error level".

Our problem can be modeled using the concept of *insertion distance*. The insertion distance from a to b, denoted $id(a, b)$, is the number of insertions necessary to convert a into b. We say that $id(a, b) = \infty$ if this is not possible. Clearly, $id(a, b) = |b| - |a|$ if a is a subsequence of b, and ∞ otherwise.

We search for the pattern P in a text T allowing insertions. At each text position $j \in 1..n$ we are interested in the minimum number of insertions needed

to convert P into some suffix of $T_{1..j}$. This is defined as

$$lid(P, T_{1..j}) = \min_{j' \in 1..j} id(P, T_{j'..j}) \ .$$

The search problem can therefore be formalized as follows: given $P^1 \ldots P^r$, T and k, report all text positions j such that $lid(P^i, T_{1..j}) \leq k$ for some $i \in 1 \ldots r$.

In [5], two different multipattern search algorithms specifically tailored for this pattern matching problem are presented. They are based on "bit-parallelism", a technique to represent the state of the search using the bits of a computer word of w bits (typically $w = 32$ or 64). The basic algorithm takes $O(nm \log(k)/w)$ time to scan the text for one pattern. A first multipattern search algorithm is $O(nr(1+\alpha)^{1+\alpha}/(\sigma\alpha^{\alpha}))$, which is better than r applications of the basic algorithm whenever $\alpha < (\sigma/e) - 1$. A second multipattern search algorithm takes $O(nr \log(m+k)/w)$ time and is useful for $\alpha < (\sigma/m) - 1$.

It is shown in [5] that the algorithms can achieve impressive scanning speeds in practice. For example, they show the case of 4-letters patterns searched allowing 4 insertions, which is a case of interest in intrusion detection applications. On a Sun Enterprise 450 server, these algorithms allow searching for 100 patterns at a rate of 4 megabytes per second.

The focus of this paper lies in the probabilistic model for the occurrences of patterns in text when insertions are allowed, and its relation to the problem of false detection of nonexistent attacks (i.e., "false positives") and misdetection of existing attacks (i.e., "false negatives"). By characterizing this relation, the optimal filtering efficiency of the model may be determined.

2 Probability of Matching

We start by giving an upper bound on the matching probability of a random pattern of length m at a random text position, with up to k insertions. Consider a random text position j. The pattern P appears with k insertions at a text position ending at j if and only if the text window $T_{j-m-k+1..j}$ contains the m pattern letters in order. The window positions that match the pattern letters can be chosen in $\binom{m+k}{m}$ ways. Those letters are fixed but the other k can take any value. Therefore the probability that the text window matches the pattern with k insertions is at most

$$\binom{m+k}{m} \frac{\sigma^k}{\sigma^{m+k}} \quad = \quad \binom{m+k}{m} \frac{1}{\sigma^m} \tag{1}$$

where we are overestimating because not all the selections of window positions give different windows. For instance the pattern `"abcd"` matches in text window `"abccd"` with $k = 1$ in two ways, but only one text window should be counted. In particular, our overestimation includes the case of $k' < k$ insertions, which is obtained by selecting the first $k - k'$ characters of the text window as insertions and distributing the k' remaining insertions in the remaining text window of length $m + k'$.

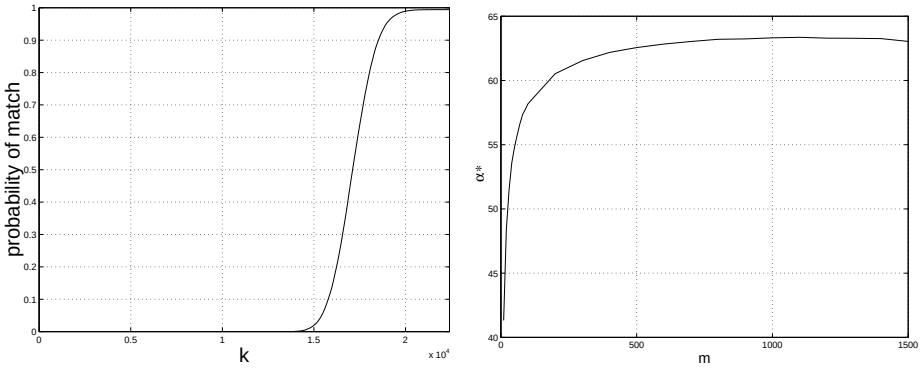

Fig. 2. On the left, matching probability for increasing k values and fixed $m = 300$. On the right, the α^* limit as m grows.

An asymptotic simplification (for large m and $\alpha = k/m$ considered constant) of the cost can be obtained using Stirling's approximation to the factorial $m! = (m/e)^m\sqrt{2\pi m}(1 + O(1/m))$:

$$\left(\frac{(1+\alpha)^{1+\alpha}}{\sigma\alpha^{\alpha}}\right)^m \tag{2}$$

which, as α moves from zero, grows from $1/\sigma^m$ to 1. To determine where the probability reaches 1, we require that $\sigma\alpha^{\alpha} \leq (1+\alpha)^{1+\alpha}$, i.e., $\sigma \leq (1+\alpha)(1+1/\alpha)^{\alpha}$. A sufficient condition can be obtained by noticing that $1 \leq (1+1/\alpha)^{\alpha} \leq e$, and therefore $\alpha \geq (\sigma/e) - 1$ suffices.

This means that a model based on insertions can be useful only if we keep k reasonably low, i.e., $k < m((\sigma/e) - 1)$. However, this is a pessimistic analytical model that needs experimental verification.

We test experimentally the probability that a random pattern matches at a random text position. We generated a random text and 100 random patterns for each experimental value shown. Figure 2 (left) shows the probability of matching in a text of 3 Mb for a pattern with $m = 300$, where pattern and text were randomly generated over an alphabet of size $\sigma = 68$. The reason to choose such a long pattern is given shortly.

As can be seen, there is a k value from where the matching probability starts to grow abruptly, moving from almost 0 to almost 1 in a short range of values. This phenomenon is sharp enough to make this k value the most important parameter governing the behavior of the algorithm. We call k^* this point, and $\alpha^* = k^*/m$ the corresponding error level.

On the right part of figure 2 we have shown this limiting α^* value for different pattern lengths, showing that α^* tends to a constant for large m, despite that it is smaller for short patterns. The fact that α^* tends to a constant limit when m grows motivated us to use $m = 300$ to show the process at a stable point. On the other hand, it must be noted that the limit is much lower for short patterns than

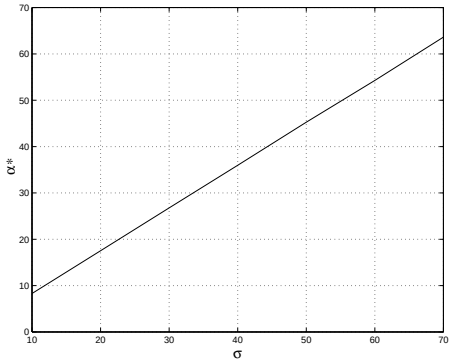

Fig. 3. The α^* limit as σ grows.

its asymptotic value, and therefore the exact combinatorial formula of Eq. (1) should be preferred, leaving Eq. (2) just as a conceptual tool to understand how the process behaves in general.

Finally, we show in figure 3 how the alphabet size σ affects the α^* value. As can be seen, the curve looks as a straight line, where least squares estimation yields $\alpha^* = (\sigma/1.0856) - 0.8878$. Again, this corresponds to long patterns, while the real values for short patterns should be obtained from the exact formula.

All this matches our analytical results in the sense that (a) there is a clear error level α^* where the matching probability goes almost from 0 to 1; (b) this point does not depend on m asymptotically; and (c) it depends on σ linearly as predicted by the analysis ($\alpha^* = (\sigma/e) - 1$) except because the e has been changed to about 1.09. Interestingly, this is similar to the result obtained for the k differences problem in [6,7] when relating their analytical predictions ($\alpha^* = 1 - e/\sqrt{\sigma}$) with the experiments ($\alpha^* = 1 - 1.09/\sqrt{\sigma}$) and shows a consistent behavior of the pessimistic analytical model used in both cases.

3 Experimental Results

We experimentally studied how the probabilistic model of string matching allowing insertions relates to the problem of false negatives and positives. Our interest is to determine how α^* relates to the ratio between false negatives and positives and the total number of reported attacks and, consequently, to the filtering efficiency of the model.

The experimental input data consists of an audit trail and an attack database. Both of them are very simple. The audit trail was collected using the $\mathrm{G^A_S SA_T A}$ IDS in a real environment. The format of events given to $\mathrm{G^A_S SA_T A}$ is for the moment an extension of the one proposed in [8]:

```
#S#version=suntrad5.6#system=SOLARIS#deamon=system#ahost=amstel#no=28#
event=AUE_EXECVE#date=2000.3.14@14.29.41#program=/var/audit/ls#
```

```
file=/var/audit/ls#euid=root#egid=other#ruid=root#rgid=other#pid=13949#
error=-1#return=KO#E#I#

#S#version=suntrad5.6#system=SOLARIS#deamon=system#ahost=lancelot#no=29#
event=AUE_EXECVE#date=2000.3.14@14.29.41#program=/usr/bin/ls#
file=/usr/bin/ls#arg=ls,-als#euid=root#egid=other#ruid=root#rgid=other#
pid=13949#error=0#return=OK#E#I#
```

The attack database consists of attacks signatures with the following format:

```
>>> Attack_login
rule1
rule1
rule1
>>> Attack_file_creation
rule2
>>> Attack_ps_cmd
rule3
rule7
```

Rules are defined in the following way:

```
rule1 ::= ( (event=AUE_login)||(event=AUE_rlogin) ) && (return=KO) ;
rule2 ::= (event=AUE_CREAT) && ( (file co ls)||(file co cd) ) ;
rule3 ::= (event=AUE_EXECVE) && (program=/usr/bin/ps) ;
rule4 ::= (event=AUE_EXECVE) && (program co crack) ;
rule5 ::= (event=AUE_su);
```

where the co operator stands for "**co**ntains".

The audit trail and attack signatures are translated into a pattern matching representation in three steps. First, a different letter is assigned to each rule (e.g., rule1 = 'a'). Attack signatures are then translated into patterns by mapping their rules to the corresponding letters. Finally, the audit trail is scanned and its events are matched against the rules. Events which match more than one rule are assigned the corresponding letters. Events which do not match a rule are assigned arbitrary letters. The final string is constructed by concatenating the sequence of letters corresponding to matches of rules and the arbitrary letters.

We used an audit file of 24,847 events and studied three different series of actions[1]:

Chained who: represented as a pattern of four events of a "who" command. The probability of the corresponding letter in our audit file is 0.004382 and there are four real attacks of this kind in the audit file.

[1] These are not really attacks, but it makes no difference from the algorithmic point of view.

Attack	m	Occs.	Prob. letter	Nec. k	Max. k	Fract. of text
Chained who	4	4	0.004382	225	500	8.21%
Sensitive commands	10	2	0.007187	580	620	14.50%
Chained whois	4	1	0.001402	1425	1570	5.74%

Table 1. Main parameters for the three search patterns.

Sensitive commands: represented as a pattern of ten events of any command in the set { `"last"`, `"ps"`, `"who"`, `"whois"` }. The probability of the corresponding letter in our audit file is 0.007187 and there are two real attacks of this kind in the audit file.

Chained whois: represented as a pattern of four events of a `"whois"` command. The probability of the corresponding letter in our audit file is 0.001402 and there is one real attack of this kind in the audit file.

We have searched the three patterns in our audit file allowing an increasing number of insertions k. Our goal is to determine the effectiveness of the proposed filtering algorithm[2]. That is: how much text is it able to filter out in order to retrieve what fraction of the real attacks that occur in the audit file?

By applying the analytical predictions of Section 2 to our real data, we computed the maximum k value for which the matching probability does not reach 1 (recall that the model is pessimistic). To compute that maximum value, we have used the most precise formula (Eq. (1)) for the matching probability. Given that the text is biased we have replaced $1/\sigma^m$ by p^m, where p is the relative frequency of the letter that forms the pattern (all the attacks are repetitions of a single letter, otherwise we can just multiply the probabilities of the participating letters).

Together with the maximum k recommended by the model we have computed the fraction of the text that the filter selects (for that k) as a candidate for further evaluation. This is simply the $m+k$ characters preceding every match, avoiding to count multiple times the overlapping areas.

Table 1 shows that using the maximum k recommended by the model selects just 6% to 15% of the text to be processed by a more costly algorithm. Moreover, we show in the column of "necessary k" the minimum k value that is necessary to detect all the attacks present in the audit file. This turns out to be below (and generally close to) the maximum k recommended by the model. Therefore, the model can be used to obtain a good estimator of the k value to use in order to detect all the real attacks. Of course, it is also possible to use specific knowledge

[2] The text that our filter is not able to discard has to be processed by a more sophisticated algorithm in order to determine the presence of a real attack. As those algorithms are much slower than our pattern matching based approach, the effectiveness of the filter is crucial.

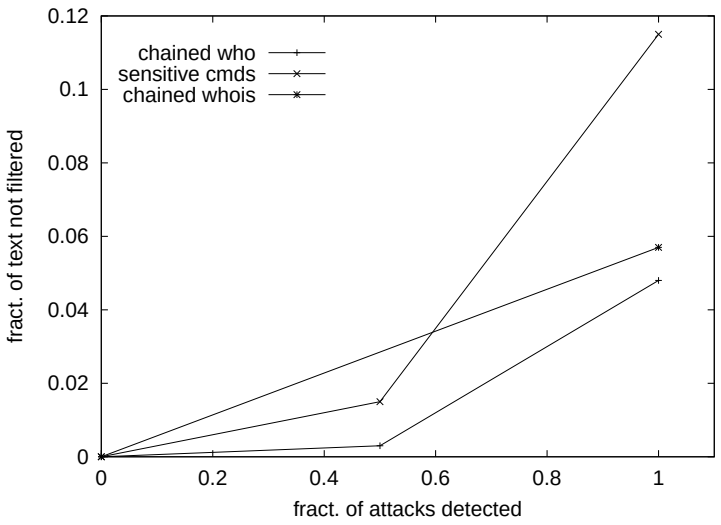

Fig. 4. Fraction of attacks detected versus fraction of text left for further processing.

of the application to determine the appropriate k.

Regarding the false negatives, we evaluated, for our three particular patterns, the fraction of text filtered as a function of the fraction of attacks detected (see figure 4). As can be seen, the curve is concave, which suggests that considering a very small fraction of the text permits to detect most of the attacks. For example, with a k value that leaves just 2% of the text for further evaluation we get 50% of the attacks (and thus 50% of false negatives). We have here a way to balance the false negatives rate and the speed of detection. Of course, in many cases, no false negative is required. In that situation, the value of k determined by the model is an upper bound of the value to be used for the corresponding pattern.

Regarding the false positives, we studied the evolution of the number of matches as a function of k for the three patterns (se figure 5). Of course, for some patterns, using a too large k value leads to many false positives. Let's note that these false positives may be discarded by the more accurate detection algorithm which may analyse the output of our pattern matching mechanism (recall that we give a part of the trail containing a *potential* attack). To limit false positives without allowing false negatives, the value of k determined by the model appears as a near-optimum.

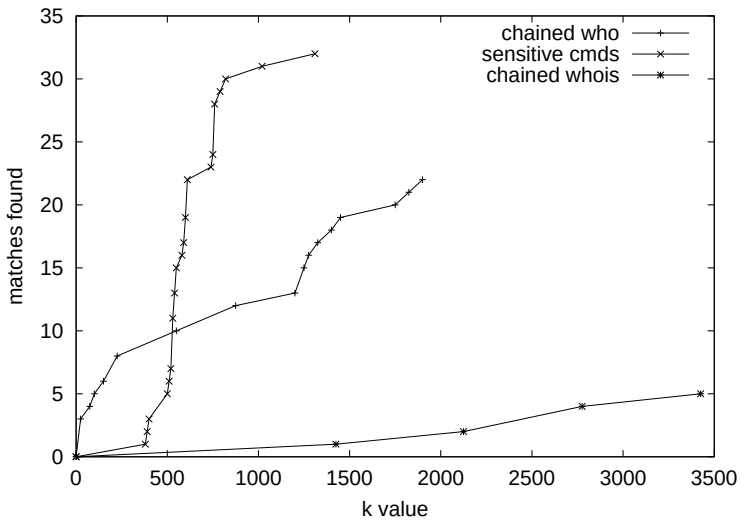

Fig. 5. Number of matches found as a function of k.

Conclusions and Future Work

We have addressed a performance problem in intrusion detection. The problem is that the algorithms that accurately detect attacks in audit trails are complex and slow, and therefore can not cope with huge amounts of data that are generated when a system or a network is monitored.

We have presented a pattern matching based filter for intrusion detection. The idea is that, despite that it may be difficult and slow to determine that an attack has occurred, it is possible to quickly determine that there is no attack in large portions of the audit trail.

The pattern matching model is based on the concept of *insertion distance*, where the attack is seen as a sequence of letters (events) and the algorithms detect the text portions where all the events of the attack appear in order within a window of k other events. Recent pattern matching algorithms [5] specialized for this problem are able to spot the suspicious areas of the audit trail by scanning millions of events per second. In this way, the pattern matching algorithm quickly *filters out* a large portion of the text, leaving the rest to be examined by a more sophisticated (and slower) algorithm.

We have presented an analytical model and preliminary experimental results about the fitness of the insertion distance model to detect attacks in a real application. We have shown that the k value predicted by the model is large enough to detect virtually all the relevant attacks, yet it still filters out most of the text (85% to 94%). This means that only 6% to 15% of the original text needs to be analyzed by a more sophisticated algorithm. Moreover, we have shown that most of the attacks are indeed found with a much smaller k value, so that speed

can be traded for precision. For example, leaving just 2% of the text to examine we were able to detect 50% of the attacks. An optimal value for k can be found which minimizes false negatives and false positives for groups of patterns with specific characteristics.

Some work to undergo to improve the proposed approach follows.

Further experiments with larger and more realistic data sets must be carried out in order to provide more accurate estimations of the filtering efficiency.

The algorithm is assumed to take a random text with uniform distribution as input, which is not the case of our converted audit trails. The study of the implications of that fact is to be done.

Our experiments were conducted off-line. We now need to conduct some on-line experiments. In that context, the efficiency of the mapping process will have to be studied with care.

To conclude, the proposed approach was used for misuse detection. It could be also used for anomaly detection. The algorithm may then be used to verify that a process behaves as during a training period, as proposed by [9].

References

1. J.P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.
2. Teresa F. Lunt. A survey of intrusion detection techniques. *Computers and Security*, 12, 1993.
3. T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, October 1991.
4. Ludovic Mé. Gassata, a genetic algorithm as an alternative tool for security audit trails analysis. In *First international workshop on the Recent Advances in Intrusion Detection*, 1998.
5. J. Kuri and G. Navarro. Fast multipattern search algorithms for intrusion detection. Technical Report TR/DCC-99-11, Dept. of Computer Science, Univ. of Chile, December 1999. To appear in *7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, IEEE CS Press.
6. R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.
7. G. Navarro. *Approximate Text Searching.* PhD thesis, Dept. of Computer Science, Univ. of Chile, December 1998. Technical Report TR/DCC-98-14. `ftp://-ftp.dcc.uchile.cl/pub/users/gnavarro/thesis98.ps.gz`.
8. M. Bishop. A standard audit log format. In *Proceedings of the 19th National Information Systems Security Conference*, pages 136–145, October 1995.
9. P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: Algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy.* IEEE Computer Society, IEEE Computer Society Press, May 1996.