Reducing False Positive Rate in Anomaly Detection through Generalization of System calls

[1]Syed Shariyar Murtaza, [2]Abdelwahab Hamou-Lhadj, [3]Mario Couture

[1, 2] Software Behaviour Analysis Lab, Concordia University, Montreal, QC, Canada

[3]System of Systems Section, Software Analysis and Robustness Group, Defence Research and Development Canada, Valcartier, Québec, QC, Canada

[1]smurtaza@encs.concordia.ca, [2]abdelw@ece.concordia.ca, [3] mario.couture@drdc-rddc.gc.ca

Abstract— **Prior researchers only focus on applying different algorithms and heuristics on data to improve the accuracy of anomaly detection systems. However, high false positives are still the main issue in anomaly detection despite the use of different algorithms. This means the problem is not just with the application of algorithms but also with the discriminating strength of the data. This paper addresses the problem of reducing the false positive rate (irrespective of the algorithm) by removing the unnecessary contiguous repetition of system calls. We apply sliding window algorithm on traces that do not contain the contiguous repetition of system calls. Our results on the subject programs Sendmail, Stide, lpr and Xlock show that average false positive rate on the actual traces is higher than the 69% results on traces without contiguous repetitions; whereas, true positive rate remains the same. This shows that false positives could be reduced significantly if we remove unnecessary repetitions of system calls from traces.**

Index Terms—Anomaly detection systems, system call traces, sliding window, software security.

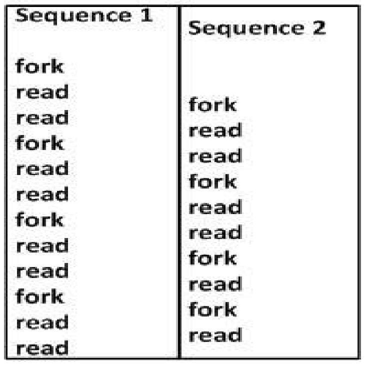
I. Introduction

Cconsiderable research has been performed in the identification of anomalous software behavior or intrusions in a system. Two main methods for detection of intrusions in software systems have been employed by scientists: misuse detection systems that detect known attacks; and anomaly detection systems that detect unknown attacks by using the knowledge of normal behavior. In this paper we focus on the anomaly detection systems.

Researchers have used variety of information to detect software anomalies, such as: (a) system call traces (e.g., application of pattern matching [1] [2] and HMM [1] [3] on system calls); (b) audit records of shell commands (e.g., multivariate statistical analysis of events in systems [3], entropy measurement on logs of shell commands [10]); and (c) audit records of network traffic(e.g., application of Bayesian network [5] and association rules [6] on network traffic).

Manuscript received October 9, 2001. This work was supported in part by Defence Research and Development Canada (DRDC) and NSERC.

Despite using variety of algorithms and datasets, a major problem with the anomaly detection systems is the generation of number of incorrect alarms on normal software behavior—i.e., false positives. Hofmeyr et al. [2] mentioned that generation of false positives is a more important issue than the identification of true positives in anomaly detection scheme. The reason is that usually there are more than one layer of detection mechanisms and if one layer doesn't detect an anomaly than there is a good chance that other layers can detect that anomaly. In fact a large number of false positives in anomaly detection systems have made the misuse detection systems first choice in the industry.



Sequence 1	Sequence 2
fork	
read	fork
read	read
fork	read
read	fork
read	read
fork	read
read	fork
read	read
fork	fork
read	read
read	

Figure 1: Two sequences in traces of Sendmail program with contiguous repetition of system calls

This raises a question that: is the problem in the application of algorithms on different datasets or is it in the properties of underlying data? During our preliminary investigation of system call traces pertaining to the Sendmail[1] program—an open source mail server—we observed that some patterns of system calls represent the same task but the number of repetition of individual calls in the pattern are different. For example, consider two sequences of system calls found in Figure 1 that were found in different normal traces of Sendmail program. The task performed is the creation of a process (fork) and reading data from the I/O device. However, in the case of "Sequence 1" fork is always followed by two reads; whereas, in the case of "Sequence 2" fork is sometimes

[1] We downloaded traces of Sendmail program form the university of New Mexico [25] and the traces were collected on SunOS.

followed by one read. If an anomaly detection algorithm is trained on the traces containing only "Sequence 1" then a mismatch will occur with "Sequence 2" when it is found in the test set—though the task performed is exactly the same. This will result in a false positive alarm on a normal trace.

Such types of false positives can be removed by only ignoring the contiguous repetitions of system calls; that is, only considering "fork, read" as one pattern of a task and ignoring the multiple contiguous repetitions of a system call. It is possible that by applying this form of generalization (i.e. removing contiguous repetitions) we might reduce the number of correctly detected anomalies (true positives). Therefore, these observations warrant an empirical investigation but if these observations hold true then we can decrease false positives rate.

Thus, our hypothesis is that by removing contiguous repetitions of system calls we can reduce false positive rate of an anomaly detection algorithm without affecting the true positive rate.

This is the fundamental problem addressed by this paper and it distinguishes it from prior work which only focused on employing different algorithms and heuristics to improve the accuracy of anomaly detection. We tested our hypothesis by using a sliding window algorithm, first used by Forrest et al. [9] for anomaly detection. Another most popular anomaly detection algorithm is hidden Markov model (HMM) which was found to be better than other algorithms in many cases. However, we chose the sliding window algorithm because Warrender et al. [1] found out that there is no clear winner on their datasets and in some cases the sliding window algorithm performs better and in some cases HMM performs well. However, the time spent by HMM in training is few days compared to few minutes of the sliding window algorithm on the same large datasets.

Our results by using the sliding window algorithm on actual traces show that average false positive rate on actual traces is higher than 69% of the results obtained by training sliding window on contiguous-repetition-removed traces. Also the results show that there is no effect on the true positive rate. This proves that our hypothesis is true. The significance of removing contiguous repetition of system calls is not only in improving the accuracy of anomaly detection but also in other areas, such as:

a) It can facilitate in reducing the storage space and decreasing the training time by reducing the size of traces.
b) It can facilitate in identifying common pattern of system calls doing the same unit task. Such unit tasks can be assigned a name and similar unit tasks can be identified across different operating systems. This can increase security of systems by facilitating co-operation among different operating environments in intrusion detection. For example trace correlation can be used to detect anomalies among diverse operating environments [8].

The rest of the paper is articulated as: Section II describes related work; Section III describes in detail the technique that we employ; Section IV explains the experimental setup; Section V shows results; Section VI mentions the threats to validity; and Section VII concludes our work.

II. Related Work

Traditional intrusion detection systems detect known software attacks by comparing them with the signatures of known attacks. They are called misuse detection systems. Another type of intrusion detection systems is anomaly detection systems that detect unknown abnormal software behavior by building a model that characterizes normal behaviour [3] .This paper focuses on anomaly detection systems.

A type of anomaly detection techniques focus on using different algorithms on normal audit records of a host (e.g., CPU usage, process id, user id, etc.) or network traffic information (e.g., TCP/IP connection information, ports scanned, etc.). These techniques measure an abnormality threshold and raise alerts when particular attribute values of a new record are above a threshold value. The use of multivariate statistical analysis on audit record to identify anomalies [4], the use of Bayesian network on network traffic records to detect intrusion [5], using frequency distribution based anomaly detection on shell command logs [10] and a framework MADMAID [6] that extracts rules by mining tcp-dump data are few examples of this type of technique.

Another type of anomaly detection techniques focus on training different algorithms on normal system calls. These techniques raise alerts when the deviation in system calls is observed in traces of new system usage. For example, using hidden Markov models [1] and pattern matching technqiues [9] on system calls to detect anomalies. Anomaly detection systems focusing on system calls deviations are related to our work and are described below.

Forrest et al. [7] propose to build a database of normal sequences by sliding a window of length 'n' on normal traces. Similar sequences were also extracted from a test trace and compared with the database. If a sequence is not matched the trace is considered as anomalous. Hofmeyr et al. [2] improve sliding window algorithm technique by computing hamming distance between two sequences to determine how much one sequence differs from another. Warrender et al. [1] compare sliding window algorithm, hidden Markov model and association rules (called RIPPER). For sliding window algorithm they used a locality frame count (i.e., number of mismatches during last 'n' calls) instead of hamming distance. They found out that HMM and sliding window performs well but on different datasets.

Warrender et al. [1] train HMM on system-call traces and raise alerts when the probability of a system-call in a sequence is below a certain threshold. Wang et al. [7] also train hidden Markov model on normal system-call traces and raise alerts when a probability of a whole sequence of system-calls is below a threshold rather than individual calls in a sequence

[1]. Yeung and Ding [10] employ HMM on system-calls (called dynamic modelling) and measure frequency distributions for shell command logs (called static modelling). They show that dynamic modelling performs better than static modelling. Cho and Park [11] use HMM to model the execution of only normal root privilege acquisitions rather than all the system events. This allows them to detect 90% of the illegal privilege flow attack. Hoang et al. [11] propose a multiple layer detection approach in which one layer train sliding window algorithm on system-calls and another layer train HMM on system-calls. They combine the output of both to detect anomalies. Hoang et al. [12] improved their earlier work [11] by combining HMM and sliding window algorithm using fuzzy inference engine. Hu et al. [13] propose an incremental HMM training method to reduce its training time and they claim that it reduces the time by 50%.

Jiang et al. [15] extract varied length n-grams from call traces of normal behaviour, and build an automaton from the n-grams that represent the generalized state of the normal traces: they use this automaton to detect abnormal traces. Ghosh et al. [8] employ standard multilayer perceptron and Elman [16] recurrent neural network on system calls to detect anomalous system calls in test data. Ghosh et al. [8] discover better accuracy with recurrent neural networks but at the expense of more time than the standard multilayer perceptron. Yuxin et al. [17] use support vector machines, decision trees and the k-nearest neighbor algorithm to classify malicious and normal software code. They [17] first extract static system-call sequences for a program (i.e., extract system calls without running a program) and then train algorithms for classification. They [17] show that their static system-call based technique produce better results than dynamic system-call based techniques.

Tandon [18] propose different variations of LEARD, a conditional rule learning algorithm [19], to learn rules with sequences of system-calls and their arguments. They also propose that for a limited training data, coverage can be increased by generating new rules for the attributes that are removed due to pruning in LEARD. This is because increasing coverage can help in reducing false positives. In experiments of this paper, we used system-calls without argument because dataset did not contain arguments. However, we envisage that the results should be similar for calls with arguments because our technique is independent of arguments.

Research Gap

Prior research in the field of anomaly detection only focuses on using different algorithms or employing different datasets to detect anomalies. However, to the best of our knowledge, no one has addressed the issue of reducing false positives by improving the discriminating strength of the data. If input data feed to algorithms have enough discriminating characteristics, then algorithms will automatically yield better true positive accuracy with low false positives. Thus removal of contiguous repetitions from system call traces to improve accuracy of anomaly detection is a research problem that has not been addressed yet and our work is novel.

III. Anomaly Detection by Using Sliding Window Algorithm Over System call sequences Without Contiguous Repetition

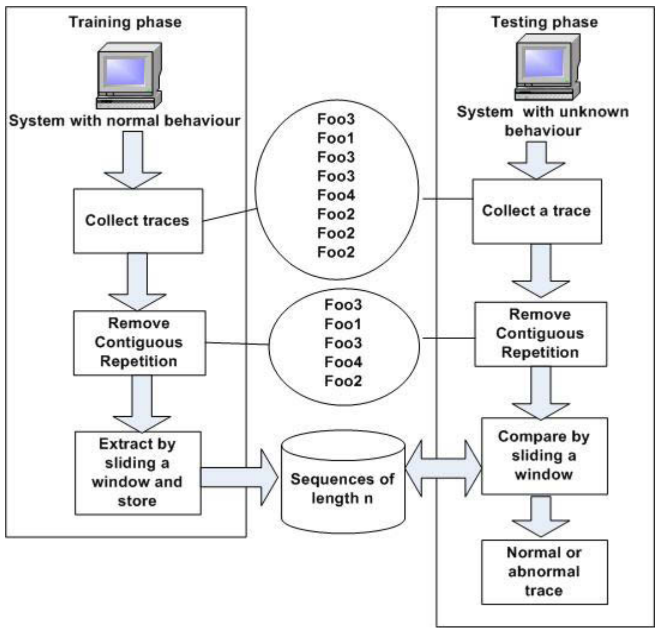

Training phase
System with normal behaviour
Collect traces
Remove Contiguous Repetition
Extract by sliding a window and store
Foo3
Foo1
Foo3
Foo3
Foo4
Foo2
Foo2
Foo2
Foo3
Foo1
Foo3
Foo4
Foo2
Sequences of length n
Testing phase
System with unknown behaviour
Collect a trace
Remove Contiguous Repetition
Compare by sliding a window
Normal or abnormal trace

Figure 2: The proposed scheme

Our proposed anomaly detection scheme is shown in Figure 2. It works in two stages: (a) training phase; and (b) testing phase. In the training phase, the anomaly detection model is constructed form the normal traces of system calls of a program. In the testing phase, the constructed model is used to detect intrusions in the traces of system calls collected for the same program with unknown behaviour. The details of two phases are as follows:

Training Phase: The steps of training phase are:

(a) Collect Traces: First, traces are collected during the normal usage of a program on a system. An example of a trace of system-calls is shown in the call-out in Figure 2, where Foo3, Foo1 and Foo2 are system calls.

(b) Remove Contiguous Repetition: Second, the contiguous repetitions of system calls is removed from the traces; that is, if a system call occurs repeatedly in a sequence then the repetitions are removed and only one occurrence is kept. This is shown in the second call-out in Figure 2. It can be observed in the first call-out in Figure 2 that after Foo1, Foo3 occurred twice, and after Foo4 Foo2 occurred three times in a sequence in the first call-out. After removing contiguous repetitions, Foo3 and Foo2 appeared only once at their particular positions in the sequence as shown in the second call-out in Figure 2.

(c) Sliding Window: Third, the sequences of length 'n' are extracted by sliding a window over the sequences of system calls of a trace. The sliding window algorithm is explained in the next section (Section III.A). The sequences of length 'n' are then stored in a database/file.

Testing Phase: In the testing phase, the traces are collected for a program when its behaviour is unknown and it is required to detect whether an intrusion has taken place or not. The first two steps of the testing phase are same as the training phase. In the third step, the sliding window algorithm is used to extract the sequences from a test trace and then those sequences are compared with the sequences of normal traces. If a sequence is not found in the database then the trace is considered as anomalous.

The rest of the section continues as follows: first we explain the sliding window algorithm in Section III.A and second we explain the evaluation criteria for our experiments in Section III.B.

A. Sliding Window Algorithm

The sliding window algorithm for anomaly detection was first used by Forrest et al. [8] for detecting anomalies using system calls. In this algorithm, sequences of system calls are extracted by sliding a window of size k across the traces of system calls. To understand the sliding window algorithm clearly, we we formally define this algorithm as follows:

- Consider a set R of system calls. For example R= {lseek, sigvec, vtimes, sigblock, setpgrp, vtrace, sigvec} are system calls on SunOS.

- An event is expressed as a pair (A, t), where:
  $A \in R$ is a system call, and t is an occurrence time of the event expressed as an integer—indicating the order of the system call in a temporal sequence.

- An event sequence S on R can be formally expressed as a triple (s, Ts, Te), where:
  Ts <= Te and Ts is the (integer) starting time, and Te is the (integer) ending time, and s= $< (A_1, t_1), (A_2, t_2) \ldots\ldots (A_n, t_n) >$ is an ordered sequence of events such that $Ai \in R$ for all i=1….. n, and $t_i <= t_{i+1}$ for all i=1….. n-1, and $T_s <= t_i <= T_e$ for all i = 1 to n.

For example, a sequence of events is shown in Figure 3. In Figure 3, the (lseek, 1) is an example of an event (A, t), and lseek is the system call. This sequence S of events occurred when the Sendmail (a mail server) program was running on SunOS. In Figure 3, Ts = 1 and Te = 12; and s = < (lseek,1), (lseek,2)…… (sigvec,9) >.

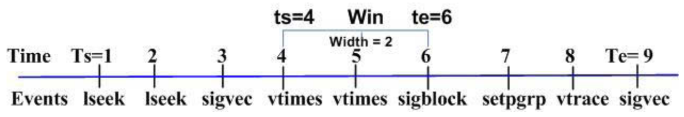


Figure 3: System call sequence for the Sendmail program.

- A window on an event sequence S (s, Ts, Te) is also an event sequence W (w, ts, te), where: ts <= Te, te >= Ts, ts <= te, and w consists of the events (Ai, ti) with ts <= ti <= te and Window width, win(w), is the time span te –ts.

An example of the window width of win (w) = 2 is shown in **Error! Reference source not found.**, and it is W (<(vtimes, 4), (vtimes,5), (sigblock,6) >, 4, 6).

In the sliding window algorithm, we slide a window one time unit on the sequence S and record all the system calls with respect to their order in a database. For example, after sliding the window win(w)=4 across the sequence S of Figure 3, the following database of sequences of length 4 is generated (shown in
Figure 4).

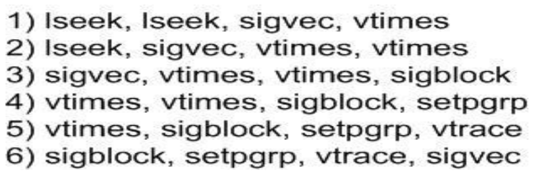

1) lseek, lseek, sigvec, vtimes
2) lseek, sigvec, vtimes, vtimes
3) sigvec, vtimes, vtimes, sigblock
4) vtimes, vtimes, sigblock, setpgrp
5) vtimes, sigblock, setpgrp, vtrace
6) sigblock, setpgrp, vtrace, sigvec

Figure 4: Sequences obtained after sliding a window of width 4 on a sequence of Figure 3.

In the above example, we slide a window of width 4 on an actual trace of system-calls which contain several contiguous repetitions of the same system-calls. Recall from lead text of Section III and Figure 2 that we removed the contiguous repetitions from a trace before applying the sliding window algorithm. Thus if we apply sliding window algorithm on the sequence S of Figure 3 after removing contiguous repetition then
Figure 4 will result in: (a) elimination of sequence (1) and; (b) merging of sequence (3) and (4) to a new sequence "sigvec, vtimes, sigblock, setpgrp".

In the sliding window algorithm, we actually consider each trace of system calls as one sequence S. During the training phase we slide a window of width win(w) = n over each normal trace (with contiguous repetitions being removed) one by one to build a database of sequences of length n. The window width win(w) is chosen by a user. In the later sections we shall show our results with different window widths.

During the testing phase, we slide a window of the same width as the sequences in training database across a trace, for which the contiguous repetitions have been removed. If new sequences (of width 'n') are found which do not match with sequences in a database then we consider the trace as anomalous, otherwise we consider it as normal. For example, the sequences (a) "open, write, close, chown" and (b) "sigvec, lseek, sigblock, setpgrp" are anomalous sequences for the database of Figure 4 because (a) has new system-calls and (b) has a new sequence of system calls that is not present in the database of
Figure 4.

B. Evaluation Criteria

The sequence of length 'n' extracted from a test trace is either anomalous or normal. We have only two decisions to make and hence we have two types of classification errors: false positives and false negatives. We consider that a false positive occurs when a single sequence (of length 'n') in a normal trace is classified as anomalous. A false negative occurs when all the sequences (of length 'n') in an anomalous trace are classified as normal. In statistical decision theory [21], false negatives are Type I errors and false positives are considered as Type II error.

In line with convention of previous techniques [1] [5], we evaluate the accuracy of our proposed technique using the two measures: true positives (TP) and false positives (FP). False negatives can be obtained by subtracting true positive rate from one (FN=1-TP). We measured TP rate and FP rate by following two equations:

$$TP\ rate = \frac{Number\ of\ anomalous\ traces\ detected\ as\ anomalous}{Total\ number\ of\ anomalous\ traces}$$

Equation 1: True positive rate

$$FP\ rate = \frac{Number\ of\ normal\ traces\ detected\ as\ anomalous}{Total\ number\ of\ normal\ traces}$$

Equation 2: False positive rate

IV. Experimental Setup

In this section we explain the dataset (see Section IV.A) we used in our study and the process (see Section IV.B) that we used to conduct our experiments.

A. Dataset

In this paper, we used the system call datasets publicly available on the site of University of New Mexico [22]. This dataset consists of traces of system calls of normal and intrusive behaviour of several programs. The intrusion data were generated by intruding these programs according to the public advisories posted on the Internet. The dataset has also been used by several researchers in past (e.g., [1] [2] [3] [5] [11] [13]).

Table 1summarizes the information about traces of different systems (in total 5) that we used in our study from this dataset. Each trace is a list of system calls generated from the execution of one process. A process can correspond to a single task, or a part of a task fulfilled by multiple processes of the same program. In conformance with other researchers [1] [5], we considered all the system calls that belong to one process as one trace pertaining to that program. Table 1 shows the program name, number of intrusion traces, total number of normal traces, number of normal traces used for training and the number of normal traces used for testing.

Sendmail is a production mail server and we used the synthetic data for sendmail that were collected at UNM and the dataset was termed UNM synthetic sendmail data by researchers at UNM [22]. The intrusions generated on the Sendmail server were “Sunsendmailcp” intrusion and decode alias attack. The “Sunsendmailcp” intrusion caused Sendmail to append a message to an email and decode intrusion allowed remote user to modify files. A third type of anomaly which sends sendmail into a forwarding loop was also generated and its traces were also used in our evaluation [22].

Stide is an anomaly detection program which was processing sendmail data. A denial of service attack was tested against stide: the attack actually tied up all the memory of the system and affected any program that request memory [1] [22].

Data for lpr were collected at two different universities: MIT and UNM. UNM dataset include 15 month of activity and MIT dataset include 3months of inactivity. The intrusion was lprcp symbolic link intrusion for both the datasets and it consists of 1001 jobs [1] [22].

Data for Xlock included 71 synthetic normal traces (approximately average 5000 calls per trace) and one live normal trace. The live trace was very long and collected continuously for tow days and it had about 16,000,000 system-calls in a single trace. We divided this trace into 1600 traces of 10,000 system calls each and used 50 of these traces for training along with 71 synthetic normal traces. The intrusion on Xlock was a buffer overflow and resulted in two traces of approximately 500 system calls each [1] [22].

B. Process

We implemented our algorithm in Java which first removed contiguous repetitions from the traces. Subsequently, it extracted sequences by sliding a window from the traces for a given window width and stored them in a database. We experimented with different window widths for training and used the same window width for testing. For example, if we trained with window width of 6 (win(w)=6) we performed testing with the same window width. We chose window widths of length 6, 10, 15 and 20 for our experiments. We started with window width of 6 because other researchers [8] found length 6 best in their experiments.

The sliding window based technique was first used by by Forrest et al. [8] on actual traces and was later onmodified by other researchers [1] [2] for anomaly detection. They [1] [2] [8] employed different heuristics on the sliding window algorithm such as comparing sequences [8], measuring the hamming distance between sequences [2] and identifying the number of mismatches within last few system calls (called locality frame count) [1]. Subsequently, they [1] [2] determine a threshold value and use that threshold value to detect anomalies.

We downloaded Stide program, which implements sliding window algorithm with different heuristics [1] [2] [8], from the site of UNM [22]. We set the locality frame count to 20, which was the default value and also used by other researchers [22] for anomaly detection. We trained Stide on original traces. If at least one anomalous sequence was found we considered the trace as anomalous. We also experimented by enabling the hamming distance but the results were the same as the locality frame count because we considered a trace anomalous if a single sequence was anomalous. In the next section, we only report results for the locality frame count.

V. Results

In this section we explain the results of our study. In this section we contrast our results with Stide —i.e., application of the sliding window algorithm on a normal trace without removing contiguous repetitions. Later on in the section, we validate our hypothesis, introduced in Section I, by applying statistical analysis is conducted to verify the significance of results.

Table 2: Summary of the traces of the subject programs

Program	Intrusion traces	Normal traces	Normal traces used for training	Normal traces used for testing
Sendmail	25	346	135	211
Stide	105	13726	600	13126
MIT live lpr	1001	2703	415	2288
UNM live lpr	1001	4298	390	3908
Xlock	2	1731(71)	121(71+50)	1610

Table 2 summarizes the results for each of the subject programs shown in Table 1. For each program, the false and true positives are shown. There are four main rows, one for each window width, and each row is further divided into sub-rows. Each sub-row shows the results for "Stide" and our contiguous repetitions removal based anomaly detection method (to save space we termed it CRA in Table 2). For example, first row in Table 2 for Win(w)=6, the anomaly detection method "Stide", and the subject program "Sendmail" can be read as 24 false positives (FP) were found in the normal test dataset and 16 true positives were found in the anomalous dataset by using the anomaly detector "Stide". Similarly, for the program "Sendmail" at window width 6, only 23 false positives were found and true positives remained 16 in number when anomaly detection method was CRA The total number of anomalous traces, normal traces in the test set can be seen in Table 2.

Table 1: False Positives (FP) and True Positives (TP) on the subject programs obtained using Stide (sliding window algorithm on a trace) and our method CRA (sliding window on contiguous-repetition-removed traces)

		Sendmail		Stide		MIT live lpr		UNM live lpr		Xlock	
		FP	TP	FP	TP	FP	TP	FP	TP	FP	TP
Win(w) = 6	Stide	24	16	69	104	196	1001	571	1001	24	2
	CRA	23	16	66	104	181	1001	327	1001	18	2
Win(w)= 10	Stide	27	16	12746	104	350	1001	803	1001	24	2
	CRA	25	16	137	104	183	1001	356	1001	18	2
Win(w)=15	Stide	30	16	12760	104	458	1001	869	1001	24	2
	CRA	27	16	187	105	183	1001	423	1001	18	2
Win(w)=20	Stide	33	18	12770	104	537	1001	958	1001	24	2
	CRA	33	18	188	105	212	1001	473	1001	18	2

It can be observed from Table 2 that in the case of CRA (our approach) false positives are lower than Stide. We can also see that as the window width increases the difference between false positives of CRA and anomaly detection method Stide increases. If we only focus on the results obtained using window width 6 then we can see that for programs "MIT live lpr" and "UNM live lpr" the difference between false positives increases by almost 50%. Similarly in the case of the "subject program Stide" the difference between anomaly detector Stide and CRA false positives is almost 80% in false positives when the window width is increased beyond 6. In this case of "subject program Stide" CRA was able to detect all 105 anomalous traces but the anomaly detector Stide could only detect 104 anomalies on large window widths.

In Table 2, true positives for CRA only improve slightly in some programs with higher window width over the anomaly detector Stide. However the difference in false positives between CRA and anomaly detector Stide is noticeable on larger window widths. This implies that when using CRA we can extract longer sequences from traces and detect more true positives and fewer false positives than Stide. This is shown in Figure 5, where we show the average false positive rate and average true positive rate for all the subject programs.

In Figure 5, we first measured the true positive rate and false positive rate using

Equation 1and

Equation 2 and then took the average. Each point in Figure 5 represents the average FP rate and average TP rate for a particular window width. There are four points in total for window widths of 6, 10, 15 and 20. For example for the series "CRA", line starts with average FP rate and average TP rate for window width 6 for the subject programs and ends at the values for window width 20.

Figure 5 actually shows that with higher window widths false positive rates for the CRA remains almost the same (approximately 8% on the average), whereas in the case of anomaly detector Stide it increases to approximately 30%. In the case of true positive rate, higher window width yield only a slightly better true positive rate than Stide for CRA.

In order to validate that there is a significant difference between false positive rates of CRA and anomaly detector Stide, we also conducted a Wilcoxon signed rank test between false positives obtained using CRA and Stide. We chose

Wilcoxon test because it was not know whether the data follows normal distribution or not. Our null hypothesis is that there is no significant difference in the rates of FP rate and TP rate. We chose alpha level 0.05 to test the hypothesis. We used false positive values of all the five programs for four window widths as shown in Table 1. This resulted into 20 values and a Wilcoxon test with 20 observations resulted into Z = 3.823 and p = 0.000. This implies that null hypothesis is rejected and there is a significant difference between false positive rate with the increase in window widths as $p<0.05$.

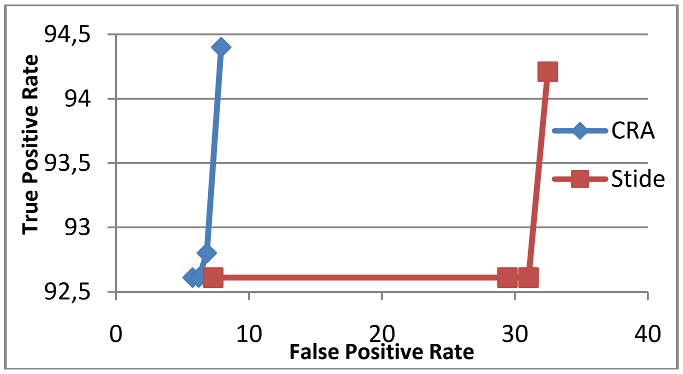


Figure 5: Average false positive rate vs. average true positive rate of the subject programs.

Similarly, we applied a Wilcoxon signed rank test between true positive rates of CRA and Stide at alpha 0.05. A Wilcoxon signed test between 20 true positive values resulted in Z=1.342 and p=0.180 > 0.05, implying that no significant difference exist between true positive rates with the increase in window widths. This means that using lower window widths (e.g., win(w) =6) will be sufficient for analysis as TP rate doesn't increase signficantly.

If we look closely at the results on win(w)=6 in Figure 5 and Table 2 the false positives obtained using CRA are still lower than the false positives obtained using Stide, and shows the effect of removal of contiguous repetition. To statistically measure the strength of CRA (i.e., removal of contiguous repetition) we measure the standardized effect size [23]. The advantage of the use of standardized effect size is that they are comparable across different studies even with different sample sizes [23] . In our case, we calculated the effect size using Cohen's d measure [21] and the effect size between Stide and CRA at win(w)=6 was 0.5482. An effect size of 0.5482 between false positive rates of Stide and CRA can be interpreted as the average false positive rate of Stide will be 0.5482 standard deviations above than the average false positive rate of CRA. In terms of percentile standing [21], the results are interpreted as the average false positive rate obtained using the Stide would be higher than 69% results of CRA—that is, results obtained after removing contiguous repetitions.

Thus the removal of contiguous repetition reduces the false positives rate significantly when a sliding window algorithm is used for anomaly detection. This proves our hypothesis, stated in the Section I, removing contiguous repetitions reduces the false positive rate without affecting the true positive rate when detecting software anomalies.

VI. Threats to Validity

In this section we describe threats to validity of our experiments according to four categories: conclusion validity, internal validity, construct validity, and external validity [24].

A threat to validity to our conclusions exists because we use traces of only one intrusion in all the programs except Sendmail. Only one type of intrusion traces could have discriminating sequences of system calls from normal traces that made it easier to detect and made it biased towards high true positive rate. More than one intrusion as in the case of Sendmail program would have given us a better idea of true positive rate.

A threat to internal validity exists in the implementation for removing contiguous repetitions and applying sliding window algorithm. An internal threat also exist in separating the system-calls of multiple processes provided by UNM [22] in a single file. If more than one process id exists in a trace we created a new trace for every process in a trace and put all the calls pertaining to a process in a separate file. Thus a threat exists as all of these processes were automated. We have minimized this threat by manually investigating the outputs.

A threat to construct validity exists in measuring the false positive rate and true positive rate by only using the simple sequence matching. An improved true positive rate could be obtained by measuring the frequency of sequences and using a threshold to detect anomalies, and a reduced false positive rate could be obtained by identifying similarity proximity among sequences when sequences mismatch. We mitigated these situations by:

(a) measuring the frequency of whole sequence in a trace without contiguous repetitions and identifying abnormal frequency values to improve TP rate. For example ABC occurs twice in a trace on average and occurrences of ABC beyond 3 standard deviation of mean in a new trace is anomaly

(b) measuring the frequency of occurrences of system calls with in a sequence of a trace without contiguous repetitions and identifying abnormal frequency values to improve TP rate. For example if a sequence ABBCCC occurs in a trace then its representation without contiguous repetitions will be ABC with frequencies A=1, B=2 and C=3. A similar procedure to point (a) was adopted to detect anomalies.

(c) applying sequence alignment algorithm such that only if a certain percentage of a sequence matches then it is considered as a normal trace. This is to increase FP rate.

However, these heuristics either resulted in higher false positives or no true positives at all. The contiguous repetition removal method presented in this paper is an optimal solution that we obtained through variety of experiments.

A threat to external validity exists in generalizing the results of this study as we have only experimented on five of the programs and this publicly available dataset is about 10 years old. This threat is mitigated by the fact that attacks were real attacks and programs are actual commercial programs.

VII. Conclusion

Prior research in the field of anomaly detection systems only focus on applying variety of algorithms (e.g., HMM [1] [3], pattern matching [1] [2], Bayesian network [5]) on audit [5] or system call [1] [2] data. Despite using different of algorithms a large number of false positives are still an important issue in the field of anomaly detection. This implies that the problem is not only in the application of algorithms but also in the properties of dataset. This paper focuses on a problem of reducing false positives in anomaly detection by removing unnecessary repetitions of system calls from the traces of system calls (see Figure 1). Based on our preliminary observations (see Section **Error! Reference source not found.**) we state a hypothesis "*by removing contiguous repetitions of system-calls we can reduce false positive rate of an anomaly detection algorithm without affecting the true positive rate.*"

We used a sliding window algorithm (see Section III.A) to test our hypothesis. We tested our hypothesis on five datasets of open source programs (see Section IV.A) with the intrusions generated according to public advisories posted on the internet. Our results show that when a sliding window algorithm is trained on actual traces then the average false positive rate is higher than 69% of the results obtained by training sliding window on contiguous-repetition removed traces. The true positive rate however remains the same. This proves that our hypothesis is true.

We only experimented with system-calls without arguments on traces collected about 10 years ago [22]. Researchers [19] have shown that accuracy of anomaly detection can be improved by using system-calls with arguments. In future we plan to evaluate our method on system-calls with arguments and on the traces of intrusions generated from the latest data as we are planning to simulate attacks in our own lab. We also plan to effect of removal of contiguous repetitions across traces of different operating environments, which can facilitate in trace correlation and anomaly detection in distributed environments such as cloud computing.

References

[1] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proc. of 1999 IEEE Symposium on Security and Privacy*, Oakland, USA, May 1999, pp. 133-145.

[2] S. A. Hofmeyr, S. Forrest, and and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Security*, vol. 6, no. 3, pp. 151-180, Aug. 1998.

[3] A. Patcha and J.,M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448-3470, Aug. 2007.

[4] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert, "Multivariate Statistical Analysis of Audit Trails for Host-Based Intrusion Detection," *IEEE Trans. on Computers*, vol. 51, no. 7, pp. 810-820, July 2002.

[5] A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," in *Proc. of third Intl. Workshop on Recent Advances in Intrusion Detection, LNCS*, Toulouse, France, Oct. 2000, pp. 80-92.

[6] W. Lee and S.J. Stolfo, "A framework for constructing features and models for intrusion detection systems.," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 227-261, Nov. 2000.

[7] W. Wang, X. H. Guan, and X. L. Zhang, "Modeling program behaviors by hidden Markov models for intrusion detection," in *Proc. of Intl. Conf. on Machine Learning and Cybernetics*, Shanghai, China, Aug. 2004, pp. 2830-2835.

[8] A. K. Ghosh, C. Michael, M. Schatz, and l, "A Real-Time Intrusion Detection System Based on Learning Program Behavior," in *Proc. of the third Intl. Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, Oct. 2000, pp. 93-109.

[9] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff, "A sense of self for Unix processes," in *Proc. of the 1996 IEEE Symp. on Security and Privacy*, Washington, DC, USA, May 1996, pp. 120-128.

[10] D. Y. Yeung and Y Ding., "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, pp. 229-243, Jan. 2003.

[11] H.J. Park S.B. Cho, "Efficient anomaly detection by modeling privilege flows using hidden Markov model," *Computers and Security*, vol. 22, no. 1, pp. 45-55, Jan. 2003.

[12] X. D. Hoang, Jiankun Hu, and P Bertok, "A multi-layer model for anomaly intrusion detection using program sequences of system calls," in *11th IEEE Conf. on Network*, Sep 2003, pp. 531-536.

[13] X. D. Hoang, J. Hu, and and P. Bertok., "A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference," *J. Netw. Comput. Appl*, vol. 32, no. 6, pp. 1219-1228, Nov. 2009.

[14] J. Hu, X. Yu, D. Qiu, and and H.H. Chen, "A simple and efficient hidden Markov model scheme for host- based anomaly intrusion detection," *Netwrk. Mag. of Global Internetwkg*, vol. 23, no. 1, pp. 42-47, Jan. 2009.

[15] G. Jiang and C. Ungureanu, and K.i Yoshihira H. Chen, "Multi-resolution Abnormal Trace Detection Using Varied-length N-grams and Automata," in *Proc. 2nd Intl. Conf. on Automatic Comp.*, Seattle, USA, June 2005, pp. 111-122.

[16] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179-211, April 1990.

[17] D. Yuxin, Y. Xuebing, Z. Di, D. Li, and A. Zhanchao, "Feature representation and selection in malicious code

detection methods based on static system calls," *Computers & Security*, pp. in-press, 2011.

[18] Tandon, G., "Machine Learning for Host-based Anomaly Detection," Florida Institue of Technology, Melbourne, Florida, USA, Ph.D. thesis 2008.

[19] M. Mahoney and P.Chan, "Learning rules for anomaly detection of hostile network traffic," in *Proc. 3rd IEEE Intl. Conf. on Data Mining*, Melbourne, Florida, USA, Nov 2003, pp. 601-604.

[20] S. Nakagawa, "A farewell to Bonferroni: the problems of low statistical power and publication bias," *Behavioral Ecology*, vol. 15, no. 6, pp. 1044-1045, 2004.

[21] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. NJ, USA: Lawrence Earlbaum Associates, 1988.

[22] A. Zahalka, K. Goševa-Popstojanova, and J. Zemerick, ""Empirical Evaluation of Factors Affecting Distinction between Failing and Passing Executions," in *Intl. Symp. on Soft. Reliability Engg.*, San Jose, USA, Nov. 2010, pp. 259-268.

[23] C. Yuan et al., "Automated known problem diagnosis with event traces," *SIGOPS, OS. Syst. Rev.*, vol. 40, no. 4, pp. 375-388, Oct. 2006.