Detection and Classification of Intrusions and Faults using Sequences of System Calls*

João B. D. Cabrera[a], Lundy Lewis[b] and Raman K. Mehra[a]

Scientific Systems Company[a]

500 West Cummings Park, Suite 3000

Woburn MA 01801 USA

cabrera,rkm@ssci.com

Aprisma Management Technologies[b]

121 Technology Drive

Durham, NH 03824 USA

lewis@aprisma.com

Abstract

This paper investigates the use of sequences of system calls for classifying intrusions and faults induced by privileged processes in Unix. Classification is an essential capability for responding to an anomaly (attack or fault), since it gives the ability to associate appropriate responses to each anomaly type. Previous work using the well known dataset from the University of New Mexico (UNM) has demonstrated the usefulness of monitoring sequences of system calls for detecting anomalies induced by processes corresponding to several Unix Programs, such as sendmail, lpr, ftp, etc. Specifically, previous work has shown that the Anomaly Count of a running process, i.e., the number of sequences spawned by the process which are not found in the corresponding dictionary of normal activity for the Program, is a valuable feature for anomaly detection. To achieve Classification, in this paper we introduce the concept of Anomaly Dictionaries, which are the sets of anomalous sequences for each type of anomaly. It is verified that Anomaly Dictionaries for the UNM's sendmail Program have very little overlap, and can be effectively used for Anomaly Classification. The sequences in the Anomalous Dictionary enable a description of Self for the Anomalies, analogous to the definition of Self for Privileged Programs given by the Normal Dictionaries. The dependence of Classification Accuracy with sequence length is also discussed. As a side result, it is also shown that a hybrid scheme, combining the proposed classification strategy with the original Anomaly Counts can lead to a substantial improvement in the overall detection rates for the sendmail dataset. The methodology proposed is rather general, and can be applied to any situation where sequences of symbols provide an effective characterization of a phenomenon.

1 Introduction

Data based techniques have been applied to problems in Information Systems Security for over fifteen years now, under the often interchangeable labels of Data Mining, Machine Learning, Pattern Recognition and Artificial Intelligence. Mainstream efforts have focussed in Intrusion Detection (eg. [1], [11], [15], [16]), centered on Anomaly Detection schemes, where a model of normal operation is extracted from the data, and deviations from the model are flagged as security viola-

*This work was supported by the Defense Advanced Research Projects Agency (Arlington, VA - USA) under contract DAAH01-01-C-R027 to Scientific Systems Company.

tions. If records corresponding to known anomalies are also available, they can also be used for detector design, and are known to improve the accuracy of Intrusion Detection Systems ([12]). Detection is however only one of the phases in the defender's cycle of activities, which also include Prevention, Damage Assessment/Containment, Recovery and Fault Treatment ([9]). The last three phases loosely correspond to the Reactive component of the defense mechanism, which we call the Response Phase for short. Clearly, an effective response needs to be tuned to the nature of the anomaly. The ability to distinguish between Intrusions and Faults, as well the ability to distinguish between sub-classes of Intrusions and Faults are an unvaluable aid for the Response Phase. If nothing is known about the anomaly, only a "blanket" course of action can be taken, such as killing suspicious processes, closing down routers at the periphery of the Information System (IS), etc. In practice, detection is combined with domain knowledge about the IS operation, some form of anomaly discrimination is empirically performed, and a Response mechanism is triggered. Clearly, it would be desirable to have a system capable of performing anomaly discrimination (or classification) automatically, with adequate accuracy. As in the case of Detection, Data based techniques can be used for designing Anomaly Classifiers, as long as we have *labeled* datasets corresponding to the various types of anomalies (Intrusions or Faults). These datasets are used for training the classifier, and computing appropriate classification thresholds on a properly selected Feature Space ([5]).

In this paper we describe and evaluate a Classifier scheme based on the *stide* methodology for detecting anomalies induced by Malicious Software Applications using sequences of system calls. We call it the String Matching Classifier, since it operates by counting the number of string matches between incoming processes and dictionaries of sequences which were previously found on anomalies. We stress that these dictionaries are *automatically* computed from Training Sets, following the procedure outlined in the sequel of the paper.

The remaining of the paper is organized as follows: In section 2 we summarize the *stide* methodology introduced at the University of New Mexico, that was shown to be effective for designing anomaly detectors for security violations induced by Privileged Programs in Unix. Section 3 describes the well-known *sendmail* dataset, which has been extensively used in the past few years for the evaluation of Intrusion Detection Systems designed on the basis of Data-centered schemes (eg. [7], [8], [12]). Section 4 investigates the performance of an Anomaly Count Detector designed on the basis of *stide*. Our procedure for Performance Evaluation is carefully described, and follow an adaptation of the *leave-one-out* approach of Statistical Pattern Recognition ([5]). Section 5 describes the String Matching Classifier, which is centered on Anomaly Dictionaries, also defined and motivated in this section. Performance Evaluation of the String Matching Classifier was also performed, using the same data ensemble as in section 4. When comparing the detection results provided by the Anomaly Count Detector of section 4 with the classification results of the String Matching Classifier of section 5 it becomes clear that the combination of the two approaches can provide a better detection scheme. In section 6 we discuss the possibility of combining the two approaches into a hybrid scheme. Mechanisms for training the resulting scheme are also outlined, and the detection results are described. Section 7 closes the paper, with our Conclusions.

2 The *stide* methodology for Intrusion Detection

Starting with the seminal work of S. Forrest and coworkers at the University of New Mexico ([4]) there has been a lot of interest in the problem of devising schemes for detecting malicious software applications using sequences of system calls. Besides the obvious importance of the problem, the paper described a simple, yet effective methodology for its solution. While the work reported so far concentrated on System Calls made by privileged processes in Unix, it is apparent that the methodology can be applied to other computational systems, such as the Java virtual machine and the CORBA distributed object, as suggested in [17]. The methodology, named *stide* (sequence time-delay embedding) can be summarized as follows:

1. **Modeling**: The phenomenon under study is characterized by a sequence of categorical variables or symbols drawn from a finite, but possibly large alphabet. In the case of privileged processes in Unix, the variables are the OS (Operating System) Calls made by Privileged Programs, such as *sendmail*, *lpr*, *finger*, etc. There are 182 of these calls in the Sun OS 4.1 Operating System, which is the size of the alphabet in this case. Each run of a Privileged Program results in a collection of Processes. The samples are obtained by running the Program from the start to end, and recording the OS calls made by the various spawned Processes.

2. **Collecting normal data**: Collect a large dataset corresponding to normal operation, and record the sequence(s) of symbols generated by the system under study. In the case of Privileged Programs, the sample is obtained by running the program under

study several times, producing various processes, attempting to explore all modes of normal operation.

3. **Feature Extraction**: Select a short sequence size (k), and construct a dictionary of short sequences encountered in the normal sample. This is performed by running a sliding window of size k across the sample, moving one symbol at a time, and recording the distinct sequences of size k. This dictionary characterizes normality for the system under study. According to [4] the dictionary of normal sequences of OS calls provides a characterization of Self for the privileged Programs.

4. **Detection**: Given an incoming sample of the system under study, detection of anomalies is performed by comparing the sequences appearing in the incoming sample, with the sequences in the dictionary. Sequences encountered in the incoming sample that are not found in the dictionary are called anomalous sequences. An incoming sample is labeled as anomalous if and only if the number of anomalous sequences in the sample is larger than a given threshold.

A summary of the work at the University New Mexico (UNM) related to *stide* is presented in [8]. A survey of the work on Intrusion Detection using System Calls is presented in [18], which contrasts the performance of *stide* with Data-Mining ([12]) and Hidden Markov Models. The main conclusion is that all methods fare about the same for the datasets under study. In [13] the authors relate this observation with the regularity, or predictability of the datasets. A principled way is proposed in [13] to determine the optimal sequence length based on Information Theoretic measures. In [6] *stide* is applied for detecting anomalies in programs audited using Sun's Basic Security Module (BSM). In [14] and [19] the idea of combining sequences of different lengths is exploited. Finally, in [3] the benefits of using an *adaptive* sequence length are demonstrated.

3 The *sendmail* dataset

The UNM dataset for *sendmail* is available at `http://www.cs.unm.edu`. We have utilized UNM's shareware software to process these datasets. The traces were obtained at UNM and at CERT-CMU, using Sun SPARCstations running SunOS 4.1.1 and 4.1.4. Each trace contains the sequence of calls made by a number of processes spawned by *sendmail*. Processes vary widely in size, from about ten calls to over 1,000 calls. To produce the dictionaries of normal operation, we used seven traces containing a total of 346 processes and about 1.8 million OS calls. The sizes of the dictionaries corresponding to sequences with sizes ranging from 2 to 12 are shown in Table 1. We denote $\mathcal{D}_k$ as the dictionary of normal sequences of size k, and $L(\mathcal{D}_k)$ as the number of sequences in $\mathcal{D}_k$. To investigate the effectiveness of *stide* for detection and classification, we utilized 14 traces corresponding to anomalous runs of *sendmail* (five attacks and one error condition), and one (large) trace of normal operation, which was not used to construct the dictionaries. Table 2 presents the key statistics for each type of anomalous condition, and for the normal trace. Each anomalous run spawns a number of processes; Table 2 gives the total number of processes for all traces corresponding to a given anomaly. As shown in Table 2, there is a wide variation on the size of the processes in each trace. It is also worth noticing that the attacks *sm5x* and *sm565a* were unsuccessful, since the OS had a protective patch against them. Table 3 displays the results obtained using *stide* with dictionaries $\mathcal{D}_6$ and $\mathcal{D}_{10}$. The results below essentially reproduce the results in [8] for the *sendmail* Program. The second column gives the **total** number of OS calls found in all traces of a given data type. Each OS call is the end of a sequence of size k, except for the first $k-1$ OS calls in a process. Since the large majority of processes is much larger than 10, the number of sequences of sizes 6 and 10 in a trace are just a little below (less than 2% error in the

k	2	4	6	8	10	12
$L(\mathcal{D}_k)$	296	619	901	1137	1332	1505

Table 1. Dictionary sizes for sequences with different lengths.

Data Type	No. of Tr.	No. of Proc.	$\frac{\text{OSCalls}}{\text{Process}}$ Min-Max
decode (Attack)	2	12	8-1002
syslog (Attack)	4	21	30-919
forward-loop (Fault)	5	10	17-533
sscp (Attack)	1	1	373-373
sm5x (Attack)	1	8	9-838
sm565a (Attack)	1	3	17-178
All anomalous	14	57	8-1002
Normal dataset	1	147	7-387

Table 2. Key statistics for the *sendmail* dataset.

worst case) the number of OS calls in the trace[1]. Table 3

Data Type	No. of OS calls	Abn. (%) $k = 6$	Abn. (%) $k = 10$
decode	3067	0.42	0.82
syslog	6504	10.9	18.0
forward-loop	2569	8.8	13.9
sscp	373	18.2	24.1
sm5x	1537	14.4	24.5
sm565a	275	8.0	9.4
All anomalous	14325	8.78	14.3
Normal dataset	19526	0.91	3.62

Table 3. Using *stide* for detection of anomalies in the *sendmail* dataset: Percentage of abnormal sequences of sizes 6 and 10.

suggests that a simple threshold in the percentage of abnormal sequences is a valuable feature for distinguishing normal traces from all anomalous traces except *decode*, that could not be detected using *stide*. In section 4 we formally evaluate the performance of the *stide* based detector using a variation of the well known *leave-one-out* procedure of Statistical Pattern Recognition (eg. [5], p. 220).

Remark 1 (The space of sequences of OS calls - Normal) Figure 1 adapted from [10] presents a Venn Diagram relating various sets of interest as seen in terms of sequences of OS calls for $k = 4$. The Normal Dictionary is constructed by associating Observed Behavior from anomaly-free traces with Good Behavior. The Observed Behavior of traces containing anomalies is disregarded. □

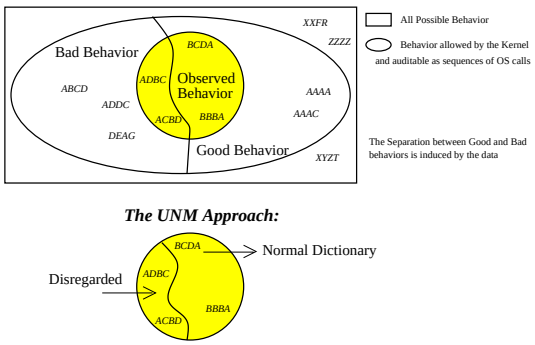

Figure 1. Behavior of Privileged Programs.

[1]The exact number of sequences of size k also depend on the number of processes in the trace, but a precise computation is not needed in this case.

Remark 2 (*stide*, RIPPER and String Matching) As noted in [12], *stide* based detectors do not use information about the anomalous sequences. When this information is utilized, it is shown in [12] that the *decode* intrusion could be distinguished from the normal traces using if-then rules extracted using RIPPER ([2]). The String Matching Procedure introduced in this paper can also detect the *decode* intrusion, as shown in section 5. □

4 The Anomaly Count Detector - Performance Evaluation

As described in previous work ([8], [12]) a trace is the detection unit for the problem, i.e. the objective is to determine if a whole trace is anomalous or not. However, in order to have a sufficiently large normal data set for training and testing, we considered individual *processes* for the normal trace as if they were individual normal runs. We discarded short normal processes (less than 50 OS calls) and kept 98 processes with sizes ranging from about 50 to about 400 OS calls. False Alarm Rates where computed in terms of these processes. In the sequel, we use the expressions trace and process interchangeably to refer to these 98 entities. Notice however that these processes/traces are used for detector design to characterize normal behavior in the same way as the anomalous traces are used to characterize anomalous behavior (Remark 3). The Anomaly Count Detector is depicted in Figure 2. In order to evaluate the performance of the Anomaly Count Detector, we performed a total of 4,000 Experiments with varying Training Sets and Testing Sets determined as follows:

- **Training Set**: The Anomalous Set consists of 1 trace from *decode*, 3 traces from *syslog* and 4 traces from *forward-loop*. The Normal Set consists of 78 processes randomly selected from the 98 normal processes available.
- **Testing Set**: The Anomalous Set consists of 1 trace from each one of the six Anomalies - the remaining *decode*, *syslog* and *forward-loop* traces which were not used in the Training Set, and the traces for *sscp*, *sm5x* and *sm565a*. The Normal Set consists of the remaining 20 processes which were not used for Training.

There are $2 \times 4 \times 5 = 40$ possible ways to select the Anomalous Training and Testing Sets as described above (*decode*: one trace for Training, one trace for Testing; *syslog*: three traces for Training, one trace for Testing; *forward-loop*: four traces for Training, one trace for Testing). For each of these 40 combinations, we considered 100 randomly selected partitions

of the Normal Processes into Training and Testing Sets of sizes 78 and 20, respectively. This gives the total of 4,000 experiments used in the evaluation. To complete the description of the detector, it is necessary to define the threshold for detection in each experiment. Let ${}^A\alpha_i,\ i = 1, 2, \cdots, 8$ denote the relative Anomaly Count of the $i-th$ anomalous trace in the Training Set, and ${}^N\alpha_i,\ i = 1, 2, \cdots, 78$ denote the relative Anomaly Count of the $i-th$ Normal Trace in the Training Set. By relative Anomaly Count it is meant the total number of anomalous sequences found in the trace, divided by the total number of sequences in the trace. The overall Anomaly Count for Anomalous Traces - ${}^A\alpha$ -, and the overall Anomaly Count for Normal Traces - ${}^N\alpha$ -, are defined as:

$$ {}^A\alpha = \frac{1}{8}\sum_{i=1}^{8} {}^A\alpha_i, \quad {}^N\alpha = \frac{1}{78}\sum_{i=1}^{78} {}^N\alpha_i $$

The detection threshold t is chosen as the center of the gap between the Normal and the Anomalous trace populations, i.e.:

$$ t \ = \ \frac{1}{2}\,({}^A\alpha + {}^N\alpha) $$

Traces in the Testing Set are labeled as Anomalous if their relative Anomaly Count α is larger than t. Otherwise, they are labeled as Normal. The "Mid Gap Rule" above attempts to strike a balance between False Alarms and Detections. Table 4 present the results obtained with the Anomaly Count Detector in the *sendmail* dataset. The results in Table 4 reflect the overall statistics for the *sendmail* dataset given in Table 3. The following specific points are worth noticing:

Data Type	Sequence Length (k)				
	2	4	6	8	10
decode	0	0	0	0	0
syslog	100	100	100	100	100
forward-loop	80	80	75.4	66.4	70.6
sscp	100	100	100	100	100
sm5x	75	100	100	100	100
sm565a	100	100	100	100	24.9
Normal dataset	1.33	1.40	5.86	10.3	26.5

Table 4. Percentage of abnormalities detected in the Testing Set by the Anomaly Count Detector. The results reflect the average across 4,000 experiments - 4,000 samples for Anomalous Traces and 4,000 $\times$ 20 = 80,000 samples for Normal Traces.

1. The rate of False Alarms for Normal Traces increase with the sequence length.

2. Detection Rates for Anomalous Traces are relatively independent of sequence length, except for two extreme cases - $k = 2$ for *sm5x* and $k = 10$ for *sm565a*. Four out of six anomalies (*syslog*, *sscp*, *sm5x*, and *sm565a*) are consistently flagged as anomalous by the Detector.

3. The results for *forward-loop* reflects the internal differences among traces in this group. It is observed that there are four traces with a large number of anomalies, and a fifth trace with a small number. Based on Anomaly Counts, this fifth trace is mistakenly labeled as Normal, leading to a detection rate of about 80%, which is confirmed in Table 4.

4. The *decode* anomaly is not detectable by the Anomaly Count Detector.

5. $k = 4$ is the best choice overall; a False Alarm Rate very close to the False Alarm Rate for $k = 2$, and the best possible detection rates.

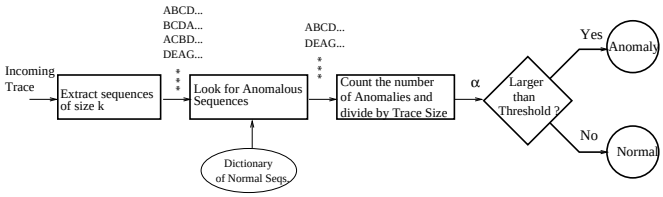

Figure 2. The Anomaly Count Detector.

Remark 3 (Usage of Normal Traces for design) Notice that Normal Traces are utilized for the design of the Detector in two ways:

1. To construct the dictionary of normal sequences;

2. To determine the relative anomaly counts for Normal Traces.

In [8] and [18] the Normal Traces are only used for dictionary construction; overall False Alarms rates are computed using additional Normal Traces, but the detection unit in these studies are sequences, not traces. □

Overall, the Anomaly Count Detector is an effective tool for Intrusion Detection; however, it does not address the key issue of classification, i.e. the discrimination between the various anomaly types. Classification is an essential capability for the Response Phase, since it enables the utilization of pre-determined actions suited for

each Anomaly Type. Clearly, the course of action following the detection of an intrusion may be different from the course of action following the detection of a Fault. The timeliness of response, as well as the entities involved will be different in each case. Evidently, classification is only possible if there are features capable of discriminating between the various types of anomalies. In the next section we show that anomaly discrimination is possible if we look to the anomalous sequences *individually*.

5 The String Matching Classifier

5.1 Anomaly Dictionaries - Motivation

Feature Extraction is understood in classical Statistical Pattern Recognition as a mechanism for data reduction. Starting with the raw data with multiple attributes, the objective is to transform it into a lower-dimensional space where properties of interest are preserved. This space is the Feature Domain. As shown in earlier work ([8]) and also in section 4 the Anomaly Count is a valuable feature for Anomaly Detection in the *sendmail* dataset. The search for features capable of providing good discrimination between classes is driven by two factors:

- **Separation between Classes**: The clusters corresponding to the different Classes need to be as separated as possible in the Feature Domain. Following the determination of an appropriate metric, a distance between classes is defined, and this distance is considered as an element in the process of Feature Extraction for Class Discrimination. This distance is called the between-class-scatter in classical literature (eg. [5]).
- **Clustering within Classes**: The samples within the clusters for each class need to be as close together as possible in the Feature Domain. A distance between samples is defined, and the aggregate measure corresponding to all classes is the other element for Feature Extraction for Class Discrimination. This is the within-class-scatter of classical literature.

Our search for good features for Class Discrimination for the *sendmail* dataset led us to the concept of Anomaly Dictionaries. The Anomaly Dictionary for a given Anomaly is the set of anomalous sequences (i.e. the sequences not found in the Dictionary of Normal Sequences) that appear in the available Anomaly Trace. When an Anomaly is first identified as such, this dictionary can be constructed by simply matching the available trace with the Normal Dictionary, and actually recording (not just counting) the anomalous sequences. Suppose that we record this Dictionary, and later on we observe a trace where a sequence belonging to the dictionary appears. Could we use this information in some way? In other words, can the sequences in the Anomaly Dictionary be used as signatures for individual Anomalies? In their original work, S. Forrest and co-workers suggested the Normal Dictionary as a characterization of Self for Privileged Programs. In the same vein, here we investigate if the Anomalous Dictionaries can serve as a characterization of Self for the Anomalies. We introduce the following terminology:

- **Unique Sequences** for an Anomaly are the union of all anomalous sequences (each counted once) encountered in all available traces for the Anomaly. The **Anomaly Dictionary** is the set of Unique Sequences.
- **Shared Sequences** for an Anomaly are the Unique Sequences which appear in at least one other Anomaly Dictionary corresponding to a different Anomaly.
- **Self Sequences** for an Anomaly are the Unique sequences which appear in *all* traces for that Anomaly.
- **Kernel Sequences** for an Anomaly are the Unique sequences which appear in *all* traces for that Anomaly, and *do not* appear in any other Anomaly Dictionary.

Given an incoming trace with Anomaly Count $\alpha > 0$, let $\alpha^{[\ell]}$, $\ell = 1, 2, \cdots, N$ denote the normalized number of anomalies of type ℓ encountered in the trace[2]. Clearly $\sum_{\ell=1}^{N} \alpha^{[\ell]} \leq \alpha$. The difference $(\alpha - \sum_{\ell=1}^{N} \alpha^{[\ell]})$ accounts for anomalies which are not found in any of the Anomaly Dictionaries. The String Matching Classifier labels incoming traces according the values of $\alpha^{[\ell]}$. It labels the trace as belonging to class ℓ^* if $\alpha^{[\ell^*]} \geq \alpha^{[\ell]}, \ell \neq \ell^*$ and $\alpha^{[\ell^*]} > 0$. It can pick multiple classes in case of ties, but these are rather uncommon phenomena. The trace is assigned to the Normal Label if and only if $\alpha^{[\ell]} = 0$ for all $\ell = 1, 2, \cdots, N$, i.e. when no match is found with any of the Anomaly Dictionaries. Variations of this policy are discussed in section 6. The operation of the String Matching Classifier is depicted in Figure 3. It is clear that the String Matching Classifier will perform well if the Anomaly Dictionaries do not overlap much, and have their sequences appearing in the majority of traces of a given anomaly.

[2] As in section 4 normalization is effected by dividing the Anomaly Counts by the Trace Size.

This happens if we have small numbers of Shared Sequences (little overlap - large between-class-scatter) and large numbers of Self Sequences (sequences appear in all traces - small within-class-scatter). If Self Sequences exist, generalization from the Training Set to the Testing Set is possible. To investigate these issues for the *sendmail* dataset, we constructed the Anomaly Dictionaries for the six types of anomalies in the *sendmail* dataset, and extracted the Shared Sequences and Self Sequences in each case. The results are presented in Table 5. The search for Shared Sequences can be performed for all types of anomalies. However, as shown in Table 2, only three types of anomalies - *decode*, *syslog* and *forward-loop* - have more than one trace available[3]. For this reason, the search for Self sequences can only be performed for these three types of anomalies.

The Shared Sequences counted in Table 5 are given as a percentage of the number of all Unique Sequences for the Anomaly, which is also given in Table 5. The Self Sequences counted in Table 5 are given as a percentage of the number of sequences in the trace having the smallest number of unique sequences for each anomaly. Hence, both percentages range from 0% to 100%. It is clear from Table 5 that the fraction of shared sequences is relatively small, and decrease for larger sequence sizes. This indicates that the Anomaly Dictionaries have little overlap for larger sequence sizes, which is a desirable characteristic for classification. The extreme example is *decode*, whose sequences do not appear in any other Anomaly Dictionary. Table 5 indicates that *decode* and *syslog* have a large proportion of Self Sequences for $k \geq 6$, while *forward-loop* does not have any Self Sequence in all cases. Again, the extreme case is *decode*, for which **all** Unique Sequences in the smallest trace are Self Sequences, for $k = 6, 8, 10$. For $k = 2, 4$ one of the *decode* traces does not have any anomalies, and therefore no Self Sequence exists.

To understand the results for *forward-loop* one needs to examine in detail the collection of traces. Four of the five traces for *forward-loop* have Unique Sequences with very little overlap with the other anomalies, and not a single overlap with the fifth *forward-loop* trace. This explains the absence of Self Sequences for *forward-loop*. This fifth trace, by its turn, has little overlap with the other anomalies. It shows that the *forward-loop* collection is in fact the union of two disjoint populations, at the least at the level of OS calls.

The results in Tables 5 suggest that the String Matching Classifier introduced earlier holds good promise for Classification.

[3]There are three available traces of *sscp* intrusions in the UNM data repository, but these traces are identical.

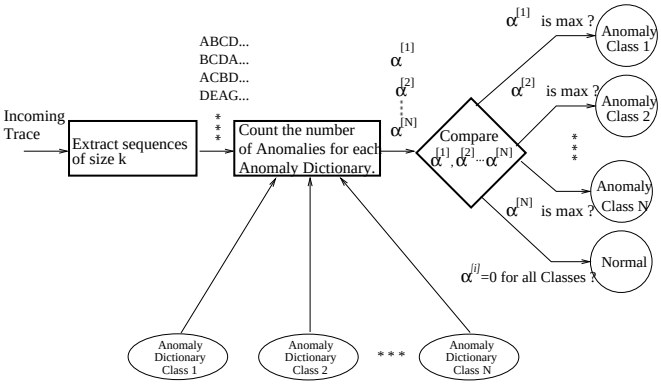

Figure 3. The String Matching Classifier. Anomaly Dictionaries are extracted using labeled datasets for each Anomaly Class.

Remark 4 (The space of sequences of OS calls - Anomalies) The results presented in this section indicates that sequences of OS calls can be used to partition the observed bad behavior as shown in Figure 4. A1=*decode*, A2=*syslog* and F=*forward-loop*. □

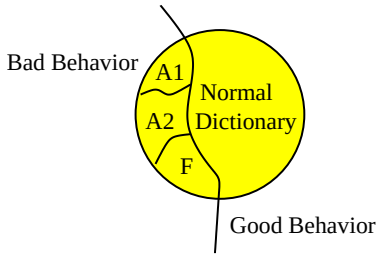

Figure 4. The observed behavior of the anomalies.

Remark 5 (Patches and Domain Knowledge) If an attack is known to the extent that sequences of OS calls could be generated in a controlled environment and used for designing a detector, it is possible the affected program could be fixed, so the attack can no longer be levied against the program. Alternately, domain knowledge about the attack could be used to generate a higher level signature, possibly more robust than the statistical signatures contained in the Anomaly Dictionary. However, we may be dealing with a new type of attack, just recently detected, for which very limited information is available, except for its higher level symptoms and a few traces. In these cases, a statistical signature at the level of audit records may be all that is available, and may be used to detect the presence of the attack, even with incomplete information about its nature and modus operandi. □

5.2 Performance Evaluation

To evaluate the Performance of the String Matching Classifier, we repeated the same experiments described in section 4, but replacing the Anomaly Count Detector by the String Matching Classifier. For each experiment, Anomaly Dictionaries are constructed for three anomalies present in the Training Set: *decode*, *syslog* and *forward-loop*. Three Anomaly Dictionaries are constructed for each experiment, and the traces in the Testing Set are matched to these Dictionaries. The remaining trace for each anomaly in each experiment allows one to verify the performance of the classifier for known anomalies. The other three anomalies not represented in the Training Set - *sscp*, *sm5x* and *sm565a* allows one to verify the behavior of the classifier when presented with unknown anomalies. The results are given in Table 5.2. A few remarks are in order:

1. The performance of the classifier for known attacks is particularly impressive: Both *syslog* and *decode* are correctly labeled in all experiments, when $k \geq 6$. As mentioned in section 5.1, one of the *decode* traces has no anomalies for $k = 2, 4$. Hence, when this trace is used for Training, the resulting Anomaly Dictionary for *decode* results empty, and the remaining trace cannot be detected. When it is used for Testing the reverse occurs.

2. The fault *forward-loop* is correctly labeled in 80% of the experiments. This result follows from the partition of the *forward-loop* into two disjoint populations, one with four traces (call it the Large Set) and the other with one trace (call it the Small Set), as described in section 5.1. In all experiments, we use four traces for Training (construction of Anomaly Dictionaries) and one trace for Testing. Hence, Unique Sequences for the Large Set will be a part of the Anomaly Dictionary for *forward-loop* in all experiments. In 80% of the experiments, the *forward-loop* trace used for Testing belongs to the large set and it will be correctly labeled. In 20% of the experiments, the Small Set is used for Testing, and no match is found with the Anomaly Dictionary.

3. Unknown anomalies are either labeled as one of the known intrusions or normal. This is an intrinsic limitation of the String Matching Classifier that can be partially resolved using a hybrid scheme, as described in section 6.

4. Finally, the performance of the String Matching Classifier for Normal Traces is also very impressive: Perfect scoring is obtained for sequences of size $k = 2, 4, 6$, and a very low score for $k = 8, 10$. It shows that there is almost no overlap between the anomalous sequences present in Normal Traces and the Anomaly Dictionaries.

6 A hybrid scheme for detection

The String Matching Classifier assigns the Normal Label to an incoming process when no match is found between the trace and all the Anomaly Dictionaries. Another possibility would be to assign the label *Unknown Anomaly* for traces with strictly positive Anomaly Count, and reserve the Normal Label only for anomalies for which $\alpha = 0$. Clearly, it would be desirable to also use the Anomaly Count for decision making, in which case the distinction between Unknown Anomaly and Normal could be determined more accurately. Consider for example the anomaly *sm565a*, which is labeled as Normal by the String Matching Classifier for $k = 8, 10$. According to the results in Table 4, *sm565a* would be labeled as Abnormal by the Anomaly Count Detector for $k \leq 8$. It suggests the use of the String Matching Classifier to improve *detection* performance[4]. In general, it is found that Anomaly Count Detectors perform better for $k = 2, 4$, while String Matching Classifiers are best for $k = 6, 8, 10$[5]. As an example, Figure 5 depicts a Hybrid Detector, that combines an Anomaly Count Detector using $k = 4$, with a String Matching Classifier using $k = 8$. Table 6 compares the detection accuracies of the three methods.

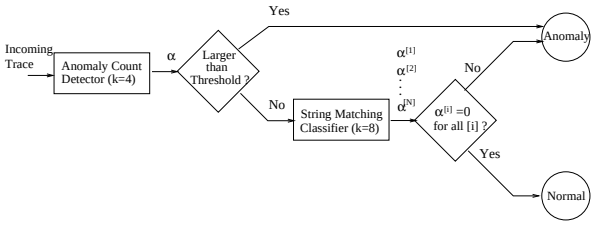

Figure 5. A Hybrid Detector.

7 Conclusions

Classification of Intrusions and Faults is a key enabling technology for the Response Phase following the detection of anomalies in the operation of Information

[4]Any classifier can also be used as a detector, by lumping all the Anomaly Classes together

[5]Recall that String Matching Classifiers using $k = 2, 4$ cannot detect the *decode* intrusion.

Systems. We showed in this paper how the *stide* methodology introduced in [4] and [8] can be utilized for the design of Classifiers for Intrusion and Faults induced by privileged processes in Unix. We introduce the concept of Anomaly Dictionaries, which were shown to provide an accurate description of Self for the various Anomalies in the *sendmail* dataset, akin to the description of Self that the Normal Dictionaries provide for Privileged Programs. It is also shown that the classification accuracy of the proposed String Matching Classifier depends on the sequence length used in *stide*. Dependence on sequence length was also observed for an Anomaly Count Detector. Both schemes were evaluated using an adaptation of the well known *leave-one-out* procedure of Statistical Pattern Recognition. It is observed that the String Matching Classifier performs better for longer sequences, while the Anomaly Count Detector performs better for shorter sequences. It is shown that a hybrid detector, combining both schemes at their best choice of sequence lengths leads to substantial improvement of detection accuracy. We have verified that different types of violations against *sendmail* give rise to specific signatures at the level of system calls. An important question at this stage is to verify if well partioned Anomaly Dictionaries can also be constructed for other Unix Programs and for other computational systems such as BSM (eg. [6]) or CORBA (eg. [17]).

References

[1] J. B. D. Cabrera, B. Ravichandran, and R. K. Mehra. Statistical Traffic Modeling for Network Intrusion Detection. In *Proceedings of the Eighth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 466–473, San Francisco, CA, August 2000. IEEE Computer Society.

[2] W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[3] E. Eskin, W. Lee, and S. Stolfo. Modeling system call for intrusion detection using dynamic window sizes. In *Proceedings of the 2001 DARPA Information Survivability Conference & Exposition*, Anaheim, CA, June 2001.

[4] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A Sense of Self for Unix Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

[5] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[6] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Using program behavior profiles for intrusion detection. In *Proceedings of the SANS Intrusion Detection Workshop*, 1999.

[7] G. G. Helmer, J. S. K. Wong, V. Hanavar, and L. Miller. Intelligent agents for intrusion detection. In *Proceedings of the IEEE Information Technology Conference*, Syracuse, NY, 1998.

[8] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[9] S. Jajodia, C. McCollum, and P. Ammann. Trusted Recovery. *Communications of the ACM*, 42(7):71–75, July 1999.

[10] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference*, pages 134–144, December 1994.

[11] W. Lee, S. Stolfo, and K. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, 16(6):533–567, December 2000.

[12] W. Lee, S. J. Stolfo, and P. K. Chan. Learning Patterns from Unix Process Execution Traces for Intrusion Detection. In *Proceedings of the AAAI Workshop on AI Methods in Fraud and Risk Management*, pages 50–56, July 1997.

[13] W. Lee and D. Xiang. Information theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.

[14] C. Marceau. Characterizing the behavior of a program using multiple length n-grams. In *Proceedings of the New Security Paradigms Workshops*, September 2000.

[15] R. K. Mukkamala, J. Gagnon, and S. Jajodia. Integrating Data Mining Techniques with Intrusion Detection. In V. Atluri and J. Hale, editors, *Research Advances in Database and Information Systems Security*, pages 33–46. Kluwer Publishers, 2000.

[16] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *Proceedings of the Usenix Workshop on Intrusion Detection*, 1999.

[17] M. Stillerman, C. Marceau, and M. Stillman. Intrusion Detection for Distributed Applications. *Communications of the ACM*, 42(7):62–69, July 1999.

[18] S. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[19] A. Wespi, H. Debar, M. Dacier, and M. Nassehi. Fixed-vs.variable-length patterns for detecting suspicious process behavior. *Journal of Computer Security*, 8:159–181, 2000.

Length	$k = 2$	
Type	Shared (%)	Self(%)
decode	0	**
syslog	10.5	100
forward-loop	27.8	0
sscp	57.1	*
sm5x	36.4	*
sm565a	0	*

Length	$k = 4$	
Type	Shared (%)	Self(%)
decode	0	**
syslog	4.2	100
forward-loop	15.5	0
sscp	37.0	*
sm5x	12.4	*
sm565a	7.1	*

Length	$k = 6$	
Type	Shared (%)	Self(%)
decode	0	100
syslog	5.2	83.1
forward-loop	10.7	0
sscp	15.2	*
sm5x	8.5	*
sm565a	9.1	*

Length	$k = 8$	
Type	Shared (%)	Self(%)
decode	0	100
syslog	5.9	90.6
forward-loop	6.6	0
sscp	10.3	*
sm5x	6.6	*
sm565a	0	*

Length	$k = 10$	
Type	Shared (%)	Self(%)
decode	0	100
syslog	6.7	87.4
forward-loop	5.9	0
sscp	10.0	*
sm5x	5.8	*
sm565a	0	*

Table 5. Percentages of Shared Sequences and Self Sequences for various sequence lengths. *: Anomalies with only one trace available. **: There are no anomalous sequences with length less than 6 for *decode*.

Length	$k = 2$			
Label	*decode*	*syslog*	*f-loop*	Normal
decode	0	0	0	100
syslog	0	100	0	0
forward-loop	0	0	80	20
sscp	0	0	100	0
sm5x	0	0	100	0
sm565a	0	0	0	100
Normal	0	0	0	100

Length	$k = 4$			
Label	*decode*	*syslog*	*f-loop*	Normal
decode	0	0	0	100
syslog	0	100	0	0
forward-loop	0	0	80	20
sscp	0	0	100	0
sm5x	0	100	0	0
sm565a	0	0	100	0
Normal	0	0	0	100

Length	$k = 6$			
Label	*decode*	*syslog*	*f-loop*	Normal
decode	100	0	0	0
syslog	0	100	0	0
forward-loop	0	0	80	20
sscp	0	0	100	0
sm5x	0	100	0	0
sm565a	0	0	100	0
Normal	0	0	0	100

Length	$k = 8, 10$			
Label	*decode*	*syslog*	*f-loop*	Normal
decode	100	0	0	0
syslog	0	100	0	0
forward-loop	0	0	80	20
sscp	0	0	100	0
sm5x	0	100	0	0
sm565a	0	0	100	0
Normal	0	0	1.1	98.9

Table 6. Performance of the String Matching Classifier in the Testing Set.

Data Type	ACD ($k = 4$)	SMC ($k = 8$)	Hybrid
decode	0	100	100
syslog	100	100	100
forward-loop	80	80	80
sscp	100	100	100
sm5x	100	100	100
sm565a	100	0	100
Normal	1.4	1.1	1.4

Table 7. Abnormalities in the Testing Set.