Motif-oriented Representation of Sequences for a Host-based Intrusion Detection System

Gaurav Tandon, Debasis Mitra, and Philip K. Chan

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901, USA
{gtandon, dmitra, pkc}@cs.fit.edu

Abstract. Audit sequences have been used effectively to study process behaviors and build host-based intrusion detection models. Most sequence-based techniques make use of a pre-defined window size for scanning the sequences to model process behavior. In this paper, we propose two methods for extracting variable length patterns from audit sequences that avoid the necessity of such a pre-determined parameter. We also present a technique for abstract representation of the sequences, based on the empirically determined variable length patterns within the audit sequence, and explore the usage of such representation for detecting anomalies in sequences. Our methodology for anomaly detection takes two factors into account: the presence of individual malicious motifs, and the spatial relationships between the motifs that are present in a sequence. Thus, our method subsumes most of the past works, which primarily based on only the first factor. The preliminary experimental observations appear to be quite encouraging.

1 Introduction

In this work we propose a technique for intrusion detection from audit logs. Audit logs of a hosting computer store process activities at the system call level, which are typically analyzed to study the behavior. Various approaches have been proposed to detect intrusions based upon identifying deviations from normal traffic at a host computer. Forrest et al. (1996) proposed an approach in which they formed correlations between audit events within a fixed window size. Lane and Brodley (1997a, 1997b) suggested techniques for capturing the user profiles. They calculated the degree of similarity between two different audit sequences by looking into adjacent events within a window of ten events. The main limitation of such approaches is that an attack may be spread beyond such a pre-determined window.

In this paper, we attempt to address the problem (of using a fixed window-size) by extracting variable length patterns (subsequences or motifs) from the audit sequences. We then transform the audit event-sequence to a pattern-based representation. This also results in some data reduction and is expected to simplify the task of the "similarity" finding algorithm. Another motivation behind developing this representation scheme is not only to detect intrusions but also to model different types of normal or abnormal behaviors. Moreover, most of the anomaly detection systems

rely on the presence/absence of some events that represent the abnormal behavior. But an attack may not always be such a novel event. Rather an attack may constitute normal events placed in a "wrong" order. We conjecture that the relative positions of some frequently observed subsequences (we call them *patterns, sub-strings* or *motifs* synonymously) within a sequence of system calls model the behavior of a sequence.

The paper is organized as follows. In Section 2, we describe some system call sequence-based approaches in IDS's. Section 3 presents our approach for detecting anomalies, and the steps involved in our experiments. In Section 4, we summarize the results obtained from the experiments with some real data. We also analyze the results and discuss certain related issues there. A brief description of the 1999 DARPA evaluation data set is also presented. Section 5 addresses some views and the future direction of our work.

2 RELATED WORK

Lane and Brodley (1997a, 1997b) examined sequences of user actions and explored various matching functions to compare the behaviors of sequences with the various user profiles. Their matching functions measured similarity of adjacent audit events within a fixed size-window. They empirically found the window-length 10 to be providing better results. Forrest et al. (1996) also proposed an approach for host based anomaly detection wherein the traces of normal executions were considered and the subsequences were recorded by creating a look-ahead window. These subsequences were then used to create a database. All subsequences in the test cases (using the same window size and look-ahead) were checked for consistency with the training database. If the number of mismatches was above a threshold for a sequence it was deemed anomalous. One of the issues involved in their approach was the use of a small window that does not correlate well over a long period of time. Similar sequences with minor variations could still be flagged as anomalous producing a very high number of false positives. Both these techniques have arbitrarily fixed the window length that could lead to an easy evasion by an attacker. Our technique detects *all* frequently occurring subsequences of variable lengths, thereby removing the need for such a pre-defined parameter for window length.

Wespi et al. (1999, 2000) proposed a scheme for producing variable length patterns using Teiresias, a pattern discovery algorithm for biological sequences (Rigoutsos and Floratos, 1998). This method improved upon the fixed length pattern usage techniques proposed by Forrest et al. Jiang et al. (2002) extended this idea by taking into account both the intra-pattern and the inter-pattern anomalies for detecting intrusions. Though all these techniques also use the notion of variable length sequences, they rely on encountering novel events to identify anomalous behaviors. We attempt to find anomalies by using an abstract representation of an audit trail sequence based on the previously known subsequences (we have extracted them from the sequences). Michael (2003) also suggested a technique for replacing frequent subsequences with "meta-symbols" when extracting variable length sequences. But his approach looks for repeated substrings that are only adjacent to one another. Our suggested technique is a generalization of his approach.

A simple methodology for comparing amino acid sequences or nucleotide sequences, called the *Matrix Method*, was proposed by Gibbs and McIntyre (1970). A

matrix is formed between two sequences, where the elements of one sequence constitute the rows and that for the other sequence constitute the columns. An entry, where the row and the column have common elements between them, is marked with an asterisk (*), as shown in Figure 1. A series of adjacent asterisks parallel to the diagonal indicate a common subsequence between the two sequences. We have borrowed this technique for extracting common motifs of system calls. Another popular method for alignment in bio-informatics is the Basic Local Alignment Search Tool (BLAST, Altschul et al, 1990). This algorithm starts with small common subsequence as a seed and builds it gradually until mismatches appear. There is a whole body of literature in the area of bio-informatics on alignment of bio-sequences. However, they are not exactly useful for our problem of motif generation for three reasons. (1) These algorithms are for aligning two (or more) sequences and not for extracting common sub-sequences. (2) Point mutation through evolution creates gaps or replaces an element with another and the alignment algorithms need to accommodate for that. (3) One of the objectives of such alignment algorithms is fast database-search over some sequence databases and not necessarily for identifying motifs. The primary goal there is to create a distance metric between bio-sequences so that one could retrieve matched sequences against a query sequence. There are many details of the problem structure (e.g., typically a small query against the large target sequences) that are not necessarily valid in our case. However, this body of literature provides a good source for our research.

3 APPROACH

Filtering the audit data: In this preprocessing phase we extract the very large finite length sequences of system calls from the audit trails, related to a particular application (e.g., *ftpd*) and related to a particular process.

Translation of system calls: Subsequently we map each system call to a unique arbitrary symbol. Thus, we form strings of finite lengths from the sequences of system calls.

3.1 Motif Extraction Phase

We have pursued two different mechanisms for generating the motifs: *auto-match* and *cross-match*. These two techniques are discussed below.

3.1.1 Motif extraction using auto-match

For our initial experiments, we have considered any *pattern* (*substring/motif*) occurring more than once in a string as frequent enough to be a candidate motif. We start by looking at such sub-strings of length two within each string. Then, we do the same for substrings of increasing lengths (3, 4, … up to an arbitrary limit of 7). Variable length motifs are generated by studying the overlapping patterns of the fixed length motifs in the sequences (explained latter). The longer motifs may be generated out of the motifs with shorter length. While the frequencies of occurrence of the motifs within a sequence provide good indication regarding such subsumption, there

are some problems associated with the issue. To illustrate this consider the following sequence

acggcggfgjcggfgjxyz (I)

One may note that in this sequence we have a motif *cgg* with frequency 3, and another motif *cggf* with frequency 2, which is longer and sometimes subsumes the shorter motif but not always. We consider them as two different motifs since the frequency of the shorter motif was more than the longer one. Thus, a motif of shorter length may be subsumed by a longer motif only if it has the same frequency as that of the shorter motif. Frequency of a shorter motif in a sequence cannot obviously be less than that of the longer one.

Another aspect of the above issue is that two or more overlapping motifs may be merged together to form a motif of greater length. This is how we create motifs of arbitrarily long sizes. After extracting motifs of length 4 in the example sequence *(I)*, we have motifs *cggf, ggfg* and *gfgj*, all with frequency 2. Since these patterns are overlapping and have the same frequency, they may be merged together in order to obtain a longer motif *cggfgj* with the same frequency 2. However, there are instances when the smaller motifs may concatenate forming a longer motif at some places, but may occur at other positions on the sequences independently (i.e., not overlapping). Even though they have the same frequency, the concatenated longer motif does not subsume the shorter ones. Consider the sequence c*ggfgjabcggfpqrggfgxyzgfgj*. Here, the motifs *cggf, ggfg* and *gfgj* each have a frequency 2. But the longer motif *cggfgj* occurs only once, though we may wrongly conclude a frequency of 2 by using the above derivation. The remaining instances of the smaller motifs are at different (non-overlapping) positions within the string. The solution to this problem is that the occurrence of the longer motif obtained from the fusion of the smaller motifs should be verified for accuracy. If the frequency of the longer motif is found to be the same as that of a smaller one, then merging of the latter ones is all right, i.e., we ignore the smaller motifs. This technique reduces the effort of going through all possible string lengths and of finding all possible motifs for those lengths. The procedure of finding motifs of variable lengths by first merging and then verifying is repeated until no more motifs could be merged.

Let us consider the following two synthetic sequences: *acgfgjcgfgjxyzcg* (II), *cgfgjpqrxyzpqr* (III). Using *auto-match* for sequence *(II)*, we obtain the motifs *cg* with frequency 3, *gf* with frequency 2, and *cgf*, *cgfg* and *cgfgj* each with frequency 2. Motifs *cgf* and *cgfg* are subsumed by *cgfgj*, so the final list of motifs for sequence *(II)* is *cg* and *cgfgi*. For sequence *(III)*, the only motif we get by auto-match is *pqr*.

After the procedure terminates running over a string the motifs extracted are (1) of length ≥ 2, (2) a motif is a substring of another motif iff the former exists independently, and (3) the frequency of each motif is ≥ 2. Motifs extracted from each string are added to the motif database making sure that there is no redundancy in the latter, i.e., the same motif appearing in different strings is not recorded twice. In subsequent phases of modeling sequence-behaviors, the frequency values of the motifs are no longer needed.

3.1.2 Motif extraction using cross-match

The technique of auto-match is meant for finding frequent patterns in each string. Some other techniques (Jiang et al, 2002; Michael, 2003; Wespi et al, 1999, 2000)

		SEQUENCE (II)															
		a	c	g	f	g	j	c	g	f	g	j	x	y	z	c	g
SEQUENCE (III)	c		*														
	g			*													
	f				*												
	g					*											
	j						*										
	p																
	q																
	r																
	x												*				
	y													*			
	z														*		
	p																
	q																
	r																

Figure 1. Matrix representation of the cross-match strategy between the sequences II &III

also follow the concept of extracting such frequent patterns in each string. However, we are additionally interested in patterns that did not occur frequently in any particular string but are present across more than one of the different strings. We believe that these motifs could also be instrumental in modeling an intrusion detection system, as they might be able to detect attack patterns across different attack sequences. We obtained these by performing pairwise *cross-match* between different sequences. Using cross-match between the example sequences *(II)* and *(III)*, we get the motifs *cgfgj* and *xyz*, since these are the maximal common subsequences within the two given sequences, as shown in Figure 1, which represents the two sequences in a matrix and depicts the common elements with a * (Gibbs and McIntyre, 1970).

Motifs extracted in cross-match are (1) of length ≥ 2, (2) appears at least in a pair of sequences. Unique motifs extracted in this phase are also added to the database. Note that the motifs extracted in the cross-match phase could be already present (and so, ignored) in the motif database generated in the previous phase of auto-match. Also note that, although a motif could be appearing only once across a pair of sequences in the cross-match, its frequency count is 2 because of its presence in both the matched sequences.

A discerning point between our work and most of the previous ones in the IDS-literature is that we do not necessarily attach any semantics (e.g., anomalous motif) to any sub-string, although we could do so if that is necessary. We extract motifs that are just "frequently" appearing and so, may have some necessity within the sequences, and we use them only for creating an abstract representation of the sequences (explained later). Since the motif database is created only once the efficiency of the procedures for extracting them is not a critical issue.

3.2 Sequence Coverage by the Motif Database

A motif database storing all the extracted unique motifs is created from the previous experiments. Continuing with the example sequences *(II)* and *(III)*, the motif database comprises of (obtained by auto-match and cross-match) *cg*, *cgfgj*, *pqr* and *xyz*.

We also produce data reduction by our proposed representation of sequences using only the relevant motifs. Hence, it is important to verify the degree of loss of information. After extracting the motifs, we performed another study of checking the coverage of the sequences by these motifs. *Coverage* of a sequence by the motif-database is parameterized by the percentage of the sequence length that coincides with some motifs in the database. For instance, the string *acgfgjcgfgjxyzcg* has 15 characters from the motif database {*cg*, *cgfgj*, *pqr* and *xyz*} out of total length of 16, so, has coverage of 93.8%. The technique proposed by Wespi et al (1999, 2000) determines anomaly when the coverage is below a threshold. But in reality an uncovered pattern may not necessarily be a representation of an attack. Rather, it may be a novel audit sequence representing a normal behavior. This increases the rate of false alarms in their method. We do not use coverage information explicitly; rather we use it only for the purpose of checking the quality of motif extraction process. It is good to have a significant amount of coverage of the sequences by the motif database, which will indicate a low loss of information for the representation.

3.3 Motif-based representation of sequences

At first (matching phase) we extract the motifs' presence in each sequence. A standard automaton is being implemented for all the motifs to find the respective ones within a sequence in this phase. Each sequence is run through the automaton and the existing motifs along with their starting positions in the sequence are recorded.

Eventually each motif in the database is translated to a unique numeral id. [It is important to note that in the pre-processing phase the translation was for individual system calls to the corresponding ids, while the translation here is for the motifs.] Each string is subsequently represented as a scatter-plot, where a motif's starting point is the abscissa and the motif's id is the ordinate of the corresponding point. This is how the spatial relationships between different relevant subsequences/motifs within a sequence are being taken into account. The representations for the two synthetic example sequences *(II)* and *(III)* are shown in Figures 2(a) and 2(b) respectively.

3.4 Testing Phase – Anomaly Detection

Let $T = \{t_1, t_2, ..., t_N\}$ be the training set comprising of N sequences and the test sequence be denoted by s. Also, let $M = \{m_1, m_2, ..., m_n\}$ denote the set of n motifs. $f(m_i, t_k)$ represents the frequency of motif m_i in the sequence t_k., whereas $d_{m_i}(t_k)$ is the distance between two consecutive occurrences of the same motif m_i in the sequence t_k. The re-occurrence interval of a motif m_i in the sequence t_k , denoted by $r_{m_i}(t_k)$, is computed by averaging over all such motif distances within the sequence. This is represented as

$$\forall m_i \in s, t_k \;\&\; f(m_i, s), f(m_i, t_k) \geq 2, \quad r_{m_i}(t_k) = \frac{\sum_{t_k} d_{m_i}(t_k)}{f(m_i, t_k) - 1}$$

An anomaly score $AS(m_i, s, t_k)$ is associated with every motif m_i common between a pair of sequences s and t_k, which is equal to the difference of the re-occurrence intervals for that motif in the two sequences. The total anomaly score $TAS(s, t_k)$ is obtained by aggregating the anomaly scores over all such motifs.

$$TAS(s, t_k) = \sum_{m_i \in s, t_k} AS(m_i, s, t_k) = \sum_{m_i \in s, t_k} | r_{m_i}(t_k) - r_{m_i}(s) |$$

An alarm is raised if the anomaly score for a test sequence with respect to each of the training sequences exceeds the threshold. That is,

$\forall k \in [1, N], \quad Total\ Anomaly\ Score(s, t_k) > Threshold \quad \Rightarrow \quad Anomalous\ Sequence$

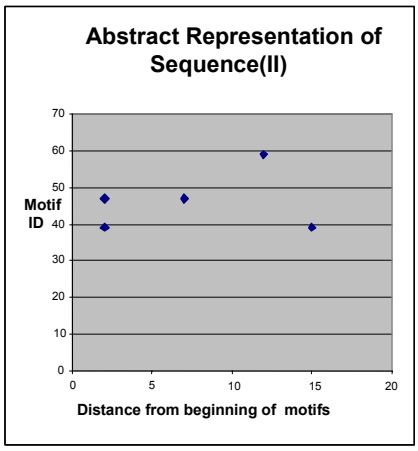

Figure 2(a)

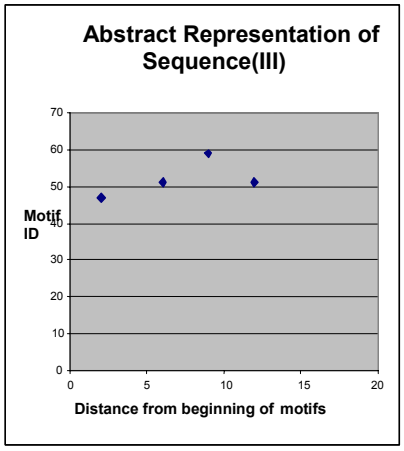

Figure 2(b)

Figures 2(a) and 2(b). Cloud representations of the sequences *acgfgjcgfgjxyzcg* and *cgfgjpqrxyzpqr* respectively

4 RESULTS AND ANALYSIS

The data set that we have used for the extraction of motifs is developed by Lippman et al. (2000) for the 1999 DARPA Intrusion Detection Evaluation. The test bed involved a simulation of an air force base that had some machines under frequent attack. These machines comprised of Linux, SunOS, Sun Solaris and Windows NT. Various IDS's are being evaluated using this test bed, which comprised of three weeks of training data obtained from the network sniffers, audit logs, nightly file system dumps and BSM logs from Solaris machine that trace the system calls. We selected the Solaris host and used the BSM audit log (Osser and Noordergraaf, 2001) to form the sequences for the processes corresponding to the *ftpd* application. For a

given process id, all the data from the *exec* system call to the *exit* system call comprised the data for that particular process. The *fork* system call was handled in a special manner. All the system calls for a child process are for the same application as the parent process until it encounters its own *exec* system call. Thus, we have used variable length sequences corresponding to different processes running over the ftp server. We have selected a total of 91 sequences for the *ftpd* application across weeks 3, 4 and 5 of the BSM audit log. In our experiments, the sequence lengths were mostly in a range of 200-400, with an average sequence length of 290.11 characters.

Table 1. Coverage's of the sequences by motifs

BSM Data Used	Range of Coverage (Auto-Match)		Range of Coverage (Cross-Match)	
	Lowest	**Highest**	**Lowest**	**Highest**
Week 3	82 %	94 %	91 %	99 %
Week 4	65 %	94 %	90 %	99 %
Week 5	65 %	92 %	96 %	99 %

The first set of experiments involved motif extraction using auto-match and cross-match methods, as explained before. We generated 66 unique motifs in the auto-match and a total of 101 unique motifs after the cross-match, for our motif database. The next set of experiments computed the percentage coverage of the audit sequences by the motifs to check the quality of the motif database for the studied sequences. Table 1 presents the lowest and the highest coverage over the sequences. It shows better coverage for cross-match since it considers motifs across sequences ignoring their frequencies. A good coverage of the sequence by the motifs implies here that there will be a low loss of information in our scatter-plot representation.

After obtaining the motifs, we created the abstract representations of the sequences as described in Section 3.3. They are the scatter-plots for the sequences in the *ftpd* application of the 1999 DARPA evaluation data set. Our next set of experiments aimed at generating alarms based upon the anomalies observed, as discussed in Section 3.4. Week 3 comprised of the training sequences and weeks 4 and 5 were used as the test data as they contained unlabeled attacks. The evaluation criterion is the number of true positives and false positives obtained by using our methodology. Utilizing a simple metric for anomaly score (average separation of each motif's multiple occurrences in a sequence) as explained in Section 3.4 we are able to detect 6 attacks with 3 false alarms. We were able to successfully detect 1 warezmaster, 2 warezclient, 2 ftp-write and 5 instances of a guessftp attack. Whereas warezmaster and warezclient attacks are denial-of-service attacks, ftp-write and guessftp belong to the remote-to-local attack category. More sophisticated metric will be deployed in future for automated clustering of the sequences.

Figure 3 depicts curves for two test sequences – a normal test sequence and an attack test sequence respectively. Each point on a curve corresponds to the anomaly score (Y-axis) of the corresponding test sequence with respect to the training sequences (each with a unique id as on X-axis). The results indicate that the execution of a normal process results in less deviation of relative motif positions and hence a lower anomaly score. A malicious sequence, on the other hand, consists of execution

of code which alters the spatial relationship between the motifs, resulting in anomaly scores (with respect to all training sequences) exceeding the threshold. We chose a threshold of 100 for our experiments. We also varied the threshold and raised it till 300 but there was no change in the results.

We would also like to mention that an attacker might devise some clever techniques to evade typical sequence-based anomaly detection systems. Wagner and Soto (2002) presented one such idea wherein they were successful in modeling a

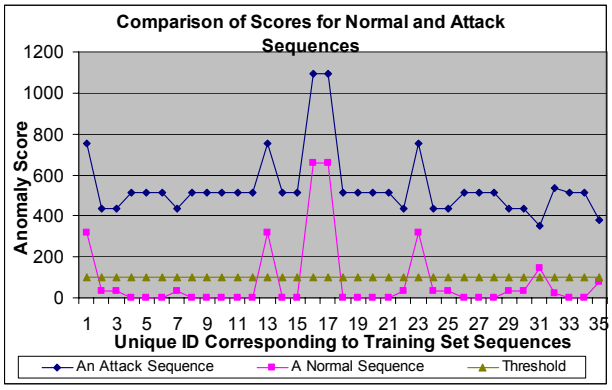

Figure 3. Anomaly score curves for a normal and an attack sequence (from the *ftpd* application in the BSM audit log of the 1999 DARPA evaluation data set). Each point on a curve represents the anomaly score of the sequence with respect to some sequence (denoted by unique sequence ids on the X-axis) from the training set

malicious sequence by adding null operators to make it consistent with the sequence of system calls. The sequence based techniques dealing with short sub-string patterns can be bypassed by spreading the attack over longer duration (or longer sub-sequences). Our technique uses variable length motifs and also takes the relative positions of the motifs for anomaly detection, and hence will be robust against such evasion. In essence, our system models sequences at two different levels – at the individual motif level and also at the level of spatial relationship between motifs within the audit sequence. The latter level adds to the security of the system and would make it even harder for the attacker to evade the system, since he has to now not only use the "normal" audit event patterns, but also to place those event-sequences/motifs within the respective sequence at proper relative positions.

5 CONCLUSIONS AND FUTURE WORK

We have described two different lines of approach for extracting motifs from finite length strings. We were successful in generating frequently occurring motifs using the first technique (auto-match) and motifs that were common (but not necessarily frequent) across different sequences using the cross-match technique. Motifs obtained from the cross-match technique displayed better coverage across all the sequences. These pattern-extracting techniques can also be used for data reduction without much

loss of information, since we now deal with only the subsequences (motifs) and not the whole audit trail. The structure of a sequence is accurately captured by the motifs and a good coverage across the sequence implies low inherent loss of information.

We have also presented an abstract motif-based representation of sequences that preserves the relative positions between the motifs in a sequence. This abstract representation enhances the efficiency and effectiveness of the resulting intrusion detection system. Our results suggest that not only are the novel system call sequences important, but also the relative positions of the motifs/patterns are important in modeling the activities over a host-system. Of course, if an attack occurs with an event or a motif not existing in our motif database we will not able to detect it. However, our model can be extended to incorporate such cases.

We intend to use an unsupervised learning technique in future, which will not only be able to distinguish the activities as normal or abnormal but also will be able to classify a richer repertoire of system behaviors. Such knowledge of a particular attacker's behavior may also help in tracking him (or them) down.

Acknowledgement: National Science Foundation has partly supported (IIS-0296042) this work.

REFERENCES

1. Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic Local Alignment Search Tool. *Jnl. Of Molecular Biology*, vol. 215, pp. 403-410.
2. Forrest S., Hofmeyr S., Somayaji A., and Longstaff T. (1996). A Sense of Self for UNIX Processes. In *Proceedings of the1996 IEEE Symposium on Research in Security and Privacy*, pages 120-128. IEEE Computer Society, IEEE Computer Society Press.
3. Gibbs A.J. and McIntyre G.A. (1970). The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences. *Eur. J. Biochem*. 16:1-11.
4. Jiang N., Hua K., and Sheu S. (2002). Considering Both Intra-pattern and Inter-pattern Anomalies in Intrusion Detection. In *Proceedings ICDM*.
5. Lane, T., and Brodley, C.E. (1997a). Detecting the abnormal: Machine Learning in Computer Security, (TR-ECE 97-1), West Lafayette, IN: Purdue University.
6. Lane, T., and Brodley, C.E. (1997b). Sequence Matching and Learning in Anomaly Detection for Computer Security. In *Proceedings of AI Approaches to Fraud Detection and Risk Management*.
7. Lippmann R., Haines J., Fried D., Korba J., and Das K. (2000). The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks* (34) 579-595.
8. Michael, C.C. (2003). Finding the vocabulary of program behavior data for anomaly detection. *Proc. DISCEX'03*.
9. Osser W., and Noordergraaf A. (February 2001). Auditing in the Solaris™ 8 Operating Environment. Sun Blueprints™ Online.
10. Rigoutsos, Isidore and Floratos, Aris. (1998). Combinatorial pattern discovery in biological sequences. *Bioinformatics*, 14(1):55–67.
11. Wagner D., Soto P. (2002). Mimicry Attacks on Host-Based Intrusion Detection Systems. *ACM Conference on Computer and Communications Security*.
12. Wespi A., Dacier M., and Debar H. (1999). An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. *Proc. EICAR*.
13. Wespi A., Dacier M., and Debar H. (2000). Intrusion detection using variable-length audit trail patterns. In *Proceedings of RAID 2000, Workshop on Recent Advances in Intrusion Detection*.