

# Lookahead Pairs and Full Sequences: A Tale of Two Anomaly Detection Methods

Hajime Inoue     Anil Somayaji  
School of Computer Science  
Carleton University  
Ottawa, Canada K1S 5B6  
{hinoue, soma}@ccsl.carleton.ca

**Abstract**—Sequence-based analysis has been both a widely imitated and widely criticized approach to anomaly detection. In virtually all of the follow-up work to Forrest et al. (1996), though, the distinction between the initially proposed “lookahead pairs” and the follow-on “full sequence” analysis methods has been overlooked. We have discovered that this oversight is significant: specifically, here we demonstrate that, on previously published and well-studied datasets, lookahead pairs produce significantly fewer false positives. Although lower false positive rates make lookahead pairs an attractive system call modeling technique, their usefulness may be compromised by an increased vulnerability to mimicry attacks. This threat can be mitigated through the use of larger sequences. Here we show that lookahead pairs produce relatively few false alarms even with longer sequences ( $n > 10$ ); we also demonstrate a new technique, random schema masks, which permits the use of even longer sequences. With these new results and techniques, we conclude that the lookahead pair method should be considered as one of the benchmark techniques for modeling system calls.

## I. INTRODUCTION

Program-level anomaly detection has been a topic of research in computer security for more than a decade. It holds the promise of detecting and intercepting attacks on network servers and applications in real-time, potentially preventing system damage or disclosure of confidential information—all without requiring signatures or handwritten specifications of legal behavior. The nature of anomaly detection, however, means that this promise comes with significant caveats: legitimate but unusual behavior can be flagged as anomalous (false positives), and malicious behavior can sometimes be classified as “normal” (false negatives). Developing methods that achieve the appropriate balance between low false alarm rates and high sensitivity to attacks is the key challenge for anomaly-based intrusion detection.

The earliest and most influential program-level anomaly detection strategy is based on monitoring sequences of system calls. First proposed by a group at

the University of New Mexico led by Stephanie Forrest [3], and that included Somayaji, it has been studied and criticized by many other researchers primarily on two grounds: that it suffers from a high false positive rate [11], and that it is susceptible to evasion by attackers via “mimicry attacks” [18]. It has been less widely appreciated, though, is that in the early literature on system call sequences, *two* modeling methods were proposed: the initial “lookahead pairs” [3] method and the later “full sequence” [2], [4] method. Virtually all of the literature on system call sequence-based intrusion detection uses the full sequence method; the notable exception is pH, a real-time intrusion detection and response system [13], [12]. In this paper we argue that the differences between these two modeling methods are more significant than has been previously appreciated. In particular, we have found that the lookahead pairs method has a clear advantage in false positive (false alarm) rates while still maintaining the ability to detect real intrusions.

Lookahead pairs generate fewer false positives because they generalize over previously observed sequences; this same quality also makes the lookahead pair method less sensitive to some attacks and more vulnerable to mimicry attacks than the full sequence method. We note, however, that the sensitivity of sequence-based methods can be improved through the use of longer sequences (but at the cost of more false positives). We have found that lookahead pairs perform remarkably well with longer sequences than have been typically used in the literature ( $n > 10$ ). Here we also present a new technique, random schema masks, that makes the use of even longer sequences feasible with both lookahead pairs and full sequences.

The rest of this paper proceeds as follows. In Section II, we explain the lookahead pairs and full sequence methods in detail. Section III presents a comparison of the two methods in terms of profile size and false

positive rates, while Section IV presents results on the feasibility of random sequence masks. In Section V, we give a brief overview of related work in modeling system call sequences. Section VI discusses the limitations and implications of this work. Section VII concludes the paper.

## II. MODELING SYSTEM CALLS

In this section we motivate and explain both the lookahead pairs and full sequence methods. We also discuss a few subtle differences between our implementation of these algorithms and the implementations described by Forrest et al. [3] and Hofmeyr et al. [4], respectively.

Profiles for both methods are parameterized by a single value, the **window** length, referred to as  $w$ . Profiles are generated by observing the system-call stream during execution.

Before we present a formalization of the two learning methods, we introduce them using an example, shown in Figure 1. The example system call stream (Figure 1a), consists of 11 system calls. The code it represents reads the metadata for two files, opens them, moves the file pointer in each, reads from each, and then closes them. In both the simulator and in operating systems, system calls are actually represented by integers.

We present the profile of the full sequence (FS) method (Figure 1b), first. It uses a window length of 4. In this profile, we can see that in row 1, the first system call sequence is `fstat`, with `sentinel` in the other positions. Unlike in the original Forrest et al papers [3], [4], we use sentinels to initialize the system, allowing the system to signal anomalies at the earliest possible moment, following the pH example [12]. The sequences are added to the profile as they appear in the stream, until row 9 (the last row). In row 9, the system call in current is `close`, not `read`, because `<read, lseek, open, fstat>`, already appeared in row 3. The finished profile is a set of all unique sequences of length 4.

Generalization only occurs in the full sequences method through the window length. Every legal sequence seen during anomaly detection must have been observed during training. However, because different code paths can share sequences, the representation allows for “jumps” between code paths that the program text would never allow. Such “impossible paths” of execution potentially make it much easier for an adversary to craft attacks that could evade detection by a sequence-based anomaly detector [17].

The lookahead pairs (LAP) profile for the sample data (Figure 1c) also uses a window length of 4. The difference between the FS and LAP methods is that the

system calls are indexed on the current position: all rows with the same current system call are merged. During testing, any sequence is then allowed that contains one of the valid system calls in the appropriate position in the row for the current system call.

For example, consider row 1 of the LAP profile. On the first system call of the trace, `fstat`, the pairs `<fstat, sentinel, sentinel, sentinel>` is added to the profile. At system call 7, instead of a new sequence being added, the previous entry is modified to `<fstat, {sentinel, close}, {sentinel, read}, {sentinel, read}>`.

The LAP method allows both “impossible paths” generalization as well as substitutions. To illustrate, consider the previous example. Legal system-call streams under the LAP method would include `<fstat, close, sentinel, sentinel>` and `<fstat, close, read, sentinel>`, which do not appear in the original stream.

Now we introduce a formalization of the two methods, adapted from Somayaji’s dissertation [12]. Let

$$\begin{aligned}
 C &= \text{alphabet of possible system calls and a sentinel} \\
 c &= |C| \text{ (221 in Linux 2.4, 317 in Linux 2.6)} \\
 T &= t_1, t_2, \dots, t_\tau | t_i \in C \text{ (the trace)} \\
 \tau &= \text{the length of } T \\
 w &= \text{the window size, } 1 \leq w \leq \tau \\
 P &= \text{a set of patterns associated with } T \text{ and } w \\
 &\quad \text{(the profile)}
 \end{aligned}$$

For the full sequences (FS) method, the profile  $P_{seq}$  is defined as:

$$\begin{aligned}
 P_{seq} &= \{ \langle s_i, s_{i+1}, \dots, s_j \rangle : s_i, s_{i+1}, \dots, s_j \in C, \\
 &\quad 1 \leq i, j \leq \tau, \\
 &\quad j - i + 1 = w, \\
 &\quad s_i = t_i, \\
 &\quad s_{i+1} = t_{i+1}, \\
 &\quad \dots \\
 &\quad s_j = t_j \}
 \end{aligned}$$

Alternately, for the lookahead pairs method, the profile  $P_{pair}$  is defined as:

$$\begin{aligned}
 P_{pair} &= \{ \langle s_i, s_j \rangle_l : s_i, s_j \in C, 2 \leq l \leq w \\
 &\quad \exists p : 1 \leq p \leq \tau - l + 1, \\
 &\quad t_p = s_i, \\
 &\quad t_{p+l-1} = s_j \}
 \end{aligned}$$

The difference in generalization can be quantified by comparing the total number of sequences each profile method recognizes. The number of sequences the FS method recognizes is simply the number of unique sequences it saw during training. The number of sequences ( $S$ ) the LAP recognizes must be calculated

fstat, open, lseek, read, read, close, fstat, open, lseek, read, close

(a)

position 3	position 2	position 1	current
sentinel	sentinel	sentinel	fstat
sentinel	sentinel	fstat	open
fstat	open	lseek	read
open	lseek	read	read
lseek	read	read	close
read	read	close	fstat
read	close	fstat	open
close	fstat	open	lseek
open	lseek	read	close

(b)

position 3	position 2	position 1	current
{sentinel, read}	{sentinel, read}	{sentinel, close}	fstat
{sentinel, read}	{sentinel, close}	{fstat}	open
{sentinel, close}	{fstat}	{open}	lseek
{fstat, open}	{open, lseek}	{lseek, read}	read
{lseek, open}	{read, lseek}	{read}	close

(c)

Fig. 1. An example system call stream (a), the sequence profile generated from that stream (b), and the lookahead pairs profile (c). The code represented by this system call stream does similar things with two files: It reads metadata about the file (stat), opens the file, moves the file offset pointer, reads from the file, and closes them. Both methods are shown using a window length of 4.

combinatorially:

$$S = \sum_{i=0}^c \prod_{j=1}^w \sum_{k=0}^c \mathcal{F}(i, j, k) \quad \text{where}$$

$$\mathcal{F}(i, j, k) = \begin{cases} 1 & \text{if } \langle i, k \rangle_j \in P_{pair} \\ 0 & \text{otherwise} \end{cases}$$

The total number of sequences recognized by a profile is simply the sum of the number of sequences recognized with each system call in the “current” position (characterized by the summation with the  $i$  index). The total for each system call is the product of the number of system calls that are in the profile for each position. This is calculated by adding them together in the second sum, and the multiplication is then carried out by the product indexed by  $j$ . We call  $S$  the *size* of the profile.

Generalization in LAP is dependent on the density of the profiles. A “sparse” profile, with few system call in each window position set, will recognize far fewer sequences than one with larger sets in those positions. We examine the generalization of LAP in practice in Section III.

In the experiments described in this paper, we implemented the full sequence and lookahead pairs methods as described above; however, we also added some code to handle a few special cases that are not directly addressed in the early work by the UNM group. These changes, however, are in accordance with the online implementation of lookahead pairs developed for Somayaji’s pH [13], [12].

In earlier work, sequences were not checked until the window filled up; that is, no anomalies could be registered until  $l$  system calls were invoked, allowing potentially dangerous behavior. In this research we create entries and check for anomalies after every system call. We then enter a special, non-existing sentinel system call in the empty positions for the first  $l - 1$  calls.

In addition, we treat two system calls as special: `fork` and `execve`. When a fork occurs in the system call stream, we make a copy of the system-call sequence at that point of execution and use it to initialize the sequence of the new process. This makes sense because a fork carries almost all the state of its parent process. Also, this prevents a process from using fork to avoid anomaly checks.

With the `execve` call we reinitialize the sequence window. When an `execve` is called, the operating system loads a new program binary into the currently running process, destroying most of the process's previous state. In our simulator we continue using the current profile. This differs slightly with the behavior of `pH`, which reinitialized the sequence window and switched profiles based on the actual executable being launched. Because the data files do not contain arguments, we cannot simulate this behavior.

### III. FULL SEQUENCES VERSUS LOOKAHEAD PAIRS

We developed our own anomaly detection simulators for the lookahead pair and full sequence models. The UNM group never released the original simulator used in the lookahead pairs studies. They did release the source code used in the sequence studies (`stide`), but given the changes we used in Section II, we felt it easier to write our own simulator.

We verified our own version of full sequences against `stide`. Unfortunately, `stide` 1.1, the version released by UNM in 1998 [5], no longer compiles on modern systems. We rewrote `stide` to use modern C++ template mechanisms and checked that the results of `stide` were similar to our own simulator in full sequence mode.<sup>1</sup> The results are not exact because of the different behavior after `fork` and `execve`.

Our tests are against data provided by the UNM group [1]. We report data from four different traces:

- 1) `lpr-mit`: 1942917 training calls and 811953 testing calls (about a month of data from many different machines).
- 2) `named`: 21 training days (6,256,177 calls); 12 testing days (2,974,395 calls).
- 3) `sendmail`: 29666817 training calls and 14833409 testing calls.
- 4) `xlock`: 11,065,759 training calls and 5,532,880 testing calls (approximately 2 days total).

We believe these are the most interesting of the data traces available. Most of the other traces available from UNM are either synthetic or are of less interesting programs, such as `login` or `ps`.

On each, we attempted to divide the raw traces with two thirds as training and one third as testing. On the `lpr`, `sendmail`, and `xlock` traces we did this by using system call counts. On the `named` data we divided the trace into training and testing by using the date logs. Note that this training/testing division is different than that used by Warrender et al. [20].

<sup>1</sup>`stide` 1.2 is now available from UNM [1].

Our analysis concentrates on `named`, because we have complete kernel logs from that trace. Those logs allow us to calculate false positive per day rates which are more useful in determining anomaly IDS performance. Such logs were either not available or not appropriate for calculating per-day false positive rates for the other three data sets.

Table I presents the false positive rates per system call. We used 14 different window lengths from 2 to 128. We were not able to gather the largest data sizes for full sequences because of the prohibitive runtime of our full sequence analysis program when using very long sequences.

There are two conclusions to draw from the data: 1) results vary widely depending on the trace, and 2) lookahead pairs have significantly smaller false positive rates than full sequences. Our first conclusion merely confirms the result Warrender et al. found in comparing alternative models of system call-based anomaly intrusion detection [20]. The second result is more interesting.

The lookahead pairs false positive rates are lower than the full sequence rates for each of the traces. On the `named` and `xlock` traces the differences are dramatic. Figure 2 shows the ratios for the four traces over the various window lengths. The lookahead pairs false positive rates are a minimum of ten times lower, sometimes much lower, for both `named` and `xlock`.

The ratios for `lpr` are more modest. We believe insufficient training accounts for this discrepancy. It is the smallest trace, and it is also the one whose data comes from many different machines. A greater diversity coupled with less training leads to higher false positive rates for both lookahead pairs and full sequences, leading to a smaller discrepancy in performance.

The ratios for `sendmail` fall in between the two groups. This trace has the largest number of system calls, but its behavior is atypical because of the large number of forks. Still, the full sequences method experiences more than twice as many anomalies as lookahead pairs for window lengths long enough to detect attacks.

The reason behind the difference in false positives is due to the extra generalization of the LAP method. One can compare the two methods by looking at the “size” of each profile—the number of unique sequences it recognizes as normal. Table II shows the sizes of the profiles for the LAP and FS methods for each window sizes. For windows greater than 4 the LAP method recognizes many magnitudes more sequences than the FS method.

Figure 3 plots the ratios of the LAP sizes to FS sizes for the four traces over the selected window sizes.

Window	lpr		named		sendmail		xlock	
	LAP	FS	LAP	FS	LAP	FS	LAP	FS
2	1.85E-05	0.00E+00	0.00E+00	0.00E+00	1.03E-04	0.00E+00	9.04E-08	0.00E+00
3	3.69E-05	1.85E-05	0.00E+00	0.00E+00	1.51E-04	1.03E-04	1.81E-07	9.04E-08
4	6.16E-05	4.80E-05	0.00E+00	1.34E-06	2.07E-04	2.03E-04	2.71E-07	2.17E-06
5	9.11E-05	8.01E-05	6.72E-07	4.03E-06	2.41E-04	3.24E-04	3.61E-07	4.70E-06
6	1.17E-04	1.15E-04	1.01E-06	1.88E-05	2.66E-04	4.31E-04	4.52E-07	8.59E-06
8	1.50E-04	1.47E-04	2.35E-06	3.56E-05	3.31E-04	5.39E-04	9.04E-07	1.24E-05
10	2.00E-04	2.03E-04	3.70E-06	7.36E-05	3.80E-04	7.38E-04	1.08E-06	2.03E-05
12	2.27E-04	2.92E-04	5.72E-06	1.34E-04	4.28E-04	9.79E-04	2.17E-06	3.08E-05
16	4.77E-04	4.09E-04	1.04E-05	2.14E-04	5.27E-04	1.22E-03	2.80E-06	4.15E-05
20	6.93E-04	8.72E-04	1.51E-05	5.32E-04	6.30E-04	1.83E-03	3.71E-06	6.33E-05
24	1.15E-03	1.90E-03	2.15E-05	1.21E-03	7.36E-04	2.56E-03	4.79E-06	8.25E-05
32	1.50E-03	3.28E-03	2.79E-05	2.30E-03	8.93E-04	3.46E-03	6.33E-06	1.01E-04
64	2.62E-03	NA	4.37E-05	NA	1.37E-03	NA	1.05E-05	NA
128	4.06E-03	NA	6.32E-05	NA	1.76E-03	NA	2.21E-05	NA

TABLE I

FALSE POSITIVE RATES PER SYSTEM CALL FOR THE FOUR TRACES FOR BOTH LOOKAHEAD PAIRS (LAP) AND FULL SEQUENCE (FS) ANALYSIS METHODS.

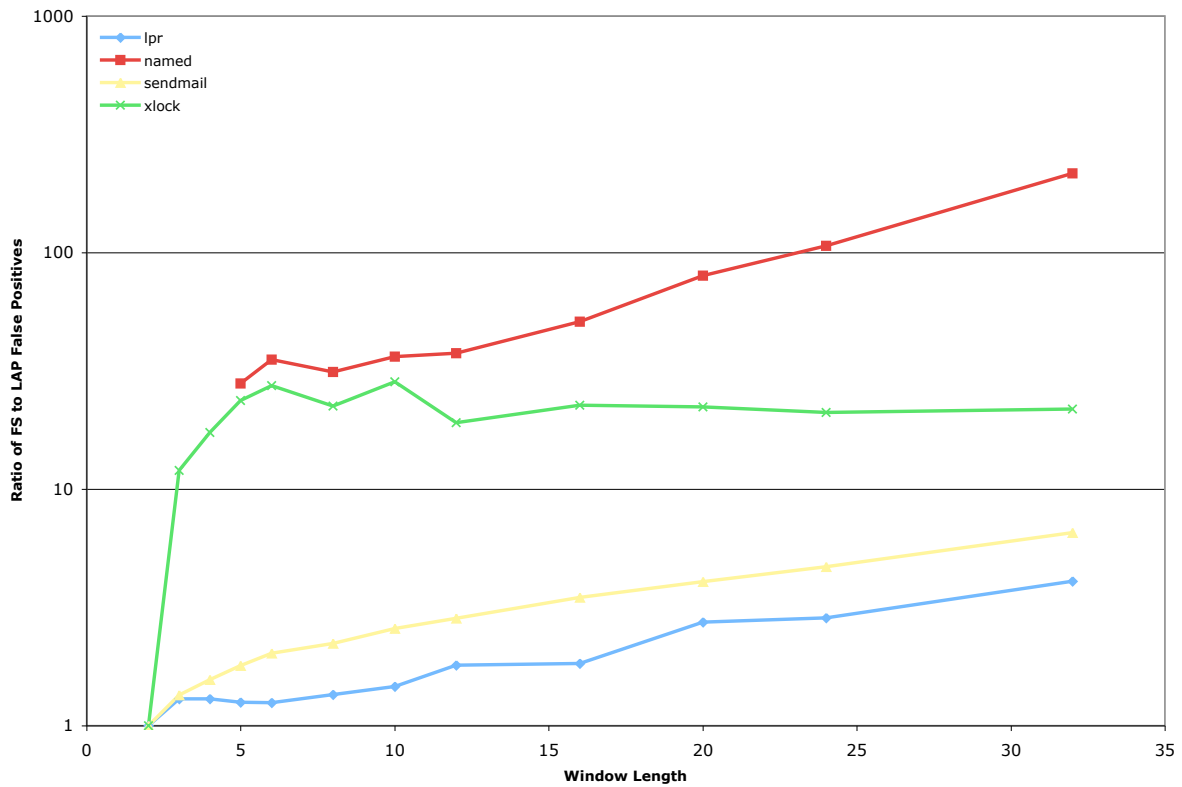


Fig. 2. Window length versus the ratio of the false positive rates of full sequence and lookahead pairs models for the four traces. Note the logarithmic scale for the false positive ratios.

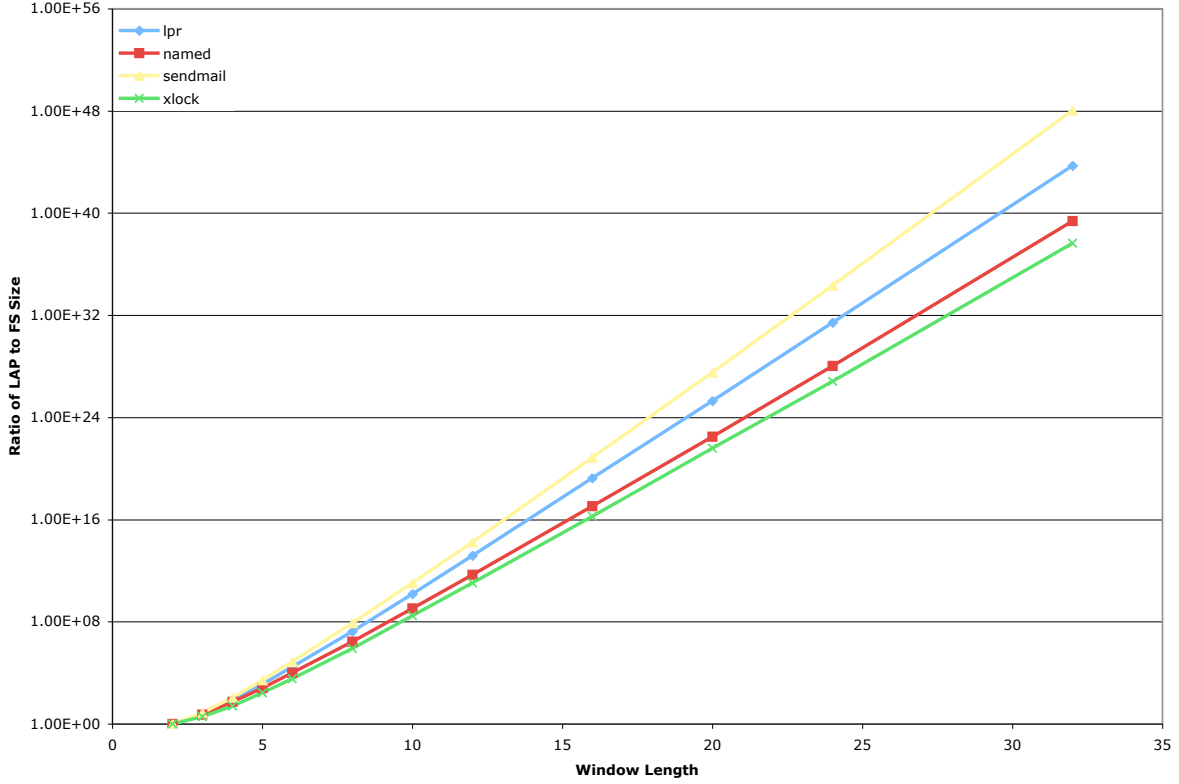


Fig. 3. Window length versus the ratio of the sizes of the lookahead pairs profiles to full sequence profiles for the four traces. The graph's shape, due to the slow rate of growth of FS profiles, is dominated by the LAP profile growth. Note the logarithmic scale for the false positive ratios.

Window	lpr		named		sendmail		xlock	
	LAP	FS	LAP	FS	LAP	FS	LAP	FS
2	1.98E+02	1.98E+02	1.83E+02	1.83E+02	4.23E+02	4.23E+02	9.60E+01	9.60E+01
3	1.91E+03	3.40E+02	1.85E+03	3.45E+02	9.20E+03	1.20E+03	5.55E+02	1.50E+02
4	3.08E+04	4.38E+02	2.76E+04	5.16E+02	2.80E+05	2.30E+03	4.94E+03	1.97E+02
5	6.69E+05	5.25E+02	4.74E+05	7.18E+02	9.96E+06	3.63E+03	6.28E+04	2.37E+02
6	1.75E+07	6.04E+02	9.42E+06	9.58E+02	3.96E+08	5.23E+03	9.04E+05	2.72E+02
8	1.30E+10	7.72E+02	4.45E+09	1.57E+03	7.41E+11	9.16E+03	2.80E+08	3.35E+02
10	1.49E+13	9.52E+02	2.67E+12	2.44E+03	1.61E+15	1.45E+04	1.14E+11	3.87E+02
12	1.71E+16	1.15E+03	1.67E+15	3.61E+03	3.86E+18	2.12E+04	4.91E+13	4.36E+02
16	2.90E+22	1.64E+03	8.46E+20	7.36E+03	2.74E+25	3.91E+04	9.74E+18	5.30E+02
20	4.29E+28	2.20E+03	4.21E+26	1.35E+04	2.21E+32	6.34E+04	2.37E+24	6.16E+02
24	7.49E+34	2.84E+03	2.42E+32	2.26E+04	2.07E+39	9.39E+04	4.78E+29	6.97E+02
32	2.33E+47	4.31E+03	1.13E+44	4.89E+04	1.84E+53	1.67E+05	3.88E+40	8.57E+02
64	1.66E+97	NA	2.23E+91	NA	8.01E+109	NA	7.33E+83	NA
128	2.24E+199	NA	2.27E+185	NA	3.33E+223	NA	5.48E+171	NA

TABLE II

PROFILE SIZES (IN TERMS OF NUMBER OF REPRESENTED SEQUENCES) FOR EACH OF THE FOUR TRACES FOR BOTH LOOKAHEAD PAIRS (LAP) AND FULL SEQUENCE (FS) ANALYSIS METHODS.

Window	Lookahead Pairs			Full Sequences		
	Anomalies	FP / call $\times 10^7$	FP / day	Anomalies	FP / call $\times 10^7$	FP / day
2	0	0.00	0.00	0	0.00	0.00
3	0	0.00	0.00	4	13.45	0.34
4	0	0.00	0.00	12	40.34	1.03
5	2	6.72	0.17	56	188.27	4.79
6	3	10.09	0.26	106	356.37	9.07
8	7	23.53	0.60	219	736.28	18.75
10	11	36.98	0.94	400	1344.81	34.24
12	17	57.15	1.46	638	2144.97	54.62
16	31	104.22	2.65	1581	5315.37	135.35
20	45	151.29	3.85	3590	12069.68	307.33
24	64	215.17	5.48	6845	23013.08	585.98
32	83	279.05	7.11	17920	60247.55	1534.08
64	130	437.06	11.13			
128	188	632.06	16.09			

TABLE III

FALSE POSITIVE RATES FOR THE `named` DATA SET WITH BOTH THE LOOKAHEAD PAIRS AND FULL SEQUENCE METHODS, USING SEVERAL SEQUENCE LENGTHS.

These are computed using the profile generated by our simulator and the equation described in Section II. One might expect them to match Figure 2. However, they differ greatly. Although three of the four traces (excluding `xlock`) show exponential growth, the rate of growth of false positives is much, slower. Also, there is not a qualitative match. The order from greatest to least in false positive ratios (`named`, `xlock`, `sendmail`, `lpr`) does not match the order of profiles size ratios (`xlock`, `named`, `lpr`, `sendmail`).

Although we can explain some of the differences in performance, other aspects are a bit more mysterious. For example, we find the shape of the graph in Figure 2 a bit puzzling. The `lpr` and `named` trace ratios increase gradually as the window length increases. The `xlock` ratios, though, do the opposite. Also, neither the `named` nor `xlock` show monotonic behavior, as one might expect by the behavior shown in Figure 3.

Table III shows the false positive rates per system call and per day for the `named` trace. Comparisons with real time for false positive rates are a better indicator than per system call because some programs make many more system calls than others. Also, anomalies, for the most part, are handled by human administrators. False positive rates given in time units provides more information about the feasibility of using such systems in production environments.

We can see here that for `named`, lookahead pairs

perform much better than full sequences, even for small window lengths. Full sequences creates more than one false positive per day at window lengths of four. Lookahead pairs do not impose that burden until lengths greater than 10. At a window length of 32, lookahead pairs experience a false positive about once every three and a half hours, compared with more than one a minute for full sequences.

Although lookahead pairs show much better false positive rates than full sequences, it is not clear that long windows ( $> 10$ ) are feasible in production systems due to performance reasons and absolute false positive rates. At a length of 32, the lookahead pairs false positive rate is 14 times greater than at 6, the value used in the original UNM simulations. Furthermore, if one implements lookahead pairs as in the `pH` system [12], each profile check is 5 times as expensive and the profile is 5 times as large. In the next section, we investigate one strategy for mitigating the runtime and accuracy cost of longer windows.

#### IV. RANDOM SCHEMA MASKS

One of the key results of Tan and Maxion’s work [16] was that the performance of `stide` was highly dependent upon sequence length. Specifically, they found that some attacks could not be detected if the window length was too small.

Longer windows increase the sensitivity of both the lookahead pairs and full sequences methods, while also

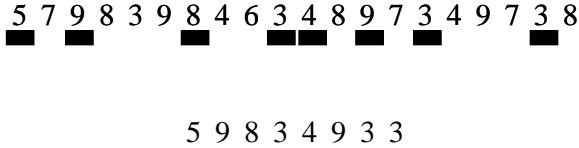


Fig. 4. A random window schema mask.

increasing false positive rates, as is evident in Table I. While longer windows can help detect some types of mimicry attacks, they cannot prevent all of them: if an attack imitates a full trace of system calls that is actually executed during normal operation, then neither the full sequence or lookahead pairs methods are capable of detecting the attack. However, longer sequences do constrain an attacker’s ability to use “impossible paths” to evade `stide`.

Here we investigate a system that may combine the ability to increase the difficulty of mimicry and “impossible paths” attacks while maintaining both low false positive rates and high performance.

#### A. Description

The basic idea is to maintain a large window length while ignoring some of the calls within that window. To illustrate this, consider Figure 4. The top list of numbers represents a window of length 20, with 5 as the most recent system call. The lines under the system call numbers represent window schema, the system calls we will actually consider. Here the size of the schema is 8, with the positions 0, 2, 6, 9, 10, 12, 14, and 18 chosen. This can then be represented as the bottom sequence.

By using schema that mask out several positions in the system call window we create another source of generalization. This generalization should result in fewer false positives while potentially maintaining sufficient sensitivity to attack-generated anomalies.

#### B. Results

For several window sizes and schema lengths, we randomly generated 10 schema masks and calculated their false positive rates. We test the new representation against only `named` due to the large number of experiments required. We believe these results translate to other programs.

Table IV gives evidence for the potential benefits of random schema masks. The false positive rates for each window size are dramatically lower for both lookahead pairs and full sequences using the schema masks. There is one interesting difference to note, however. The expectation was that the false positive rate for each schema

size would be similar but larger than that for that using windows without schema masks. This holds true for lookahead pairs but not for full sequences.

For full sequences the false positive rates are much *lower*. Why is this? It must be due to the generalization of the “holes” in the schema mask. Such holes would also permit more “impossible paths”; this fact does not seem to affect the algorithm’s ability to detect real-world exploits. When compared against the first exploit from Warrender et al. [20], the number of anomalies signaled by the simulator are similar to the unmasked versions. Therefore, it seems that using schema masks may be a viable way to extend window sizes.

Creating schema masks seems to create a generalization that allows attacks through. This is theoretically the case, but without knowing the schema mask beforehand, an attacker would have to construct an attack considering the window length instead of the schema size. The attacker might consider finding an attack using substitutions and relying on repeated attempts to determine the masked positions in the schema, but this would involve several, potentially thousands, of repeated attempts, and would likely alert administrators before it succeeded.

## V. RELATED WORK

Forrest et al. [3] first proposed that attacks on privileged program be detected by detecting unusual behavior, and that unusual program behavior could be detected by monitoring system calls. In this first paper, they also proposed a sequence-based method for modeling system calls that they referred to as using “lookahead pairs.” Although lookahead pairs performed well in their initial experiments, follow-up work from the UNM group [2], [8], [4], [20] instead switched to the “full sequence” method. (See Section II for a full description of these two methods.) The command-line analysis tool `stide` (sequence time-delay embedding) was made available by the UNM group along with most of the data sets that were used for these papers. Note that `stide` only implemented the full sequence method, not the lookahead pairs method.

Many other researchers have built upon this work. Although some of this literature focuses on applying the UNM group’s sequential analysis technique to other data sources [7], [14], the most significant work has focused on critiquing their approach and proposing alternatives. Wagner and Dean [17] were the first to note that an attacker can potentially evade detection of a sequence-based analysis system by “mimicking” normal behavior (i.e. by generating system call sequences present in the targeted program’s normal profile). Tan and Maxion [16]

Window	Lookahead Pairs			Full Sequences		
	6	8	10	6	8	10
12	0.78 (0.27)	1.20 (0.24)	1.34 (0.11)	1.11 (0.08)	1.46 (0.17)	1.85 (0.12)
16	1.06 (0.47)	1.60 (0.39)	2.05 (0.34)	1.69 (0.29)	2.58 (0.45)	3.49 (0.45)
20	1.52 (0.48)	2.09 (0.35)	2.46 (0.42)	2.21 (0.56)	3.99 (0.75)	5.40 (0.95)
24	2.00 (0.66)	2.59 (0.49)	2.85 (0.56)	3.04 (0.66)	5.18 (1.09)	9.25 (1.51)
32	2.01 (0.37)	2.82 (0.53)	3.35 (0.35)	4.36 (1.08)	8.46 (1.68)	15.83 (4.08)

TABLE IV

PER DAY FALSE POSITIVE AVERAGES (AND STANDARD DEVIATIONS) FOR LOOKAHEAD PAIRS AND FULL SEQUENCES WITH SCHEMA MASKS. WINDOWS OF 12, 16, 20, 24, AND 32 ARE COMPARED WITH SCHEMA THAT USE 6, 8, AND 10 VALUES IN THE COMPACTED SEQUENCE.

provided a theoretical analysis of the relationship between detection ability and sequence length for *stide*. In so doing they introduced the concept of minimal foreign sequences, which is the smallest injected sequence that could be detected. They later followed up this work with a full implementation of a mimicry attack on *stide* [15], which was published shortly before Wagner and Soto’s mimicry attack implementation [18]. Mimicry attacks were recently automated by Kruegel et al. [9].

The documented limitations in *stide* have, in part, inspired many others to develop alternative program behavior modeling techniques. Alternatives such as rule-based systems, frequency-based sequence detectors, and hidden Markov models, were shown to not have significant accuracy advantages relative *stide*, even though such methods all require more computational overhead [20]. Other methods, though, that incorporate additional information such as program counter state [11], have been shown to have lower false positives than *stide*.

Note that in all of this literature, the standard of performance has been the full sequence method as implemented by *stide*, not the lookahead pairs technique. For runtime performance and implementation reasons, Somayaji chose to go with the lookahead pairs model for pH, a system call-based real-time intrusion detection and response system [13], [12]. Somayaji’s dissertation [12] contained an entropy-based comparison between lookahead pairs and full sequences showing that lookahead pairs contained less information than full sequences, especially for larger windows. No comparison was performed on the basis of true and false positives.

Recently Wang, Parekh, and Stolfo [19] developed a “randomized testing” approach to n-gram analysis as part of their anagram network packet-based IDS. This technique is closely related to our random sequence masks; however, they are focused on partitioning rather than excluding data. Further, in the context of their (much

more complicated) system their randomization strategy often increased the observed rate of false positives. An interesting topic for future research is the evaluation of different randomization strategies for sequence-based anomaly detection methods in system calls, network packets, and other domains.

## VI. DISCUSSION

This research creates many questions about the learning and generalization ability of the different models of program behavior we have analyzed in this paper. It is remarkable that the dramatic difference in the performance between lookahead pairs and full sequences has never been documented before. After some preliminary testing, the UNM group assumed that lookahead pairs had similar behavior to full sequences and moved to full sequences when an implementation became available. The work on alternative models and mimicry attacks then simply ignored the lookahead pairs method. One master’s thesis [10] claimed that Hofmeyr showed that “fixed sequences give better discrimination than lookahead pairs”. It is true that full sequences generalize less than lookahead pairs, but *it has never been shown that they are a better model for detecting intrusions*. Section III provides evidence that lookahead pairs is a better model for intrusion detection than full sequences. Indeed, lookahead pairs appear to be significantly better than any of the data models explored in the Warrender paper [20].

Arguably, the lookahead pairs method is a better anomaly detection method than the full sequences method because it generalizes more. That is, multiple entries in the full sequences model correspond to one entry in the lookahead pairs model. There are three sources of generalization apparent in these models: the finite window length in all the models, the substitutions allowed in lookahead pairs, and the masked calls within

the windows in the schema masks models. According to our analysis, these generalization methods empirically generate fewer false positives.

It is an open question what the costs and benefits are for each kind of generalization. Much of the literature on mimicry attacks shows that larger windows, and thus less generalization through window length, are better for detecting attacks. However, the larger false positive rates incurred through less generalization have driven research in statically generated “models” of normal behavior that cannot incur false positives. Model-based approaches overgeneralize in other ways, however, leading to vulnerabilities beyond mimicry attacks [6].

The generalization from substitutions in the lookahead pairs model is an even more puzzling problem. The source of the difference in the behavior between lookahead pairs and full sequences, described by the generalization equation in Section II, does not fully explain the differences between the two methods. If the false positive rates were proportional to sizes, the LAP method would have even fewer false positives, and would potentially be unable to detect many true positives. However, the UNM showed that for sizes of 6 to 10 the two techniques had similar detection capabilities [3], [20], [12]. This is an important problem to solve because a decade of research, both within and outside the UNM group, has assumed that full sequences is better than lookahead pairs. Is this truly the case for programs beyond those studied by the UNM group?

Another question generated by this research concerns the false positive rates for full sequences with schema masks. Those rates are both lower than the representative window lengths, but also lower than the schema mask size. This is not the case for the lookahead pairs method. Could it be that adding “holes” to the representation allows the full sequences model to mimic the substitution generalization that apparently gives lookahead pairs its advantage? Further research is needed to resolve these questions.

## VII. CONCLUSION

Given the large number of papers devoted to system call based anomaly detection, it is remarkable that there has never been a direct comparison between lookahead pairs and full sequences. We have provided that comparison in this paper. Although it was previously assumed that full sequences were either better or similar in performance to lookahead pairs, we have shown that the opposite is true.

In addition, we have shown that anomaly systems built around lookahead pairs are practical with long

window lengths, and that very long window lengths can be accommodated through the use of schema masks. By using random schema, an IDS can force an attacker to craft attacks that assume a complete window length—a more difficult task.

When referencing the UNM efforts, research in system call based anomaly detection and mimicry attacks should compare against both lookahead pairs as well as full sequences. Our surprising results indicate that there is significant scope for better theoretical understanding of the program behavior.

## ACKNOWLEDGMENTS

We thank the members of Carleton Computer Security Laboratory and the anonymous reviewers for their suggestions.

This work was supported by the Discovery grant program from Canada’s National Sciences and Engineering Research Council (NSERC) and MITACS.

## REFERENCES

- [1] Stephanie Forrest et al. Computer immune systems—data sets and software. <http://www.cs.unm.edu/~immsec/systemcalls.htm>, December 2006.
- [2] Stephanie Forrest, Steven Hofmeyr, and Anil Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, October 1997.
- [3] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 120, Washington, DC, USA, 1996. IEEE Computer Society.
- [4] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3), 1998.
- [5] Steven A. Hofmeyr and Julie Rehmeyr. Stide: Sequence time-delay embedding, 1998.
- [6] Hajime Inoue and Anil Somayaji. What happened to anomaly detection? Technical Report TR-07-09, School of Computer Science, Carleton University, Ottawa, Canada, March 2007.
- [7] Anita Jones and Yu Lin. Application intrusion detection using language library calls. In *Proceedings of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, December 10–14, 2001.
- [8] Andrew P. Kosoresow and Steven A. Hofmeyr. Intrusion detection via system call traces. *IEEE Software*, 14(5):35–42, September/October 1997.
- [9] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Automating mimicry attacks using static binary analysis. In *14th Annual Usenix Security Symposium*, Aug 2006.
- [10] Svetlana Radosavac. Detection and classification of network intrusions using hidden mark models. Master’s thesis, University of Maryland, 2002.
- [11] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.
- [12] Anil Somayaji. *Operating System Stability and Security through Process Homeostasis*. PhD thesis, University of New Mexico, 2002.

- [13] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, August 14–17, 2000.
- [14] Matthew Stillerman, Carla Marceau, and Maureen Stillman. Intrusion detection for distributed applications. *Communications of the ACM*, 42(7):62–69, July 1999.
- [15] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID '02)*, 2002.
- [16] Kymie M. C. Tan and Roy A. Maxion. “Why 6?” defining the operational limits of stide, an anomaly-based intrusion detector. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 188, Washington, DC, USA, 2002. IEEE Computer Society.
- [17] David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 156–169, 2001.
- [18] David Wagner and Paolo 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, New York, NY, USA, 2002. ACM Press.
- [19] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID '06)*, volume 4219 of *LNCS*, pages 226–248, Sep 2006.
- [20] Christina Warrrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.