Sequencegram: n-gram Modeling of System Calls for Program based Anomaly Detection

Neminath Hubballi, Santosh Biswas, Sukumar Nandi
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati, India
neminath@iitg.ernet.in

***Abstract*—Our contribution in this paper is two fold. First we provide preliminary investigation results establishing program based anomaly detection is effective if short system call sequences are modeled along with their occurrence frequency. Second as a consequence of this, built normal program model can tolerate some level of contamination in the training dataset. We describe an experimental system $Sequencegram$, designed to validate the contributions. $Sequencegram$ model short sequences of system calls in the form of n-grams and store in a tree (for the space efficiency) called as n-gram-tree. A score known as $anomaly\ score$ is associated with every short sequence (based on its occurrence frequency) which represents the probability of short sequence being anomalous. As it is generally assumed that, there is a skewed distribution of normal and abnormal sequences, more frequently occurring sequences are given lower $anomaly\ score$ and vice versa. Individual n-gram $anomaly\ score$ contribute to the $anomaly\ score$ of a program trace.**

***Index Terms*—Intrusion detection system, Program based anomaly detection, n-gram based analysis.**

I. Introduction

Intrusion Detection Systems (IDS) are an integral part of modern day security components. IDS are of two types signature based and anomaly based. Former method detects only known attacks and later can detect both known and new attacks. It is this ability of anomaly based techniques that lead for considerable research interest [1], [14], [10]. Anomaly detection techniques model the normal behavior and detect intrusions as deviations from this model. Modeling data can be derived from either network or from host. At the network level it can be (i) packet header modeling [8] (ii) payload modeling [9], similarly at host level it is (iii) system call modeling [6]. First two techniques are generally called as network intrusion detection techniques and third one is host based intrusion detection. Network based intrusion detection scans the input to the target application and host based (system call) intrusion detection system monitors the execution time behavior of the process/program. Normally network based anomaly detection techniques use either statistical measures of some events or use a data mining algorithm trained with normal data for detecting abnormal behavior. Host based intrusion detection model the sequence ordering of system calls generated by the program in execution thus it is also called as event order modeling which is the primary interest of this paper.

A system call is a routine that user programs call to get service of the operating system. Every user process when used under normal operating conditions (intended behavior) has a definitive sequence of system calls. However under abnormal operating conditions, i.e when an attacker tries to abuse the privileges, program takes code paths that are not seen before and results in generating system call sequences that are different from the sequences generated under normal conditions. This is the principle by which system call sequential data is labeled as either normal or abnormal. The idea of modeling system calls to detect abnormal executions of the programs was studied by Forrest et al [5]. Modeling system call ordering is useful in detecting privilege escalation kind of attacks where a user abuses her privileges and gains root access to the machine. For example these can be buffer overflow attacks. Table I shows a sample of system call data traces.

TABLE I
Example sequences

S_1	open, read, mmap, mmap
S_2	fstat, open, lseek, read, close
S_3	fstat, open, read, close

It is generally believed that, monitoring system calls is far more effective in detecting intrusions than monitoring network traffic. This is because of the fact that system call modeling is actually the program behavior modeling and in order to cause real damage to any system, programs on the target machine must behave in way that are not intended to behave. If the normal behavior of these programs is effectively represented any abnormal executions are detected. It also leads in generating minimal number of false positives compared to network based intrusion detection system.

It is observed that, system call sequences are highly localized i.e., the set of sequences generated depends on the configuration and environment where the program is running. Even same program running on same OS versions on different machines exhibit different set of sequences [4]. This results in target specific learning of normal behavior. Learning can be done either offline with collected system call traces or can be online where system learns as it encounters the execution traces live. Since every program on the target machine needs training and building a model, it is the second approach that is appealing. Online learning techniques have some limitations. Some of the limitations are

- Learning incompletely results into false positives: This

978-1-4244-8953-4/11/$26.00

indicates that before the IDS is put into use all possible legitimate execution paths must be captured.
- If learning is done in a vulnerable machine abnormal behavior can contaminate the normal profile and generate false negatives.

Thus it is necessary to build a stable and consistent profile which avoids these problems. Of the two issues first one can be avoided by incorporating more example sequences into training set. However addressing the second issue in an online learning system is not trivial. How does one get to know wether program is behaving normally or abnormally unless it is verified by some credible means. In this context it is useful to answer the question can we build models which can tolerate such accidental contamination of training dataset ?. We report preliminary investigation results to answer this question. Our contributions in this paper are following.

- It is established that short sequences (even of order 3) are good discriminators of normal and abnormal program executions when modeled along with their occurrence frequency.
- An efficient data structure called as n-gram-tree is used to build the normal profile of program execution.
- Proposed model is tolerant to some level of contamination in the training dataset.

The rest of the paper is organized as follows. In section II we review the seminal work of Forrest et al. and the follow up work in system call modeling for intrusion detection. In section III we highlight the steps involved in the proposed approach for system call sequence modeling. In section IV-A to IV-B3 we present the details of training phase and similarly in section V we describe the details of testing phase. We explore the asymptotic complexity of both training and testing phases in section VI. We discuss the experimentation done to verify the method in section VII. Finally the paper is concluded in section VIII.

II. Related Work

Literature related to intrusion detection on system call monitoring can be grouped into 3 categories as described in next 3 subsections.

A. Window based Methods

These methods generate a short subsequence from a sequence of data by sliding a window on the traces collected by program execution.

Forrest et al. [5] define normal behavior as short sequence of system calls in a running process. A database of such short sequences is initially built by sliding a window of length k over the trace. Anomalies are detected if there are sufficiently new (which are not seen) subsequences of length k in the testing sequence.

Another variation of the above method called STIDE [7] also extracts short subsequence of system calls with a window of length k for building the database of normal sequences. Given a test sequence a rate $S^{anomaly}$ is assigned to it which indicates how anomalous is a given sequence. It used inverse of the hamming distance normalized to the window length as the anomaly score.

A window based method to automate the intrusion response is proposed in [15]. It maintains two profiles one containing the set of normal system call sequences in the form of short sequences of length k and another having set of sequences appearing in testing case. As the occurring system calls in the test sequence drift from the normal profile the system delays execution of system calls using an exponential function. This delay frustrates attacker and hence prevents compromising the machine.

A variable length window sizes which can best predict the next symbol in the sequence have been explored in [11], [19].

B. State based Models

A predictive modeling and a context dependent optimal window selection is proposed in [2]. System call traces are represented in the form a call graph and modeled using Sparse Markov Transducers (SMT). Given the $n-1$ previous system calls it will predict the probability of n^{th} system call. If this predicted probability is less than a threshold system call is declared as abnormal.

An automatic Finite State Machine (FSM) generation technique is proposed in [12]. FSM is constructed using the audit data for a program. FSM represents the n-grams present in the training data sequences. In the testing phase it detects new set of n-grams which are not seen before. An alternative predictive model which predicts the next sequence based on the probability of sequences in the training dataset is also proposed. During training probability distribution of training sequences is calculated and during testing any deviation from this estimated distribution is considered as anomaly.

C. Complementary Models

Nguyen et al. in [13] establishes a relationship between user profile and various processes and programs. It was noticed that some of the system users have remarkable consistency in accessing the files and other programs that they run.

A complete summary of all the follow up work of Forrest et al. on system call monitoring for abnormal program behavior detection can be found in [4].

III. Proposed Method

In this section we describe the proposed scheme $Sequencegram$ which is an n-gram based model for detecting intrusions based on system calls. Unlike models found in the literature $Sequencegram$ models the frequency information associated with each short sequence of system calls appearing in the normal execution profile of a program.

The proposed method $Sequencegram$ involves training phase and testing phase as discussed bellow.

- **Training Phase**
  - *Tree Construction:* System call traces are divided into short sequences by sliding a window of varying size. The size varies between 1 to N. These short

sequences are called as n-grams. Training phase basically involves building a n-gram-tree with n-grams generated out of training traces where some of the traces may belong to improper execution of program. A node at a particular level i ($1 \leq i \leq N$) of the tree along with all nodes along the path from root to i captures a distinct n-gram of order i found in the training dataset with its frequency of occurrence. This frequency is used to assign an $anomaly\ score$ to n-gram which indicates how anomalous the short sequence is.

  - *Binning:* Rather than assigning individual $anomaly\ score$ to every n-gram, these n-grams are grouped and bins are created. Binning is based on their frequency of occurrence. Certain frequency bands (or windows) are selected and assigned a single $anomaly\ score$ to the bin. All n-grams whose frequency lies within the band of a bin are put into the bin. All n-grams in a bin receive the $anomaly\ score$ of the bin. This involves three sub-steps: (i) selection of appropriate number of bins (i.e., size of the ranges) (ii) distributing n-grams to bins (iii) assigning $anamoly\ score$ to bins.

- **Testing Phase**
  - n-grams of desired order are generated from the test sequence of a program run and the n-gram-tree is searched for each n-gram. Frequency of n-gram in the training dataset is used to determine to which bin the n-gram falls and what is the corresponding $anomaly\ score$ given to that bin.

The training and testing phases are elaborated in sections IV-A to V.

IV. Training the Model

This phase as stated earlier has 3 sub tasks (i) n-gram-tree construction (ii) frequency binning and (iii) Assigning $anomaly\ score$ to the bins. These three sub steps are elaborated bellow.

A. n-gram-tree Generation

As mentioned in the last section, n-grams are generated by sliding a window of varying size between 1 to N. Once n-grams of all order between 1 to N are ready, n-gram-tree is constructed. The idea of n-gram-tree is motivated by the antimonotonicity property of n-grams which says, there can not be an n-gram of higher order with higher frequency than that of its constituent lower order n-grams. Literature in data mining field call it as hash tree. A node at level i is of the form $< a_i, F >$ ($1 \leq i \leq N$) where a_i is i^{th} system call and F is frequency of occurrence of n-gram involving all nodes from root to node i. Since the common prefixes get merged it forms an efficient storage structure.

The n-gram-tree consist of a dummy root node (it does not hold a system call value and frequency). A node x at level i along with all nodes in the path from root to x represents an n-gram of order i. For example, let $\langle x_0, x_1, x_2, \cdots, x_i \rangle$ be the path from root x_0 to x_i, a node at level i. This path represents an n-gram (of order i) $\langle a_1, a_2, \cdots, a_i \rangle$, where each a_t ($1 \leq t \leq i$) is the short sequence of system call corresponding to the node x_t. The same path also includes smaller n-grams e.g., the sub-path $\langle x_0, x_1, x_2 \rangle$ represents the n-gram (of order 2) $\langle a_1, a_2 \rangle$. So, nodes of the first level represents all the n-grams of order 1. Similarly, nodes of first level along with nodes of the second level form n-grams of order 2. In general, nodes at level i together with nodes at levels ranging from 1 to $i-1$ represent the n-grams of order i.

The procedure for constructing n-gram-tree is shown in the form of Algorithm 1 and Algorithm 2. Algorithm 1 takes as input, the number of traces available for training and generates sets of n-grams up to desired order i.e. N from each trace and pass it to Algorithm 2 for n-gram-tree construction. Algorithm 2 reads set of n-grams of a particular order starting from 1 to N in the increasing order and add or update the nodes of n-gram-tree as required.

Algorithm 1: **Algorithm for generating n-grams**

Input : $tracecount$ - number of system call traces
Input : $S_1, S_2, \cdots, S_{tracecount}$ traces available for training.
Input : N - highest order n-grams to be generated.
Output : Set of n-grams of order 1 to N.

1: **for** $I = 1$ to $tracecount$ **do**
2: $tracelength \leftarrow$ Length of trace I
3: **for** $J = 1$ to $tracelength$ **do**
4: Generate n-grams of order 1 to order N and store
5: **end for**
6: Merge all same n-grams of order i to a single n-gram of format $\langle a_1, \ldots, a_i, F \rangle$ where F is the frequency of n-gram
7: P_k is set of n-grams of order k generated from S_I.
8: Call Algorithm-2 with P_k, $1 \leq k \leq N$ as input.
9: **end for**

Algorithm 2: **Algorithm for constructing n-gram tree**
INPUT: P_k, $1 \leq k \leq N$ (Algorithm 1)
Output: n-gram-tree.

1: Create a dummy $ROOT$ node.
/*Level 1 construction */
2: **for** Every $p \in P_1$ where each p is of the form $\langle a_1, F \rangle$ **do**
3: **if** There is no child for $ROOT$ with value $\langle a_1 \rangle$ **then**
4: Add a child to $ROOT$ and set node value to $\langle a_1 \rangle$ and Frequency to F
5: **else**
6: Find the child node x of $ROOT$ with value $\langle a_1 \rangle$
7: increment Frequency of x by F
8: **end if**
9: **end for**
/*End of Level 1 construction*/
/*Level 2 construction */

10: **for** Every $p \in P_2$ where each p is of the form $\langle a_1, a_2, F\rangle$ **do**
11: Find the child node x of $ROOT$ with value $\langle a_1\rangle$
12: **if** There is no child for x with value $\langle a_2\rangle$ **then**
13: Add a child node to x and set its value to $\langle a_2\rangle$ and Frequency to F
14: **else**
15: Find the child node y of x with value $\langle a_2\rangle$
16: increment its Frequency by F
17: **end if**
18: **end for**
/*End of Level 2 construction*/
19: Similarly, level 3 nodes can be formed using P_3 and level 1 and level 2 nodes. The same procedure is repeated for the desired level N.

The n-gram-tree represents the normal execution profile of the program to which the traces belong. The way n-gram-tree is constructed it allows to build the tree incrementally probably online, before using it to detect abnormal program execution sequences that is one of the key advantage of model.

Construction procedure of n-gram-tree is explained with an example of a trace having 18 system call (i.e., $tracecount = 1$ and $tracelength = SP_1 = 18$). Let the sequence of system calls have the numbers[1] $1, 2, 5, 2, 3, 4, 1, 4, 2, 1, 2, 5, 2, 3, 4, 1, 4, 2$. Step 4 of Algorithm 1 generates the following n-grams of order 1: $\langle 1, 1\rangle$, $\langle 2, 1\rangle$, $\langle 5, 1\rangle$ $\cdots$ $\langle 2, 1\rangle$. Step 5 of Algorithm 1 merges the same n-grams finally generating $P_1 = \{\langle (1), 4\rangle, \langle (2), 6\rangle, \langle (3), 2\rangle, \langle (4), 4\rangle, \langle (5), 5\rangle\}$. In a similar way $P_2 = \{\langle (1, 2), 2\rangle, \langle (2, 5), 2\rangle, \langle (5, 2), 2\rangle, \langle (2, 3), 2\rangle, \langle (3, 4), 2\rangle, \langle (4, 1), 2\rangle, \langle (1, 4), 2\rangle, \langle (4, 2), 2\rangle, \langle (2, 1), 1\rangle\}$. Similar steps are followed for other higher levels.

The n-gram-tree (up to order 3) for this trace is shown in Figure 1. All the n-grams order 1, in P_1 form the level 1 nodes. Now let the first n-grams of order 2 $\langle (1, 2), 2\rangle$ (from P_2) be considered; here $a_1 = 1$, $a_2 = 2$ and $F = 2$. At level 1 the leftmost node (x_{11}) has the value equal to $a_1 = 1$. From x_{11} there is no child. So a child x_{21} is created from x_{11} and its value is set to 2 i.e., $a_2 = 2$ and frequency equal to $F = 2$. This procedure will be repeated for all the n-grams of order 2 in P_2. For layer 3 also same procedure is used with P_3.

B. Frequency Binning

The idea is to assign a single $anomaly\ score$ to a range of nearby frequencies. The $anomaly\ score$ here simply refers to how probable a particular n-gram is anomalous, higher the value more likely it is from a anomalous sequence and vice versa. The basic idea of using n-gram-tree for attack detection is based on the observation that, system calls generated by a particular program is highly consistent and forms a small portion of over all possible sequences of a particular order [4]. Higher frequency n-grams typically correspond to normal behavior. Similarly, n-grams of lower frequencies are probably from anomalous sequences (this holds good only if the assumption of proportion of number of normal and abnormal sequences in training dataset is valid).

Frequency binning although assigns a single $anomaly\ score$ to a range of n-grams, it results into some space saving compared to individual score for individual n-grams. Individual score can be assigned by having an additional field in each node of the tree holding that value. The reason which drives such a common score assignment to a range of n-grams is the fact that scores of n-grams whose frequencies are nearby may not vary by a big margin. There are three sub steps involved in frequency binning (i) selecting appropriate number of bins for the frequency range, (ii) distributing n-grams to bins and (iii) assigning a suitable $anomaly\ score$ to the bin. These sub steps are elaborated bellow.

1) Selecting Number of Bins: We use Sturges's formula [16] for finding the appropriate number of bins. Given the number of elements in a dataset, Sturges' formula determine the appropriate number of bins. Calculating number of bins denoted as $numberbins$ is given by Equation 1.

$$numberbins = \lceil (1 + \log_2(Distinct\ n - grams)) \quad (1)$$

2) Distributing n-grams Among Bins: Once the appropriate number of bins are determined, n-grams have to be distributed among the bins. One way to do it is distribute available n-grams equally among all the bins. However this may lead to improper grouping. For instance two n-grams which are having large variation among their frequencies may be put into same bin. For example if there are 6 n-grams and there are 2 bins and frequencies of n-grams are 100, 98, 92, 89, 18, 16. Equal distribution will group 100, 98 and 92 into one group and remaining into other group. It means that the n-gram with frequency of 18, 16 and 89 will receive the same anomaly score. This leads to improper grouping and affects $anomaly\ score$. The improper grouping defeats the vary reason of assigning a single $anomaly\ score$ to the bin. This n-gram would have been better grouped into the first bin. Thus it is required to minimize the variation among the frequencies within a bin and this necessitate unequal distribution. In order to distribute and minimize the frequency variation we used K-means clustering [17] algorithm and found the clusters of frequencies. Informally clustering groups logically similar objects together. K-means clustering algorithm finds K clusters (given the value of K) from training dataset. K in our case is the $numberbins$ obtained in equation 1. K-means is an optimization algorithm which minimizes an objective function called squared error function. It starts with randomly choosing K distinct cluster centers, which represent K clusters and assign available objects in training set to a nearest cluster. After assignment, cluster centers are calculated again by the average of all objects assigned to cluster and starts reassigning objects to these newly created cluster centers. Generating new cluster center and reassignment continue till there is no change in cluster centers and the assignment of objects. For an algorithmic representation of K-means algorithm readers are

[1]Assuming actual system calls are mapped to numbers

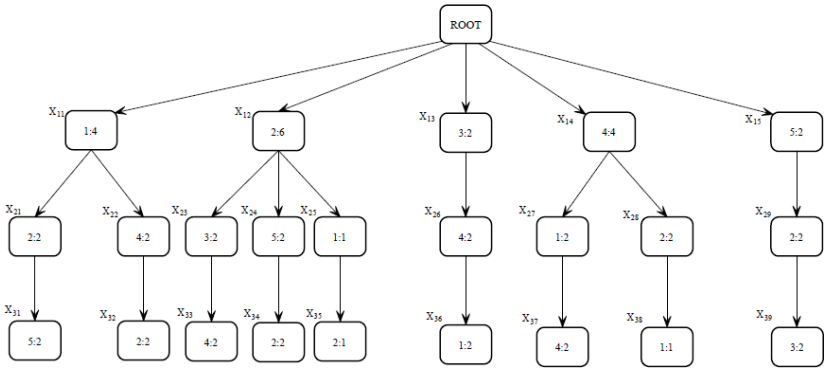

Fig. 1. n-gram-tree of order 3 built with a training traces

referred to Appendix -A. Once the clusters of frequencies are formed range of frequency of a particluar cluster is determined by the minimum and maximum frequency value in the cluster. All n-grams whose frequencies fall within the range of a particular bin is assigned to that bin. Likewise all n-grams obtained after traversing the n-gram-tree are assigned to one of the bin.

3) Assigning $anomaly\ score$ to Bin: After the n-grams are distributed to bins $anomaly\ score$ is assigned to the bin. To assign $anomaly\ score$ to bin t, we use the rating function given by Equation 2. where $f(< a_1, \cdots, a_n, F)$ is a function which returns the frequency of n-gram $< a_1, \cdots, a_n, F >$ and $bin(t, < a_1, \cdots, a_n, F >$ is another function which returns the frequency of n-gram if the particular n-gram is within the bounds of frequencies of bin t otherwise it returns 0.

This equation assigns as $anomaly\ score$ to bin t, calculated as the ratio of sum of all frequencies of the n-grams of order N over all traces to the sum of all frequencies of n-grams of order N falling in bin t.

The intuition of $anomaly\ score$, is derived from the previous discussion that the more frequently the n-gram sequence appears it is probably from normal execution and otherwise. In order to understand our scoring function does what is expected to do we present some example scores obtained in one of our experiments. Table II shows the $anomaly\ score$ assigned to various bins. The first column of the table shows the total sum of frequencies of all the n-grams, second column shows the sum of frequencies of n-grams falling in the corresponding bin, third column shows the number of n-grams in that bin and the last column shows the $anomaly\ score$ assigned to that bin. It can be inferred from table that, bins which have n-grams with less sum of frequencies are assigned higher $anomaly\ score$ and it gradually decreases as the frequency sum of the bin increases. This confirms with our objective of assigning $anomaly\ score$ based on frequency.

TABLE II
EXAMPLE $anomaly\ Score$ GENERATION AND ASSIGNEMENT

Grand Sum	Bin Sum	Number of n-grams	Rate
2920898	848	72	8.14
	2393	19	6.90
	7645	11	5.94
	20540	21	4.96
	65787	35	3.78
	100632	16	3.36
	173767	64	2.77
	2449417	2	0.17

V. TESTING PHASE

In the testing stage every trace given is evaluated against the built model to decide wether the trace belongs to some legitimate execution of the program or it is suspicious. Given a trace sequence S, n-grams of order N are generated and searched in the n-gram-tree and an $anomaly\ score$ of individual n-grams are generated and these scores contribute to the $anomaly\ score$ of the trace denoted as $S^{anomaly}$. It can be noticed that, as the scores of individual n-grams depend on frequency of n-grams, $S^{anomaly}$ too depends on the frequency of its constituent n-grams. If a particular n-gram is present in the n-gram-tree we can find a path from root node to it in the tree otherwise there will not be any path.

Once the node x at level N is identified for a particular n-gram the corresponding frequency is used to decide the bin to which it falls. The $anomaly\ score$ of the n-gram in question is given the $anomaly\ score$ of the bin. In order to calculate the $anomaly\ score$ of trace S, scores of individual n-gram is added to a running sum of current trace. If a particular n-gram is not there in the training database (i.e., in the n-gram-tree) an $anomaly\ score$ which is twice the maximum of score of all the bins is assigned for it. Finally the sum of $anomaly\ score$ is averaged over the length of the trace. If this average $anomaly\ score$ exceeds a preset threshold[2] τ, the sequence is declared as abnormal. Algorithm 3 shows the testing operation.

Algorithm 3: **Testing a Sequence**

INPUT: n-gram-tree, SP-testing trace, τ-anomaly threshold
OUTPUT: Status for SP

[2]How this threshold is selected is described in the experimentation section.

$$rate[t] = log_e\Big(\frac{\sum_{i=1}^{tracecount}\sum_{j=1}^{tracelength} f(< a_1, \cdots, a_n, F >)}{\sum_{i=1}^{tracecount}\sum_{j=1}^{tracelength} bin(t, f(< a_1, \cdots, a_n, F >))}\Big) \quad (2)$$

1: $tracelength \leftarrow$ Length of trace SP
2: **for** $J = 1$ to $tracelength$ **do**
3: Generate all n-grams of order N from SP, say $p_1, p_2, \cdots, p_{numberngrams}$
4: Search a path from root to a node p_J at N^{th} level in the n-gram-tree. Let x be the node found for p_J. (x may be NULL, if no match is found)
5: $anomaly\ score$ of p_J is $anomaly\ score(bin(f(x)))$.
6: **end for**
7: $S^{anomaly} = \frac{\sum_{J=1}^{numbergrams} anomaly\ score\ of\ p_J}{numbergrams}$
8: **if** $S^{anomaly} \geq \tau$ **then**
9: Declare SP abnormal
10: **else**
11: Declare SP normal
12: **end if**

In the above algorithm the function $anomaly\ score(bin(f(x)))$ returns the $anomaly\ score$ of particular n-gram found in the testing sequence. It involves determining frequency of n-gram, then finding bin to which it falls and assigning $anomaly\ score$ to it.

VI. Complexity Analysis

In this section we analyze the time and space complexity of both training and testing phases. This analysis is purely of theoretical interest, since usefulness of scheme solely depend on the practical time and space complexity. We show in section VII that practical time and space complexity of our algorithm is orders of magnitude smaller compared to upper bound derived in this section. Here we study these upper bounds for the sake of completeness.

To analyze time and space complexity following notations are used.

A	= alphabet of system calls for a program
$size$	= cardinality of set A
$tracecount$	= number of program traces available for training
S	= set of training traces $S_0, S_1, \cdots, S_{tracecount}$
$tracelength$	= length of the program trace
N	= order of n-grams to be generated

A. Training Phase

Training phase involves n-gram-tree construction, frequency binning and is done with Algorithm 1, Algorithm 2 and K-means clustering. Thus the complexity of both the algorithms plus the complexity of K-means clustering algorithm is the complexity of training phase.

Algorithm 1 Time Complexity

From Algorithm 1 the following can be noticed. Step 1 loops through 1 to $tracecount$, step 2 involves a constant time operation. Step 3 loops through 1 to $tracelength$. Step 4 generates the n-grams of desired order and hence involve complexity of $O(tracelength)$. This is because entire trace has to be read once to generate n-grams. Further it has to be noted that, a trace of length $tracelength$ generates nearly $tracelength$ number of n-grams. Step 6 merge all similar n-grams and hence needs comparison of one n-gram with all others hence add complexity of $O(tracelength)^2$. Step 7 (i.e., Algorithm 2) involves $O(\sum_{I=1}^{N}((tracelength).size.I))$ (will be shown in complexity analysis of Algorithm 2). So, overall complexity of Algorithm-1 and the training phase can be written as $O((tracecount).\{N.(tracelength + (tracelength)^2) + \sum_{I=1}^{N}((tracelength).size.I)\})$.

Algorithm 2 Time Complexity

This algorithm constructs n-gram-tree layerwise. Detailed analysis of complexities are as follows. Step 1 is of $O(1)$ complexity.

First level of tree construction involving steps 2 to step 9 has a complexity of $(O(size.(tracelength)))$ which can be explained as follows. Step-3 and Step-6 require comparing values of a n-gram (of order 1) with all nodes of level 1, this involves complexity of $O(size)$. Steps-4 and Step-7 have constant time complexities. Steps 3 to Step-7 are to be repeated for all $p \in P_1$. So overall complexity of Step-2 to Step-9 is $O(size.(tracelength))$.

Similarly steps 10 to step 18 involves the construction of second level of n-gram-tree. This involves a search on the first system call of n-grams of order 2 at level 1 of the n-gram-tree which has $size$ number of nodes at maximum and search operation of second system call at a child of first level node which again can have $size$ number of nodes in the worst case. This leads to overall complexity of $O((tracelength).(size + size)) = O((tracelength).size.2)$. This analysis can be generalized to any desired order N as $O((tracelength).size.N)$.

Time Complexity of Binning

Binning requires frequencies of all n-grams to be collected and this involves traversing n-gram-tree. There are $size$ number of nodes at level 1 hence add a complexity of $O(size)$. Each node at first level can have another $size$ number of children thus there can be $size^2$ number of nodes at level 2 thus adding $O(size^2)$ and so on up to $O(size^N)$ at level N. Thus the overall complexity is $O(1 + size + size^2 + \cdots + size^N)$ which is a geometric series and can be simplified to $O\Big(\frac{size^N - 1}{size - 1}\Big)$.

Time Complexity of Frequency Clustering

Our input to clustering algorithm is frequencies of n-grams. K-means algorithm has a time complexity of $O(K.T.Y.t_{dist})$ where K is the number of clusters (represented by K centroids), T is number of objects, Y is number of iterations, t_{dist}

is the time taken to compute one distance for d dimensional object. In our case K is $numberbins$, T is number of distinct n-grams which is again equal to number of nodes at the N^{th} level in n-gram-tree. In the worst case number of nodes at N^{th} level can be $size^N$. Our data to clustering algorithm is single dimensional, and if time taken to compute distance (which is just one substraction) is neglected, complexity of clustering can be written as $O((numberbins).size^N.Y)$.

Space Complexity of Training Phase

From the algorithm it is to be noted that, built profile is n-gram-tree. The space complexity of training phase is storage space required for n-gram-tree. We note that there can be $size$ number of nodes at level 1 thus a space complexity of $O(size)$ and each of these nodes can have another $size$ number of nodes in the worst case and hence there can be $O(size^2)$ number of nodes at level 2, this continues up to desired level i.e., n. Thus the overall space complexity is $O(1 + size + size^2 + \cdots + size^N)$ which can be rewritten (it is a geometric series) as $O\left(\frac{size^N - 1}{size - 1}\right)$.

B. Testing Phase

Algorithm 3 discussed above is used for testing the trace. Its complexity analysis is followed.

Time Complexity of Algorithm 3

Step 1 is of constant time complexity and step 2 loops through 1 to $tracelength$. Step 3 of algorithm generates n-grams of all the order in one reading of trace hence has a complexity of $tracelength$. For all the n-grams steps 4 and step 5 are repeated. Step 4 involves searching for a n-gram at level N which has complexity of $O(size.N)$. Once the bins are formed it is trivial to calculate the $anomaly\ score$, hence let us assume step 5 has a constant time complexity. Step 6 involves adding of individual $anomaly\ score$ and calculating average, this needs $numbergram$ additions and one division, thus this adds a complexity of $O(numbergrams)$. Remaining steps are of constant time complexity. Hence, the overall complexity of testing phase can be written as $O((tracelength) + ((numbergrams).(size.N)) + (numbergram))$.

VII. Experimental Results

A. Data Sets

We report the experimental results on the University of New Mexico (UNM) dataset available at [3]. This dataset consist of several program traces collected during the live execution of previledged programs in production systems. Every trace consist of complete listing of system calls made during execution of the program from start to end. Following 8 programs traces are available in the dataset - lpr, $named$, $xlock$, $login$, ps, $inetd$, $stide$ and $sendmail$.

There are two types of traces available for evaluation, some having both normal and intrusion traces and others having only normal traces. Out of these 8 program types 7 are having both types of traces and for another program only normal traces are available. Characteristics of traces is shown in Table III.

Out of 8 programs mentioned in Table III lpr program traces were collected at two locations one at University of New Mexico (UNM) and the other at Massachusetts Institute of Technology (MIT). MIT dataset has two weeks of normal activity and UNM has 3 months of normal activity. Both lpr programs have 1001 traces of a single intrusion. For $named$ program there is a single huge file which has traces of a single daemon and 26 of its subprocess collected over a month and also there are 2 files of two intrusions. The program $xlock$ has a single live trace collected over a weekend and 2 files of 2 intrusion traces. Also there are other 71 synthetic traces collected with the hep of a script, designed to exercise commands of $xlock$ program. There are 2 files containing 12 live normal traces and 2 files having 9 traces of intrusions for $login$ program. In case of ps too there are 2 files containing 24 normal traces and 2 files having 26 traces of intrusions. Program $inetd$ has a single file for normal execution containing traces of a startup process, a daemon process and several child processes which are exactly identical (hence considered 1) and another file for abnormal execution having 31 traces. Another program $stide$ has 13726 files of those many normal traces and a single file containing 105 traces of an attack. The last program $sendmail$ has 71760 normal traces and there are no intrusion traces. All traces except UNM live lpr shown in Table III were used in the experimentation of [18]. UNM lpr link having newer version of live traces is broken, so we used an older version of UNM lpr traces which were used in the experimentation of [7].

TABLE III
DATASET CHARACTERISTICS

Program	Number of Normal Traces	Number of Intrusion Traces
MIT lpr	2703	1001
UNM lpr	1232	1001
xlock real	1	2
xlock synthetic	71	2
named	27	2
login	12	2
ps	24	26
inetd	3	31
stide	13726	105
sendmail	71760	0

B. Implementation and Evaluation

Our objective is to study the performance of $Sequencegram$ in terms of $False\ Positive\ Rate$ and $Detection\ Rate$ it generates. For evaluation purpose we implemented $Sequencegram$ as a single C program which involves all the steps namely n-gram generation, tree building, frequency binning using K-means clustering and testing. We decided to release our proof of concept version of software freely under GNU public licence. This helps in easier replication of work as the dataset is also available openly. Current un optimized version of implementation (software) can be obtained from the URL "www.iitg.ernet.in/stud/neminath".

Our algorithm discriminates between normal and abnormal sequences based on $anomaly\ score$ assigned to the se-

quences. Choosing an optimal value for threshold τ is critical for the discrimination (performance) of IDS ($Sequencegram$). How to chose a value is a study in itself and is beyond the scope of present discussion. In this paper we adapted two different methods of threshold selection for two sets of programs hence we report results for two sets of programs separately. In the first category we report results of those programs which have both normal and abnormal program run, traces and in the second category we report results for programs which have only normal traces. In order to avoid any un due biases in the experimental results we performed 10 fold cross validation for all of our experiments and also the sequence length N was set to 3.

In the experiments of first category of programs which have both normal and abnormal traces we did some modifications to the dataset for experimentation purpose. These modifications either omit certain category of traces or modify them in order to help conducting cross validation. These modifications are elaborated bellow.

As there is a single normal file for live trace collected over a long period for $named$ and $xlock$ programs, we divided them into 462 and 220 smaller traces respectively. This division is simply to perform cross validation and was possible because these traces are highly consistent and extremely repetitive. We omit reporting results on $login$, ps, $inetd$ and $stide$ programs because of two reasons, we report results for individual traces of intrusions and in these more than one intrusion traces are mixed into a single file and are not divisible. Second some of these programs have very few normal traces to learn the frequency information of n-grams which are necessary for our algorithms to work. It has to be noted that, dividing normal trace do not hurt because as long as any portion of trace is part of a normal sequence it is still normal. However an abnormal sequence may contain only a small portion where it is deviating from normal and if it divided we miss its detection.

In this set of experiments we varied threshold and set it for the minimum value necessary to detect all abnormal traces (thus bringing $Detection\ Rate$ to 100%) and studied the variation of $False\ Positive\ Rate$. Table IV reports the $False\ Positive\ Rate$ for the program traces when the $Detection\ Rate$ is 100%. As we can notice $False\ Positive\ Rate$ is 0 for UNM lpr, $named$ and $xlock\ real$ program traces and it is very low (1.7%) for MIT lpr dataset. Since $False\ Positive\ Rate$ is either 0 or has a very low value for most of the cases it is evident that short sequences when modeled along with their occurrence frequency can discriminate normal and abnormal program executions.

Similar to above experiments for the case of $sendmail$ program which has only normal traces we did 10 fold cross validation. Since there are no intrusion traces there is no $Detection\ Rate$, thus we used a different technique for evaluation. In 10 fold cross validation, dataset is divided into 10 equal parts out of which 9 are used for training and remaining 1 is used for testing. In this case we randomly divided one unit which is being tested into two parts. One unit has 25% of test sequences called as validation unit and other has 75% of sequences as real test unit. To chose threshold τ we generated $anomaly\ score$ for every sequence in the validation unit and found the maximum $anomaly\ score$ among all sequences and set it as threshold τ. This means that, to declare all of the sequences in validation unit as normal, threshold has to be at the minimum equal to the maximum $anomaly\ score$ generated by this unit. Using this as a baseline other unit is evaluated and noticed 0.001% $False\ Positive\ Rate$ on this.

TABLE IV
$False\ Positive\ Rate$ FOR 10 FOLD CROSS VALIDATION EXPERIMENT FOR 100% $Detection\ Rate$

Program	Number of Normal Traces	Number of Intrusion Traces	$False\ Positive\ Rate$
MIT LPR	2703	1001	1.70%
UNM LPR	1232	1001	0.00%
xlock real	220	2	0.00%
named	462	2	0.00%

In order to study the performance of $Sequencegram$ for training dataset contamination, we conducted a series of experiments with 10 fold cross validation. We report the results on first unit of traces having both normal and abnormal program sequences. In these experiments, to study the impact of contamination on of training dataset, each program dataset is contaminated by adding some traces of abnormal (intrusion) execution sequences. We used the same threshold selection method as used for the first set of experiments. Table V reports the $False\ Positive\ Rate$ for various level of contamination for MIT lpr program. We can notice that, initially for up to addition of 100 abnormal traces, $False\ Positive\ Rate$ is not affected and beyond that it is increasing. At this stage frequency difference between normal and abnormal execution traces is still so wide that it is able to differentiate. As more and more abnormal traces are added frequencies of n-grams in abnormal sequences increase and hence they will be elevated to higher order bins and subsequently assigned a lower $anomaly\ score$. By virtue of this a number of misclassifications are made by the algorithm and hence $False\ Positive\ Rate$ increases. To understand the difference in frequencies of n-grams, a comparison of frequencies is made. Figure 2 shows the comparison of frequencies of n-grams generated by abnormal sequences but seen in pure normal training dataset, training dataset contaminated with 50 abnormal traces and training dataset contaminated with 1001 abnormal traces. As seen from the figure, there is a slight elevation in the beginning by the addition of 50 abnormal traces and beyond that, there is no change in frequencies seen in normal and contaminated with 50 traces dataset. However when the dataset is contaminated with 1001 abnormal traces there is a clear surge in frequencies. This is the cause of huge false alarms. From the Table V it can be noticed that, increase in the $False\ Positive\ Rate$ is not gradual but abrupt. Initially it is low, however as more and more abnormal traces are added the effect becomes severe.

Similarly Table VI reports the variation of *False Positive Rate* for different level of contamination in case of UNM *lpr* program. Table VII and Table VIII show the *False Positive Rate* variation for *named* and *xlock* program traces respectively. We can notice from these tables that, trends seen with MIT *lpr* experiments translated into other programs too. Programs *xlock* and *named* have only two abnormal traces (which are long sequences) did not affect the performance, however no conclusive remarks can be made about these two as there are not enough traces to completely study the characteristics and performance. However this set of experiments do indicate that, built program profile by *Sequencegram* can tolerate some level of contamination.

TABLE V
False Positive Rate FOR 10 FOLD CROSS VALIDATION OF MIT LPR IMPURE DATASET WHEN *Detection Rate* IS 100%

Number of Normal Traces	Number of Intrusion Traces Added	Number of Intrusion Traces	*False Positive Rate*
2703	25	1001	1.70%
2703	50	1001	1.70%
2703	100	1001	1.70%
2703	200	1001	1.82%
2703	300	1001	1.82%
2703	500	1001	3.41%
2703	1001	1001	99.80%

TABLE VI
False Positive Rate FOR 10 FOLD CROSS VALIDATION OF UNM LPR IMPURE DATASET WHEN *Detection Rate* IS 100%

Number of Normal Traces	Number of Intrusion Traces Added	Number of Intrusion Traces Tested	*False Positive Rate*
1232	25	1001	0.00%
1232	50	1001	0.00%
1232	100	1001	1.45%
1232	200	1001	1.79%
1232	300	1001	4.64%
1232	500	1001	14.79%
1232	1001	1001	100.00%

TABLE VII
False Positive Rate FOR 10 FOLD CROSS VALIDATION OF *named* IMPURE DATASET WHEN *Detection Rate* IS 100%

Number of Normal Traces	Number of Intrusion Traces Added	Number of Intrusion Traces Tested	*False Positive Rate*
462	1	2	0.00%
462	2	2	0.00%

C. Practical Time and Space Complexity

First we report results on practically how small the n-gram-tree is compared to possible size, by analyzing number of nodes created at each level. The space complexity analysis, creates a basis for practical execution time of the program.

TABLE VIII
False Positive Rate FOR 10 FOLD CROSS VALIDATION OF *xlock real* IMPURE DATASET WHEN *Detection Rate* IS 100%

Number of Normal Traces	Number of Intrusion Traces Added	Number of Intrusion Traces Tested	*False Positive Rate*
220	1	2	0.00%
220	2	2	0.00%

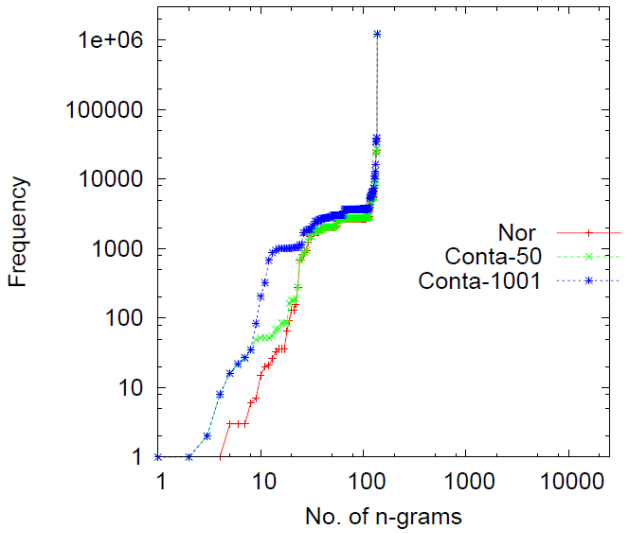

Fig. 2. Frequency comparison of n-grams from intrusion sequences, for normal, when contaminated with 50 packets and when contaminated with 1001 packets

Table IX shows number of nodes i.e., n-grams generated at different levels. Nodes at level 1 represents n-grams of order 1 and nodes of level 1 and level 2 represents n-grams of order 2 and so on. From the table we note that UNM *lpr* program has just 15 distinct system calls and hence there are only 15 nodes at first level. This also indicates alphabet size *size* becomes 15. At level 2 we can have $size^2$ number of nodes which is 225 but there are only 71 nodes. At level 3 we can have $size^3$ number of nodes which is 3375, but the number of nodes present is 118. This indicates that there are only 118 different n-gram of order 3, in the entire training dataset which is 3.5% of possible n-grams. Program *named* real dataset has 41 nodes at level 1, thus those many n-grams of order 1 and at level 2, there are 212 nodes instead of 1681. Similarly for level 3, there are only 433 nodes and hence those many distinct n-grams of order 3 instead of 68921 and is 0.62% of the upper bound. From the table we can infer that, these trends again translated into other programs too.

We also report the practical performance of *Sequencegram* in training and testing phases. Our proof of concept un optimized version of *Sequencegram* process 1234 UNM *lpr* training traces in 60ms (build the n-gram-tree, does frequency binning and calculating the *anomaly score* of each bin). It also tests 1001 intrusion traces in 200ms time. Algorithm takes

TABLE IX
SHORT SEQUENCE GROWTH

Trace	n-gram 1^{st} order	n-gram 2^{nd} order	n-gram 3^{rd} order
MIT lpr	41	141	250
UNM lpr	15	71	118
named real	41	212	433
xlock real	36	111	188

more time in testing phase, since some of the n-grams do not find paths in the tree from root and all possible paths of tree has to be searched but in case of training n-grams repeat and quickly find a path. Similar trends were translated into other programs too. These experiments were done on a Intel (R) Core (TM) 2 Duo CPU running at 3 GHz and having 2 GB RAM, running Ubuntu 9.2 operating system.

VIII. CONCLUSION AND FUTURE WORK

In this paper we described $Sequencegram$ which is a program based anomaly detection technique. $Sequencegram$ uses n-gram modeling of short sequences of system calls along with the occurrence frequency in the training traces. Our experimental evaluation with $Sequencegram$ revealed that, short sequences (even of order 3) are good discriminators between normal and abnormal traces when short sequences are modeled with their occurrence frequency. The biggest advantage of this modeling is that, detection becomes immune to accidental contamination of training dataset. An efficient tree storage structure named as n-gram-tree was used for creating normal profile. Practical sizes of n-gram-tree and operational speeds of proof of concept implementation were found to be reasonable for all practical purposes. Time and space complexity of training and testing phases were analyzed and it was noticed that, theoretical limits are many fold higher than practical limits.

It has to be noted that, our scoring function discretise the scoring. A comparative study need to be done wether there is a tolerance penalty with this, compared to absolute scoring where individual n-grams are given separate $anomaly\ score$.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Survey*, 41(3):1–58, 2009.

[2] E. Eskin, W. Lee, and S. J. Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *DISCEX '01: Proceedings of II DARPA Information Survivabilty Conference and Exposition*, 2001.

[3] S. Forrest. Computer immune systems- datasets and software. In *http://www.cs.unm.edu/ immsec/systemcalls.htm*, 2006.

[4] S. Forrest, S. Hofmeyr, and A. Somayaji. The evolution of system-call monitoring. In *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 418–430. IEEE Computer Society, 2008.

[5] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 120. IEEE Computer Society, 1996.

[6] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[7] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[8] N. Hubballi, S. Biswas, and S. Nandi. Fuzzy mega cluster based anomaly network intrusion detection. In *N2S 09': Proceedings of 1st International Conference on Network and Service Security*, pages 1–5. IEEE, 2009.

[9] N. Hubballi, S. Biswas, and S. Nandi. Layered higher order n-grams for hardening payload based anomaly intrusion detection. In *ARES 10': Proceedings of 5th International Conference on Avalaibility, Reliability and Security*, pages 321–326. IEEE, 2010.

[10] A. Lazarevic, A. Ozgur, L. Ertoz, J. Srivastava, and V. Kumar. A comparative study of anomaly detection scehemes in network intrusion detection. In *SIAM 03': Proceedings of 3rd SIAM International Conference on Data Mining*, pages 1–14, 2003.

[11] C. Marceau. Characterizing the behavior of a program using multiple-length n-grams. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 101–110. ACM, 2000.

[12] C. C. Michael and A. Ghosh. Simple, state-based approaches to program-based anomaly detection. *ACM Transactions on Information System Security*, 5(3):203–237, 2002.

[13] N. Nguyen, P. Reiher, and G. H. kuenning. Detecting insider threats by monitoring system call activity. In *WIA '03: Proceedings of 1st IEEE Information Assurance Workshop*, pages 18–20, 2003.

[14] A. Patcha and J.M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

[15] A. Somayaji and S. Forrest. Automated response using system-call delays. In *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, pages 14–14. USENIX Association, 2000.

[16] H.A. Sturges. The choice of a class interval. In *The American Statistical Association*, pages 65–66, North Washington, USA, 1926. American Statistical Association.

[17] P.N. Tan, M. Steibach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 1st edition, 2006.

[18] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *S&P '99: Proceedings of 19th IEEE Sympoisum of 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(2,3):159–181, 2000.

IX. APPENDIX A

K-means Clustering Algorithm

k-means clustering is an unsupervised learning algorithm to group feature vectors based on attributes/features into K number of group. K is a positive integer number fixed priori. The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

$$J = \sum_{j=1}^{K} \sum_{i=1}^{nopts} \|p_i^j - c_j\|^2 \tag{3}$$

The steps involved in the algorithm are

- Initially K distinct centroids are chosen randomly.
- Feature vectors are assigned to the nearest centroid by calculating the euclidian distance between the feature vector and the corresponding centroid.
- Centroids are updated by taking the mean of all the feature vectors assigned to clusters.
- Steps 2-3 are repeated until the algorithm converges i.e there is no change in the centroids generated.

The algorithm always terminates but does not guarantee an optimal solution due to local minima. The algorithm is also sensitive to initial selection of centroids.