Pairgram: Modeling Frequency Information of Lookahead Pairs for System Call based Anomaly Detection

Neminath Hubballi
Infosys Labs, Bengaluru
neminath_hubballi@infosys.com

***Abstract*—System call sequence based anomaly detection is one of the widely studied model of anomaly detection. There are two ways to model the system call sequences, one as full sequences and the other as lookahead pairs. Recently it has been shown that lookahead pairs perform better than full sequences. In this paper we propose an impurity tolerant model of anomaly detection using system calls called as $Pairgram$. $Pairgram$ exploits the frequency information of lookahed pairs and build a model of normal behavior. As it is generally assumed that there is a skewed distribution of normal and abnormal sequences, more frequently occurring system call sequences are considered as normal and other way for less frequent sequences. A series of experiments on the University of New Mexico system call dataset demonstrated the effectiveness of $Pairgram$ on impure dataset. Further the model is highly space efficient i.e., it has a constant space complexity of square of alphabet size of the program sequence.**

***Index Terms*—Intrusion detection system, Lookahead pairs, Program based anomaly detection, Impurity tolerant models**

I. Introduction

Intrusion Detection Systems (IDSs) are essential components of modern day network and system security infrastructure. Typically IDS can be of two types as signature based IDS and anomaly based IDS. Signature based IDS functions by maintaining a database of intrusion patterns and comparing the network/system activity against this database [33]. A signature based IDS can only detect known attacks. The second kind of system namely anomaly based IDS model the behavior which is considered as normal and any activity which falls outside the perimeter of normal boundary is detected as intrusion. Normally a profile of normal activity is created initially and this phase is sometimes referred as training phase. Later the system is used to detect intrusions. The advantage of anomaly detection systems is that they can detect both known and new attacks. It is this ability of detecting previously unknown attacks by anomaly detection systems which lead to considerable research interest [2], [31], [22], [1], [17]. To model the normal behavior either the data from a host or from a network can be used. Depending upon where the IDS is located and whether a host level data is analyzed or network level data is analyzed IDS can be called either as host based or network based. A host level IDS can typically analyze log files like syslog in unix and windows log, monitor the file integrity, detect abnormal system call sequences etc. While at the network level it can be packet header or payload based anomaly detection. Since this paper is mainly concerned about system call based modeling subsequent discussion is about system call modeling.

978-1-4673-0298-2/12/$31.00 © 2012 IEEE

In order to get the service of operating system, user or application programs use system calls. Every user program when used under normal operating conditions (intended behaviour) has a definitive sequence of system calls. If an intruder wants to misuse this and divert the execution elsewhere this well behaved sequence is diverted. This is due to the fact that program takes code paths which are not seen before and results in generating system call sequences that are different from the sequences generated under normal conditions. Whole of system call based anomaly detection is based on the premise of identifying these different sequences. The idea of modeling system calls to detect abnormal executions of the programs was studied by Forrest et al. [11], [10]. 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 barring other information like arguments and return values of the system calls.

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 observed that, modeling system calls is effective in detecting intrusions. In order to make any damage to the target system programs in the target system must be misused and if that is detected at-least in theory all possible intrusions can be detected. However accurate detection of these attacks depends on how accurately the normal activity is represented and modeled.

Another important observation w.r.t system call modeling is that, sequences are highly localized i.e., the set of sequences

generated depends on the configuration and environment of the target system where the program is running. It is observed that, two copies of the same program running on same OS versions on different machines exhibit different set of sequences [9]. This necessitates in target specific learning and model building of normal behavior. There are two ways how model can be built. One is off-line with collected system call traces and the other is 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 these limitations are

- Incomplete learning generates false positives: This indicates that before using the IDS for detecting intrusions every legitimate execution trace must be included in the training set.
- If learning is done in a vulnerable machine abnormal behavior can contaminate the normal profile and results in generation of false negatives.
- If the analysis is computation intensive then it can not be deployed in production systems.

By incorporating more training examples the first issue can be addressed however the second issue of contamination is not trivial to deal with. In addition the lighter the model is the better it is. A complete listing of various limitations of current system call based analysis techniques can be found in [6]. In summary the system call based model need to be stable, concise and consistent. In this context we answer two useful questions in this paper 1) can we build models which can tolerate accidental contamination of training dataset ? 2) can the model be very efficient and simpler?. In this paper we report results to answer these two questions. Our specific contributions are the following.

- We propose a constant space complexity model for system call based anomaly detection.
- We show that lookahead pairs are good discriminators of normal and abnormal program executions when modeled along with their occurrence frequency.
- Proposed model $Pairgram$ is tolerant to some level of contamination in the training dataset.

Rest of the paper is organized as follows. In section II we review the seminal work of Forrest et al. and other important follow up work in system call modeling for intrusion detection. In section III we describe the working of $Pairgram$. We explore the asymptotic time complexity of both training and testing phases in section IV. We provide the experimental results in section V. Finally the paper is concluded in section VI.

II. Related Work

Literature related to intrusion detection on system call monitoring can be grouped into 3 categories as described in the 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. [10] defined the legitimate behavior of a program in terms of short sequence of system calls. A database of such short sequences is created by sliding a window of length w over the training traces. An intrusion is detected if there are sufficiently new (which are not seen before) subsequences of length w in the test sequence.

Another method of this category, STIDE [16] also generates short subsequence of system calls with a window of length w and build a database of normal sequences. For every test system call sequence a rate $S^{anomaly}$ is assigned which decides how anomalous this given sequence is. It uses inverse hamming distance normalized to the window length as the anomaly score.

An intrusion response method is proposed in [34]. The idea here is to delay the execution of system calls as they drift from the defined normal profile. Key elements of this technique is a distance function to measure the drift from normal profile and exponential delay function to determine the delay for a particular system call. Authors claim that, this delay frustrates intruder and hence prevents compromising the machine.

A variable length $window\ size$ model which can best predict the next symbol in the sequence have been explored in [24], [40]. In [19] Inoue et al. compared performance of fixed width short sequences and lookahead pairs and concluded that contrary to the earlier belief lookahead pairs perform better than fixed sequences.

In our previous work [18] we showed that use of frequency information in short sequences is effective even if short $window\ size$s are used. Further it is showed that, use of frequency information will lead to impurity tolerant models. At least 1 prior work [41] has been reported which model the frequency information of system calls for intrusion detection. In a similar work [30] Park et al. report the noise tolerant intrusion detection.

B. State based Models

A Finite State Machine (FSM) modeling is proposed in [25]. FSM is constructed using the system calls of a program. FSM captures 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. In an alternative predictive model (discussed in the same paper) probability distribution of training sequences is estimated and deviations from this estimation are detected as intrusions. Wee et al in [39] automated the FSM generation using system calls.

In [21] Kosoresow et al. built a deterministic finite automata by using macros of observed system call sequences. These macros are based on the regular and repeated behavior of system call sequences. Kosoresow et al. made another important observation that, mismatches in the testing trace are highly cohesive i.e., mismatches occur within a small window and continuously. This observation of highly localized behavior of

intrusions w.r.t the normal profile helps in detecting intrusions effectively. Other notable work of this category is [5].

In [7] Feng et al used the rich call stack information in addition to the system call FSA. This model augments the FSA with return address of stack hence every transition now has list of all possible return addresses. In a similar spirit Goa et al [13] proposed execution graphs where return address pointer is stored along with the system call and when the program is running this information is used to construct a control graph pertaining to that program. Michael et al. in [26] use an extended FSM to calculate the conditional probabilities of a symbol given k of its previous symbols.

C. Complementary Models

Mutz et al. [28], [27] and Tandon et al. in [35] explored the idea of system call argument analysis for abnormal program behavior. Wagner et al. [37] showed possible mimicry attacks against system call based modeling, however use of context information in the analysis has shown to be effective in defeating such a possibilty. Another class of techniques [36], [42], [14] build models directly from program source code and are called static analysis techniques. Liu et al. in [23] proposed to combine the static analysis techniques of [36], [42], [14] and dynamic learning of Forest et al. [10] in order to get the best of two. By augmenting call stack information with statically analyzed system call streams the precision of system can be improved. In [12] Goa et al. showed that the behavior distance measure between two identical processes in execution can be used to detect anomalies in one of them. This behavior distance is measured with system calls using Hidder Markov Models. In [15] two models one with system calls and other the OS state is used to detect mimicry attacks. Given the state of OS when abuse takes place and the corresponding system calls which lead the OS to this state is used to detect such evasion attacks. Jones et al. [20] showed that language library calls too exhibit similar periodic behaviour of system calls and demonstrated that they too are potential models for anomaly detection. Nguyen et al. in [29] establishes a relationship between user profile and various processes and programs. It is noticed that some of the system users have remarkable consistencyn in accessing the files and other programs that they run. In [32] Provos introduced a dynamic policy enforcement scheme which avoid the need to execute the processes in previledged mode. This scheme allows a program to execute certain system calls (which requires special previledges) as and when needed based on the enforced policy. Chandola et al. [4] apply a formal framework to analyze system call data known as reference based analysis [3].

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 [9].

III. $Pairgram$: Modeling Frequency of Lookahead Pairs

In this section we describe the proposed scheme $Pairgram$ which models the frequency of occurrence of lookahead pairs for detecting abnormal program behavior. Unlike many previous methods based on system call modeling, $Pairgram$ effectively captures the frequency information for modeling.

$Pairgram$ in the training phase builds a model of normal program behavior and in testing phase it is evaluated against different sequences generated by the program. In the next 3 subsections we elaborate the concept of lookahead pairs, training and testing phases respectively.

A. Lookahead pairs

As the modeling is based on the lookahead pairs, we first define the notations and subsequently give an example that help understand the concept of lookahead pairs.

Notations used to define lookahead pairs are the following.

$$\begin{aligned} A &= \text{alphabet of system calls for a program} \\ size &= \text{cardinality of set } A \\ S &= \text{training traces } S_0, S_1, \cdots, S_{tracecount} \\ S_k &= t_1, t_2, \cdots, t_{tracelength} \text{ is the } k^{th} \text{ trace} \\ tracecount &= \text{number of traces available for training} \\ tracelength &= \text{length of the program trace} \\ W &= window\ size \text{ chosen} \end{aligned}$$

Lookahead pairs are formulated as bellow (adopted from [19]).

For every sequence S_k of length $tracelength$

$$\begin{aligned} < s_i, s_j >_l : \quad & s_i, s_j \in A, 2 \leq l \leq W, \\ \exists p : \quad & 1 \leq p \leq tracelength - l + 1, \\ s_i \quad &= t_p, \\ s_j \quad &= t_{p+l-1} \end{aligned}$$

Given a sequence of system calls and a window of particular size, the window is slid over the sequence and every system call within the window (except the starting system call) is paired with starting system call. For example let the sequence of system calls be $1, 2, 3, 4, 5, 6, 7, 8$ [1]. For a $window\ size$ of 4 the set of pairs generation proceeds as follows.

Starting at first system call 1 the window now has the following 4 elements $1, 2, 3, 4$. Now the pairing procedure pairs 1 with 2 and then with 3 and 4 respectively this generates pairs $< 1, 2 >$, $< 1, 3 >$, $< 1, 4 >$. Now the window is slid to next system call i.e., to 2 and pairing operation is repeated and following 3 pairs are generated $< 2, 3 >, < 2, 4 >, < 2, 5 >$. Following this approach it can be easily noticed that the set of pairs generated is $< 1, 2 >, < 1, 3 >, < 1, 4 >, < 2, 3 >, < 2, 4 >, < 2, 5 >, < 3, 4 >, < 3, 5 >, < 3, 6 >, < 4, 5 >, < 4, 6 >, < 4, 7 >, < 5, 6 >, < 5, 7 >, < 5, 8 >$.

B. Training phase

In the training phase a set of training sequences are given as input and a model of normal behavior is generated. The key idea of $Pairgram$ is based on following three premises

1) Use the frequency information of lookahead pairs: Combining the observation in [19] that lookahead pairs

[1] We assume system calls are mapped to numerical numbers

perform better than full sequences and our previous work [18] with full size sequences which shows use of frequency information of short sequences for effective abnormal program behavior.

2) The model need to be as simple as possible for evaluation.
3) Very efficient in terms of space complexity.

In order to address all these issues we use a matrix as a model and we call it as $Frequency\text{-}Matrix$ $\mathcal{M}$. $\mathcal{M}$ is of constant size $size$ X $size$ i.e., it has exactly those many rows and columns as the alphabet size of the program is.

The first step in model building is to read the input sequences one by one and incrementally update the frequency information of lookahead pairs in the matrix $\mathcal{M}$. Each entry $\mathcal{M}[m][n]$ in $\mathcal{M}$ represents the frequency information of lookahead pair starting with system call whose index is m and its pairing with system call indexed by n in the entire trainig dataset generated by a sliding window of size W. Each entry value is the sum of occurrences of system call pairs in the dataset. One way to calculate the frequencies is not to discriminate the relative distance of the system calls within the window when updating the frequency. This technique counts a frequency of 1 to every pair irrespective of whether they are consecutive or separated by certain distance within the window. In the previous example with a window of size 4 starting at system call 1, the pairs $< 1,2 >$, $< 1,3 >$, $< 1,4 >$ adds a frequency value of 1. The other way is to discriminate between the relative distance of two system calls within a pair while updating the frequency. This leads to a frequency updation relative to the distance the previous system call has with the second system call in the pair. In the above example sequence the first system call 1 has a distance of 1 with second system call 2 and a distance of 2 with third system call 3 and so on. The idea is to weight this distance measure while calculating the frequency information. In $Pairgram$ the frequency information is decreased inversely w.r.t the distance of the next system call. A system call distance 1 is given a frequency weight of 1 and a distance of 2 is given frequency weight of 1/2 and distance of 3 is given weight of 1/3 and so on.

The algorithm for training i.e., model building is shown in Algorithm-1. Given a set of sequences S, number of sequences $tracecount$ and a $window\ size$ of W the algorithm builds the $Frequency\text{-}Matrix$ $\mathcal{M}$ incrementally. The algorithm assumes that, each system call is mapped to a natural number known as index and this index is incrementally assigned starting from 0 as and when a new system call is seen in the sequence. In the algorithm the function $index()$ returns the numerical index of the system call that is passed as argument.

Algorithm 1: **Training Algorithm**

Input : $tracecount$ - number of system call traces
Input : $S_1, S_2, \cdots, S_{tracecount}$ traces for training.
Input : W - $window\ size$
Output : $Frequency\text{-}Matrix$ $\mathcal{M}$ of $sizeXsize$.

```
Initialze M to all 0
for I = 1 to tracecount do
    tracelength ← Length of trace S_I
    for J = 1 to tracelength-W+1 do
        ind1 ← index(S_I(J))
        for K = J + 1 to K < (J + W) do
            ind2 ← index(S_I(K))
            M[ind1][ind2] = M[ind1][ind2] + 1/(K−J)
        end for
    end for
end for
```

The working of Algorithm-1 and $Frequency\ Matrix$ updation can be understood with a simple example. Let the sequence of system calls be $2, 3, 2, 4, 2, 3, 4$. Alphabet size of this sequence is 3. Hence the Algorithm-1 creates a blank matrix $\mathcal{M}$ of size 3 X 3 and initialize it to 0. Let the indices of system calls 2, 3 and 4 be 0,1 and 2 respectively. With a $window\ size$ of 4 and begining at first system call the set of 4 system calls within the window are $2, 3, 2, 4$. In this window the lookahead pairs generated are $< 2,3 >$, $< 2,2 >$ and $< 2,4 >$. Now the relative weight of frequencies corresponding to these pairs are 1, 1/2 and 1/3 respectively. Thus the matrix $\mathcal{M}$ is updated with these values as follows. As index of system call 2 is 0 and that of 3 is 1, in the 0^{th} row and 1^{st} column of matrix, a frequency value 1 is added. Similarly an increase of frequency of 1/2 is made in 0^{th} row and 0^{th} column of $\mathcal{M}$ w.r.t the pair $< 2,2 >$. In a similar way 0^{th} row and 2^{nd} column is updated with frequency of 1/3 w.r.t the pair $< 2,4 >$. Now the $Frequency\ Matrix$ $\mathcal{M}$ looks as shown in matrix-1.

The $Frequency\ Matrix$ of size 3 X 3

$$\begin{pmatrix} 0.5 & 1 & 0.33 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{1}$$

In the next step the window is slid over to system call 3 and pairs are generated and the procedure is repeated w.r.t row 1 and respective columns. Proceeding in this way it is possible to show that the final $Frequency\ Matrix$ $\mathcal{M}$ looks as shown in matrix-2 bellow.

The $Frequency\ Matrix$ of size 3 X 3

$$\begin{pmatrix} 1 & 1.33 & 1.33 \\ 1.33 & 0 & 0.5 \\ 1 & 0.5 & 0.33 \end{pmatrix} \tag{2}$$

The important consequence of this matrix is, it generalizes the frequency information over a window length i.e., each entry in the matrix indicates the likelihood of system calls in the vicinity rather than the exact ordering of system calls. For example in the above $Frequency\ Matrix$-2, the entry $\mathcal{M}[1][0]$ is 1.33, this indicates that there is a high chance of system call 3 followed by 2 in a window. We can correlate

this information to the above example. In the above sequence starting with second system call 3 the window includes following 4 system calls $3, 2, 4, 2$. There is a system call 2 immediately following 3 and another 2 at a distance of 2. This entry in the matrix is indicating the overall chance of 3 followed by 2. On the other hand $\mathcal{M}[1][1]$ is 0 this indicates that, there was no appearance of system call 3 followed by another system call 3 within the specified $window\ size$ in the entire training sequence. Thus whenever such a pair of system calls (lookahead pairs) appear in the testing sequence they can be regarded as potential anomalies.

C. Testing phase

In the testing phase all the sequences given for evaluation are tested against the built model (which in our case is the $Frequency\text{-}Matrix\ \mathcal{M}$) to decide whether the execution sequence belong to some legitimate program behavior or it is suspicious. In order to test the sequence for possible anomalies, lookahead pairs are generated in the same way as they are generated for the training case and using the same $windo\ size$ used for training. Starting with the first system call the window is slid over the sequence and all the system calls within the window are paired with first system call of the window and corresponding frequencies are collected from the $Frequency\text{-}Matrix\ \mathcal{M}$. Given the observation that mismatches in the abnormal sequences occur locally another short window known as $scope\ window$ is used to decide whether sequence is normal or abnormal. If consecutive lookahead pairs in a sequence have lower frequency values (from the $\mathcal{M}$) than a pre-set threshold τ, the whole sequence is treated as anomalous. The steps involved in the testing phase are illustrated in Algorithm-2.

Another simple example helps in understanding the testing procedure. Consider a testing sequence $2, 3, 3, 3, 3$. With a $window\ size$ of 4, Algorithm-2 generates lookahead pairs as follows. Starting with the first system call 2 the window of $2, 3, 3, 3$ generates lookahead pairs $< 2, 3 >$, $< 2, 3 >$,$< 2, 3 >$ and all of these pairs take a frequency value of 1.33 from $\mathcal{M}$ (0^{th} row and 1^{st} column). Now the window is slid to second system call i.e., to 3 and this window involves the system calls $3, 3, 3, 3$. The set of lookahead pairs generated are $< 3, 3 >$, $< 3, 3 >$,$< 3, 3 >$. We can notice that, in the entire training set we did not have two system calls of 3 paired together. Hence all of these pairs recieve a frequency value of 0 from $\mathcal{M}$ (1^{st} row and 1^{st} column). These pairs generate a consecutive frequencies of 0's which can be treated as anomalous there by declaring the sequence as suspicious. The algorithm stops once the sequence is declared as anomalous.

Algorithm 2: **Testing Algorithm**

Input: $\mathcal{M}$ - $Frequency\text{-}Matrix$
Input: SP-testing trace
Input: SW_{size} - $scope\ window$ size
Input: W - $window\ size$
Input: τ- threshold
Output: Status for SP

1: $tracelength \leftarrow$ Length of trace SP
2: Initialze SW to all 0
3: **for** $J = 1$ to $tracelength$-W+1 **do**
4: $\quad ind1 \leftarrow index(SP(J))$
5: $\quad$ **for** $K = J + 1$ to $(J + W)$ **do**
6: $\quad\quad ind2 \leftarrow index(SP(K))$
7: $\quad\quad$ **for** $m = 0$ to SW_{size}-1 **do**
8: $\quad\quad\quad SW[m] \leftarrow SW[m + 1]$
9: $\quad\quad$ **end for**
10: $\quad\quad SW[SW_{size}] = \mathcal{M}[ind1][ind2]$
11: $\quad\quad$ **if** $SW[0]$ to $SW[SW_{size}] \geq \tau$ **then**
12: $\quad\quad\quad$ Sequence SP is abnormal
13: $\quad\quad\quad$ TERMINATE
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end for**
17: Sequence SP is normal

IV. Complexity Analysis

In this section we describe the time and space complexity of both training and testing algorithms. The subsection IV-A describes the time complexity of training and testing phases; similarly the section-IV-B describes the space complexity of $Pairgram$.

A. Time Complexity

Training algorithm

It can be noticed from the Algorithm -1 that there are 3 loops in the entire algorithm (line no 2, 4 and 6). The first one loops from 0 to $tracecount$ i.e., number of traces, second one loops from 0 to $tracelength$ i.e., number of system calls in a trace and third one from 0 to $window\ size$ times. Given that window is much small (typically of order 3-10) compared to the $tracelength$ and $tracecount$ third loop (line 6) can be treated as of constant complexity. It can be easily noticed that all of the remaining operations in the algorithm are of constant time operations. Thus the overall time complexity of Algorithm-1 can be written as $O(tracecount * tracelength)$ which is exactly the time required to read the entire training dataset once. Since irrespective of the approach taken entire training dataset has to be read at least once (incurring a time complexity of $O(tracecount * tracelength)$). Thus our algorithm has the best possible time complexity.

Testing algorithm

From Algorithm-2 we can notice that there are again 3 loops in the algorithm (line 3, 5 and 7). Of the 3 loops the loop at line number 3 iterates approximately from 0 to tracelength and it is easy to notice that second and third loop are of constant complexity as $window\ size$ and $scope\ window$ are much smaller values compared to the length of trace. In a similar way all remaining operations in the algorithm are of

constant time complexity. Thus the over all time complexity of testing algorithm can be written as $O(tracelength)$ which is exactly the time required to read the test sequence once.

B. Space Complexity

It can be easily seen that the only component of training algorithm is the $Frequency\text{-}Matrix\ \mathcal{M}$ whose size is fixed as $sizeXsize$. Thus the model has constant overall space complexity.

V. Experimental Results

In this section we report the experimental results on the University of New Mexico (UNM) dataset available at [8]. There are several system call program traces in this dataset. These program traces were collected during the live execution of privileged programs in production systems. Some of them were collected at more than one places. Each trace consists of complete listing of system calls made during execution of the program from start to end. Specifically following 9 programs traces are available in the dataset - lpr, $named$, $xlock$, $login$, ps, $inetd$, $stide$, ftp and $sendmail$.

There are traces which have both normal and abnormal traces and others which have only normal traces. If a program has both normal and abnormal traces it can be used to evaluate both $Detection\ Rate$ and $Accuracy$, on the other hand if it has only normal traces it can be used to measure the $Accuracy$ a.k.a $False\ Positive\ Rate$. Out of available 9 program traces 8 programs are having both normal and intrusion traces and for another program only normal traces are available. The characteristics of these traces is shown in Table II.

TABLE II
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
ftp	2	1
sendmail	71760	0

Out of 9 programs mentioned in Table II 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 sub-process 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 help 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 program ftp has 2 files of normal traces and another file of intrusion. The last program $sendmail$ has 71760 normal traces and there are no intrusion traces. All traces except UNM live lpr shown in Table II were used in the experimentation of [38]. 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 [16].

A. Implementation and Evaluation

We are interested in studying the performance of the proposed system $Pairgram$ in two parameters namely the $Detection\ Rate$ and $False\ Positive\ Rate$ it shows. These two are sensitive to the threshold selection, for example what frequency is considered as low and how many consecutive low frequencies are considered for declaring a particular test sequence as intrusion. This sensitivity is not specific to $Pairgram$ instead it shares this with almost all of the techniques proposed in the literature. Thus in the following experiments, we report $False\ Positive\ Rate$ of $Pairgram$ keeping $Detection\ Rate$ constant at 100%. In order to do this we varied the parameter namely threshold for low frequency while keeping the $scope\ window$ size i.e., consecutive low frequencies values constant at 3. How to optimally choose a threshold is beyond the scope of present discussion and we will not deal with it here. Here we report results on those traces which have both normal and abnormal traces for evaluation. To avoid any undue biases in the results obtained all of our experiments are validated on a 10 fold cross validation.

We did some modifications to the UNM dataset in order to help perform cross validation. There are some datasets on which we do not report the results and in other cases we divide the long single sequence into many small sequences so that cross validation can be done. There are single normal files of $named$ and $xlock$ programs, and we divided them into 462 and 220 smaller traces respectively. There are two traces of normal files for program ftp which we divided into 360 smaller traces. This division is to simply 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 there are no demarcation to separate them. Second some of these programs have very few normal traces to learn the frequency information of lookahead pairs which

TABLE III
FREQUENCY SCORES

19077.933333, 4352784.166550, 2864076.305518, 3844796.099883, 35301.533333, 1080282.605556, 1821625.816628, 1294001.633295, 34501.200000, 34501.200000, 22151.833333, 1294436.805517, 11780.333333, 540382.627778, 15444.300000, **220.466667, 64.033333, 84.700000**, 100378.833333, 134396.433333, 135309.266667, 65893.400000, 100378.833333, 134396.433333, 135309.266667,

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 a 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 is divided we may miss its detection.

We begin by justifying our argument of consecutive low frequency lookahead pair values are generated in case of abnormal sequences. Here we show the frequency scores generated in an example sequence. Table III shows a portion of the frequency values generated by lookahead pairs. This is a partial view of one of the MIT lpr abnormal sequence with a window of size 6 against the $Frequency\text{-}Matrix$ generated by MIT lpr normal sequences. From the table we can notice that there is a portion of frequency values with very low frequency in comparison with the rest of the portion. This low frequency portion of the sequence is shown in bold letters in the table. This example justifies our approach of monitoring consecutive low frequency values in the test sequence. In the subsequent paragraphs we report the $False\ Positive\ Rate$ of $Pairgram$ with a $window\ size$ of 3. First set of results are on pure dataset i.e., when there are no abnormal sequences added to the training dataset and using this as a baseline we subsequently show the tolerance of $Pairgram$ to various levels of impurities.

Table IV shows the $False\ Positive\ Rate$ shown by $Pairgram$ (column 4 of Table IV) for 100% $Detection\ Rate$[2] on a pure dataset i.e., when there are no abnormal sequences added to the training dataset. It can be noticed that $False\ Positive\ Rate$ is very low or 0 in all the cases. This confirms the claim that lookahead pairs are good discriminators when modeled with their occurrence frequency. We also compared our results to one of our previous work $Sequencegram$ which modeled frequency information of full sequences with the $window\ size$ of 3 and we can notice that $Pairgram$ performs better than $Sequencegram$ in terms of $False\ Positive\ Rate$.

To study the performance of $Pairgram$ to impure/contaminated dataset we conducted a series of experiments. In each experiment we deliberately added some of the abnormal traces into the training dataset and then use the same evaluation method as in the previous case i.e., $False\ Positive\ Rate$ when the $Detection\ Rate$ is 100%. In addition the threshold selection and remaining parameters are same as that of previous (pure dataset) case. In each case we report the number of abnormal traces added to the training dataset and report the achieved $False\ Positive\ Rate$. Table V shows the $False\ Positive\ Rate$ for various levels of impurities on MIT lpr dataset. It is evident from the table that upto 100 abnormal traces addition into the training dataset the $False\ Positive\ Rate$ remains the same as that of pure dataset. This confirms the absolute tolerance of $Pairgram$ upto this level and beyond that the $False\ Positive\ Rate$ starts increasing. It can be noticed from the table that up to 300 abnormal traces addition the achieved $False\ Positive\ Rate$ is still under practically acceptable limits and beyond that it becomes high.

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	01.47%
2703	50	1001	01.47%
2703	100	1001	01.47%
2703	200	1001	01.99%
2703	300	1001	01.99%
2703	500	1001	64.74%
2703	1001	1001	99.63%

In a similar way Table VI shows the performance of UNM lpr dataset on various levels of impurities. Table VII, Table VIII and Table IX show the $False\ Positive\ Rate$ variation for $named$, $xlock$ and ftp program traces respectively. It can be inferred from these tables that, trends seen with MIT lpr experiments were seen in other programs too. Programs $xlock$, $named$ and ftp have only one or two abnormal traces (which are long sequences) and their addition to the training dataset did not affect the performance. Since there are no additional abnormal traces available for experimentation we could not completely assess the behavior of these programs. However this set of experiments do indicate that, built program profile by $Piargram$ can tolerate some level of contamination.

B. Performance

Here we report the practical time taken by the algorithm in the training and testing phases respectively. Our current non optimized version of implementation of $Pairgram$ in C language takes total of 6.45 seconds to train 2703 sequences of MIT lpr program and test 1001 test sequences. All the above

[2]In addition it is worth noting that there is a subtle difference in the way we calculate the $Detection\ Rate$ and $False\ Positive\ Rates$ compared to the previous methods. We report the $Detection\ Rate$ and $False\ Positive\ Rates$ calculated on per sequence basis unlike the literature where per system call based evaluation is done. It makes more sense for example if the method says out of every 100 program executions there is a chance of one legitimate execution of program being reported as intrusion.

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*	
			Pairgram	*Sequencegram*
MIT LPR	2703	1001	**1.47%**	1.70%
UNM LPR	1232	1001	**0.00%**	0.00%
xlock real	220	2	**0.00%**	0.00%
named	462	2	**0.00%**	0.00%
ftp	362	1	**1.66%**	2.77%

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	00.00%
1232	50	1001	00.00%
1232	100	1001	01.45%
1232	200	1001	01.79%
1232	300	1001	04.64%
1232	500	1001	56.18%
1232	1001	1001	98.32%

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%

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%

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

Number of Normal Traces	Number of Intrusion Traces Added	Number of Intrusion Traces Tested	*False Positive Rate*
362	1	1	1.66%

experiments were done on an Intel (R) Core (TM) 2 Duo CPU running at 3 GHz and having 2 GB RAM, running Ubuntu 9.2 operating system. Observations made in this experiments translated to other experiments too. In addition it may be noted that current implementation is an off-line evaluation however in the real time monitoring it may vary.

C. Impact of variation of *window size*

In this section we show the effect of varying the *window size* on the performance of *Pairgram*. This is important from the stability point view of the proposed approach. Similar to the previous case we report the *False Positive Rate* generated for 100% *Detection Rate*. In this experiment we varied the *window size* between 3 to 15. Figure. 1 shows the variation of *False Positive Rate* with the variation of *window size*s for two different training sets of MIT *lpr* program. In one case we used the pure dataset and in the second case we used a training dataset with 25 abnormal sequences added to it. We notice that in the two cases there is a steady performance upto a certain *window size* (8 in case of pure and 6 in case of impure datasets) and after that for a certain*window size* s the *False Positive Rate* increases. When the *window size* increases further we see there is a decrease in the *False Positive Rate*. This is a rather surprising result. A careful examination into the case revealed that upto a certain *window size* the frequency information was able to discriminate normal and abnormal sequences and there after frequency information becomes insufficient to discriminate normal and abnormal cases due to generalisation effect of lookahead pairs and end up with making more errors. This can be avoided by providing more training data to the model. However if the *window size* is increased further, the effect of generalisation significantly increases thereby the model recognizes more and more cases as legitimate and there is a decreased *False Positive Rate*. We notice there is a stabilization of *False Positive Rate* for a *window size* of more than 12. The conclusion that can be inferred from this experiment is higher order *window size*s are good for a low *False Positive Rate* however they take more time for evaluation.

D. Effect of variation of scope window

In this subsection we explore behavior of *Pairgram* w.r.t another important parameter *scope window* i.e., the effect of number of consecutive low frequencies considered to decide a sequence as abnormal. In order to study this effect we experimented with MIT *lpr* dataset by keeping *window size* at 3 and varying the *scope window* between 2 to 7. Figure 2 shows the variation of *False Positive Rate* over the *scope window* value for a 100% *Detection Rate*. We can notice that a *scope window* from 2 to 5 decreased the *False Positive Rate* consistently at a low rate. There is a

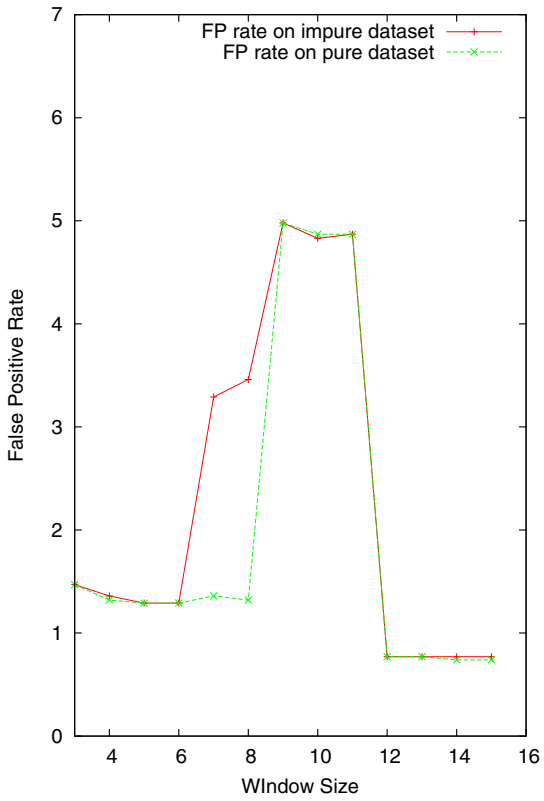

Fig. 1. Effect of *window size* variation

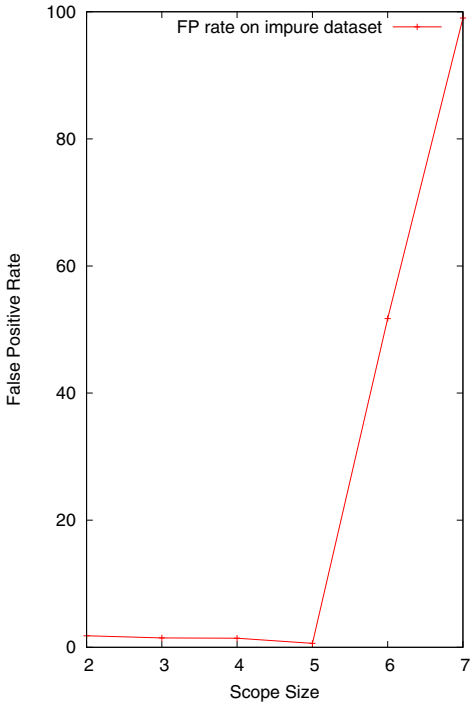

Fig. 2. Effect of scope size variation

sudden surge in the *False Positive Rate* at *scope window* values of 6 and 7 and the model become useless with very high *False Positive Rate*. This experiment reveals two interesting things one is the optimal *scope window* value of 5 (where the *False Positive Rate* is low) and the other is stability of the method within the range 2-5 (with more or less same *False Positive Rates*). The high *False Positive Rate* at *scope window* of 6 and 7 can be explained as follows. Since we report the results for 100% *Detection Rate*[3], in order to detect all abnormal traces the frequency threshold τ has to increased to a high value. When τ is increased to a high value many of the normal traces also pass this limit of high τ value and will be detected as abnormal sequences thereby increasing *False Positive Rate*.

VI. Conclusion

In this paper we described a system call based abnormal program behavior detection technique. We proposed an experimental system called *Pairgram* which works by modeling the frequency information of the lookahead pairs. *Pairgram* is highly efficient in terms of time and space complexity. It has a constant space complexity which is equal to the square of the alphabet size of the program sequence. Due to this reduced space complexity evaluation is quit simpler. Further the model is tolerant to some level of contamination in the training dataset. We conducted a series of experiments and validated the claim by a 10 fold cross validation scheme.

[3]Unlike the ROC curves normally used in literature

References

[1] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences. *IEEE Transactions on Knowledge and Data Engineering (In press)*.

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

[3] V. Chandola, S. Boriah, and V. Kumar. A framework for exploring categorical data. In *ICDM '09: Proceedings of the 9th SIAM International Conference on Data Mining*, pages 187–198. ACM, 2009.

[4] V. Chandola, S. Boriah, and V. Kumar. A reference based analysis framework for analyzing system call traces. In *CSIIRW '10: Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–3. ACM, 2010.

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

[6] H.H. Feng, J. T. Giffin, and Y. Huang. Formalizing sensitivity in static analysis for intrusion detection. In *S&P '04: Proceedings of the 24th IEEE Symposium on Security and Privacy*, pages 1–15. IEEE Computer Society, 2004.

[7] H.H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *S&P '03: Proceedings of the 23rd IEEE Symposium on Security and Privacy*, pages 62–. IEEE Computer Society, 2003.

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

[9] S. Forrest, S. Hofmeyr, and A. Somayaji. The evolution of system-call monitoring. In *ACSAC '08: Proceedings of the 24th Annual Computer Security Applications Conference*, pages 418–430. IEEE Computer Society, 2008.

[10] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In *S&P '96: Proceedings of the 17th IEEE Symposium on Security and Privacy*, page 120. IEEE Computer Society, 1996.

[11] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *S&P '01:Proceedings of the 14th IEEE Symposium on Security and Privacy*, pages 202–212. IEEE Computer Society, 1994.

[12] D. Gao, M. K. Reiter, and D. Song. Behavioral distance measurement using hidden markov models. In *RAID '06: Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, pages 19–40. LNCS, 2006.

[13] D. Gao, M.K. Reiter, and D. Song. Gray-box extraction of execution graphs for anomaly detection. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 318–329. ACM, 2004.

[14] J.T. Giffin, S. Jha, and B.P. Miller. Detecting manipulated remote call streams. In *USENIX '02: Proceedings of the 11th USENIX Security Symposium*, pages 61–79. USENIX Association, 2002.

[15] J.T. Giffin, S. Jha, and B.P. Miller. Automated discovery of mimicry attacks. In *RAID '06: Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, pages 41–60. LNCS, 2006.

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

[17] 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.

[18] N. Hubballi, S. Biswas, and S. Nandi. Sequencegram: n-gram modeling of system calls for program based anomaly detection. In *COMSNETS '11: Proceedings of the 3rd IEEE International Conference on Communication Systems and Networks*, pages 1–10. IEEE, 2011.

[19] H. Inoue and A. Somayaji. Lookahead pairs and full sequences: A tale of two anomaly detection methods. In *NYS '07: Proceedings of the 2nd Annual Symposium on Information Assurance*, pages 1–11. IEEE Computer Society, 2007.

[20] A. Jones and Y. Lin. Application intrusion detection using language library calls. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Application Conference*, pages 1–6. IEEE, 2001.

[21] A. P. Kosoresow and S.A. Hofmeyr. Intrusion detection via system call traces. *IEEE Software*, 14(5):35–42, 1997.

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

[23] Z. Liu, S.M. Bridges, and R.B. Vaughn. Combining static analysis and dynamic learning to build accurate intrusion detection models. In *IWIA '05: Proceedings of the 3rd IEEE International Workshop on Information Assurance*, pages 164–177. IEEE Computer Society, 2005.

[24] C. Marceau. Characterizing the behavior of a program using multiple-length n-grams. In *NSPW '00: Proceedings of the 9th Workshop on New Security paradigms*, pages 101–110. ACM, 2000.

[25] 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.

[26] C.C. Michael and A. Ghosh. Two state-based approaches to program-based anomaly detection. In *ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference*, pages 21–30. IEEE Computer Society, 2000.

[27] D. Mutz, W. Robertson, G. Vigna, and R. Kemmerer. Exploiting execution context for the detection of anomalous system calls. In *RAID '07: Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, pages 1–20. LNCS, 2007.

[28] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel. Anomalous system call detection. *ACM Transaction on Information System Security*, 9(1):61–93, 2006.

[29] 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.

[30] Y. Park, J. Lee, and Y. Cho. Intrusion detection using noisy training data. In *ICCSA '04: Proceedings of 2nd International Conference on Computational Science and Its Applications*, pages 215–222. LNCS, 2004.

[31] 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.

[32] N. Provos. Improving host security with system call policies. In *USENIX '03: Proceedings of the 12th Conference on USENIX Security Symposium*, pages 1–15. USENIX Association, 2003.

[33] M. Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: Proceedings of the 13th USENIX Conference on System Administration*, pages 229–238. USENIX Association, 1999.

[34] 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.

[35] G. Tandon and P. Chan. On the learning of system call attributes for host based anomaly detection. *International Journal on Artificial Intelligence Tools*, 15(6):875–892, 2006.

[36] D. Wagner and D. Dean. Intrusion detection via static analysis. In *S&P '01:Proceedings of the 21st IEEE Symposium on Security and Privacy*, pages 156–176. IEEE Computer Society, 2001.

[37] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 255–264. ACM, 2002.

[38] 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. IEEE, 1999.

[39] K. Wee and B. Moon. Automatic generation of finite state automata for detecting intrusions using system call sequences. In *ACNS '03: Proceedings of the 3rd Applied Cryptography and Network Security*, pages 206–216. LNCS, 2003.

[40] 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.

[41] Y. Wu, J. Jiang, and L. Kong. Sequential frequency vector based system call anomaly detection. In *PRDC '10: Proceedings of 16th Pacific Rim International Symposium on Dependable Computing*, pages 215–222. IEEE, 2010.

[42] H. Xu, W. Du, and S.J.Chapin. Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. In *RAID '04: Proceedings of the 7th International Conference on Recent Advances in Intrusion Detection*, pages 1–19. LNCS, 2004.