Anomaly Detection in Embedded Systems

Roy A. Maxion and Kymie M.C. Tan

October 2001

CMU-CS-01-157

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in the IEEE Transactions on Computers, January 2002.

This research was supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under contracts F30602-99-2-0537 and F30602-00-2-0528.

Keywords: Anomaly, anomaly detection, coverage, dependability.

Abstract

By employing fault tolerance, embedded systems can withstand both intentional and unintentional faults. Many fault-tolerance mechanisms are invoked only after a fault has been detected by whatever fault-detection mechanism is used, hence the process of fault detection must itself be dependable if the system is expected to be fault tolerant. Many faults are detectable only indirectly, as a result of performance disorders that manifest as anomalies in monitored system or sensor data. Anomaly detection, therefore, is often the primary means of providing early indications of faults. As with any other kind of detector, one seeks full coverage of the detection space with the anomaly detector being used. Even if coverage of a particular anomaly detector falls short of 100%, detectors can be composed to effect broader coverage, once their respective sweet spots and blind regions are known. This paper provides a framework and a fault-injection methodology for mapping an anomaly detector's effective operating space, and shows that two detectors, each designed to detect the same phenomenon, may not perform similarly, even when the event to be detected is unequivocally anomalous, and should be detected by either detector. Both synthetic and real-world data are used.

1 Introduction

As computer systems become more miniaturized and more pervasive, they will be embedded in everyday devices with increasing frequency, even to the point at which domestic and industrial consumers may not be aware of their presence. Some truck tires, for example, will soon have a processor and a pressure sensor/transponder embedded in the rubber, because this is cheaper than fitting in-hub pressure sensors in the wheels of old trailers on big rigs. Some laptop-computer batteries contain an embedded computer to track the charge remaining, thereby ensuring that battery's memory travels with the battery even when it is moved to another laptop computer. Disk drives may contain one or two embedded computers (one controller, one DSP chip). Even operating systems like Unix are being embedded in television set-top boxes (enabling pausing a live television broadcast), vending machines, Internet appliances, and the International Space Station (for controlling vibration damping) [1].

Many of these devices with embedded computers will be intrinsically safety-critical or mission-critical, and therefore will require a higher level of dependability than usual – automobile and aircraft engine controllers are one example. It is presumed that fault tolerance, which is one way of achieving high dependability, will be employed in such devices.

Several methods of fault tolerance require that a fault be detected before bringing fault-tolerance measures to bear on it. One salient example is recovery blocks [2] [3]. Fault detection, therefore, is an essential first step in achieving dependability. If the detector is not reliable, the fault-tolerating mechanisms will not be effective, because they will not be activated.

Faults can be detected either explicitly or implicitly. When a fault is detected explicitly it is typically through pattern recognition, wherein a signature is detected that is directly linked to a particular fault. When a fault is detected implicitly, it is usually due to having detected some indirect indicator, such as an anomaly, that may have been caused by the fault. System-performance anomalies are often the only indicators of problems, because some faults have no stable, explicit signature; they're indicated only through unusual behaviors. One such example is the fault condition known as an Ethernet broadcast storm, which is indicated indirectly by anomalously-high packet traffic on a network [4]. Another example is an increasing error rate reported by software sensors, and observed in system event logs; disk surface failures can be indicated this way [5] [6]. In the Ethernet example, measures of packet traffic served as a sensor. In system event logs, many different measures are available [7]. As noted in [4] there can be many sensors measuring the state of a network, system or process. These sensors can be hardware or software, although recent trends have been mainly toward software sensors [8].

The data produced by such sensors are referred to as sensor data or a sensor-data stream. The data in the sensor-data stream can be numeric or categorical. Numeric data are usually continuous, are on a ratio scale, have a unique zero point, and have mathematical ordering properties (e.g., taking differences or ratios of these measures makes sense). Categorical data, sometimes referred to as nominal data, are discrete, usually consist of a series of unique labels as categories, and have no mathematical ordering properties (e.g., an apple is not twice an orange) [9]. It seems likely that as computing power increases, more of the sensor data will be in the form of categorical data [8] [10], hence anomaly detectors will be required to operate primarily on categorical data, presenting a real challenge to developers and users of such sensors, because categorical data are much more difficult to handle statistically than numeric data are. This paper focuses on detecting anomalies in categorical data.

An anomaly occurring in such sensor data is often the indirect or implicit manifestation of a fault or condition somewhere in the monitored system or process. Detecting such anomalies, therefore, can be an important aspect of maintaining the integrity, reliability, safety, security and general dependability of a system or process. Since anomaly detection is on the front line of many fault-tolerance and dependability mechanisms, it is essential that it is, itself, reliable. One way to gauge its reliability is by its coverage.

Coverage is a figure of merit that gauges the effectiveness of a detection or testing process. Historically, a system's *coverage* has been said to be the proportion of faults from which a system recovers automatically; the faults in this class are said to be *covered* by the recovery strategy [11].

Coverage can also be viewed as the probability that a particular class of conditions or events is detected before a system suffers consequences from a missed or false detection. Another definition of coverage, and the one used in this paper, is: a specification or enumeration of the types of conditions against which a particular detection scheme guards [12]. More succinctly, the coverage of an anomaly detector is the extent to which it detects, correctly, the events of a particular anomaly class. The motivation for the concern with coverage is that one needs to know if and when one's anomaly detection system will experience a Type I error (a true null hypothesis is incorrectly rejected) or a Type II error (a false null hypothesis fails to be rejected), so that one can take precautionary measures to compensate for such errors. If an anomaly detector does not achieve complete coverage, its suitability for use should be scrutinized carefully. Anomaly classes will be discussed in Section 6.

This paper addresses the issues of how to assess the coverage of an anomaly detector, how to acquire ground-truth test data to aid in that assessment, how to inject anomalous events into the test data, and how to map the coverage of the detector in terms of sweet spots (regions of adequate detection) and blind regions (regions of inadequate detection). Once a detector's coverage map is ascertained, it can be used to judge the suitability of the detector for various situations. For example, if the environment in which the detector is deployed will never experience a condition in the detector's blind region, then the detector can be used without adverse consequences.

2 Problem and objective

A critical problem is that there is little clarity in the literature regarding the conditions under which anomaly detection works well or works poorly. To gain that clarity it is necessary to understand the details of precisely how an anomaly detector works, i.e., what the detector sees, and what phenomena affect its performance as the stream of sensor data passes through the detector's range of perception. Similarly, one would want to know how a sorting algorithm views the data it is sorting, as well as which characteristics of the data, e.g., presortedness, impinge on the algorithm's efficacy.

The objectives of the present work are (1) to understand the details of how an anomaly detector works, as a stream of sensor data passes through the detector's purview; and (2) to use that understanding to guide fault-injection experiments in which anomalies are injected into normal background data, the outcome of which is a map illustrating the detector's regions of sensitivity and/or blindness to anomalies. The results will address such questions as:

- What is the coverage of an anomaly detector?
- How does one assess that coverage?
- Do all anomaly detectors have the same coverage, for a given set of anomalies embedded in background data?

- Can anomaly detectors be composed to attain greater coverage than that achieved by a single anomaly detector used alone?

These issues are addressed using fault injection, a well-known technique for evaluating detection systems [13] [14]. Synthetic data are used to address the usual problem of determining ground truth. To facilitate simplicity and clarity, the most basic type of anomaly detection is used, namely that of a sliding-window detector moving over a univariate stream of categorical data. Anomalous sequences are considered to be contiguous. Temporal anomalies, not addressed here, can be treated similarly if they are aggregated using an appropriate feature-extraction mechanism.

3 What is an anomaly?

According to Webster's dictionary, an anomaly is something different, abnormal or peculiar; a deviation from the common rule; a pattern or trait taken to be atypical of the behavior of the phenomenon under scrutiny. This definition, fitting as it may be for everyday purposes, is too vague to be scientifically useful. A more apt definition is this: an anomaly is an event (or object) that differs from some standard or reference event, in excess of some threshold, in accordance with some similarity or distance metric on the event. The reference event is what characterizes normal behavior. The similarity metric measures the distance between normal and abnormal. The threshold establishes the minimum distance that encompasses the variation of normalcy; any event exceeding that distance is considered anomalous. The specifics of establishing the reference event, the metric and the threshold are often situation-specific and beyond the scope of this paper, although they will be addressed peripherally in Section 7.1.

Determining what constitutes an anomalous element in a stream of numeric data is intuitive; a data element that exceeds, say, the mean plus three standard deviations may be considered anomalous. Determining what constitutes an anomaly in categorical data, which is the specific problem addressed here, is less intuitive, since it makes no sense to compute the mean and standard deviation (or any other numerically-based measure) of categorical values such as *cat* or *blue*, even if these categories are translated into numbers. In categorical data, anomalous events are typically defined by the probabilities of encountering particular juxtapositions of symbols or subsequences in the data stream; i.e., symbols and subsequences in an anomaly are juxtaposed in unexpected ways.

Categorical data sets are comprised of sequences of symbols. The collection of unique symbols in a data set is called the alphabet. Typically, a data set will be characterized in terms of what constitutes normal behavior for the environment from which the data were drawn. The data set so characterized is called the training data. Training data may be obtained from some application, e.g., a process-control application that is providing monitored data for consumption by various analysis programs. Training data are obtained from the process over a period of time during which the process is judged to be running normally. Within these normal data, the juxtapositions of symbols and subsequences would be considered normal, provided that no faults or unusual conditions prevailed during the collection period. Once the training data are characterized (termed the training phase), characterizations of new data, monitored while the process is in an unknown state (either normal or anomalous), are compared to expectations generated by the training data. Any sufficiently unexpected juxtaposition in the new data would be judged anomalous, the possible manifestation of a fault.

4 Anomaly causes and manifestations

What causes an anomaly, and what does an anomaly look like? An example from a semiconductor fabrication process illustrates. If one attaches a sensor to an environment, such as the plasma chamber of a reactive ion etcher, the sensor data will comprise a series of normal categorical values (given a normally operating etcher). When a fault occurs in the fabrication process, the fault will be manifested as an event (a series of sensor values) embedded in otherwise normal sensor data. That event will contain one or more data values that are related to the normal data values in one of two ways: (1) the embedded event could contain symbols commonly found and commonly juxtaposed in normal data; (2) the embedded event could contain symbols and symbol juxtapositions that are anomalous with respect to those found in normal data. Thus the fault could manifest itself as an event injected into a normal stream of data, and that event could be regarded as normal or it could be regarded as anomalous. There are three phenomena that could make an event anomalous:

Foreign symbols. A foreign symbol is a symbol not included in the training-set alphabet. For example, any symbol, such as a Q, not in the training-set alphabet comprising A B C D E F would be considered a foreign symbol. Detection of events containing foreign symbols, called foreign-symbol-sequence anomalies, is straightforward.

Foreign n-grams/sequences. An n-gram (a set of n ordered elements) not found in the training dataset (and also not containing a foreign symbol) is considered a foreign n-gram or foreign sequence, because it is foreign to the training dataset. A foreign n-gram event contains n-grams not present in the training data. For example, given an alphabet of A B C D E F, the set of all bigrams would contain AA AB AC ... FF, for a total of $6^2 = 36$ (in general, for an alphabet of α symbols, the total possible n-grams = α^n). If the training data contained all bigrams except CC, then CC would be a foreign n-gram. Note that if a foreign symbol (essentially a foreign unigram) appears in an n-gram, that would be a foreign-symbol event, not a foreign n-gram event. In real-world, computer-based data it is quite common that not all possible n-grams are contained in the training data, partly due to the relatively high regularity with which computers operate, and partly due to the large alphabets in, for example, kernel-call streams.

Rare n-grams/sequences. A rare n-gram event, also called a rare sequence, contains n-grams that are infrequent in the training data. In the example above, if the bigram AA constituted 96% of the bigrams in the sequence, and the bigrams BB and CC constituted 2% each, then BB and CC would be rare bigrams. An n-gram whose exact duplicate is found only rarely in the training dataset is called a rare n-gram. The concept of *rare* is determined by a user-specified threshold. A typical threshold might be .05, which means that a rare n-gram would have a frequency of occurrence in the training data of not more than 5%. The selection of this threshold is arbitrary, but should be low enough for "rare" to be meaningful.

5 The ken of an anomaly detector

An anomaly detector determines the similarity, or distance, between some standard event and the possibly-anomalous events in its purview; it can't make decisions about things it can't see. The purview of a sliding-window detector is the length of the window. Since not all anomalies are the same size as the detector window, and such size differentials can affect what the

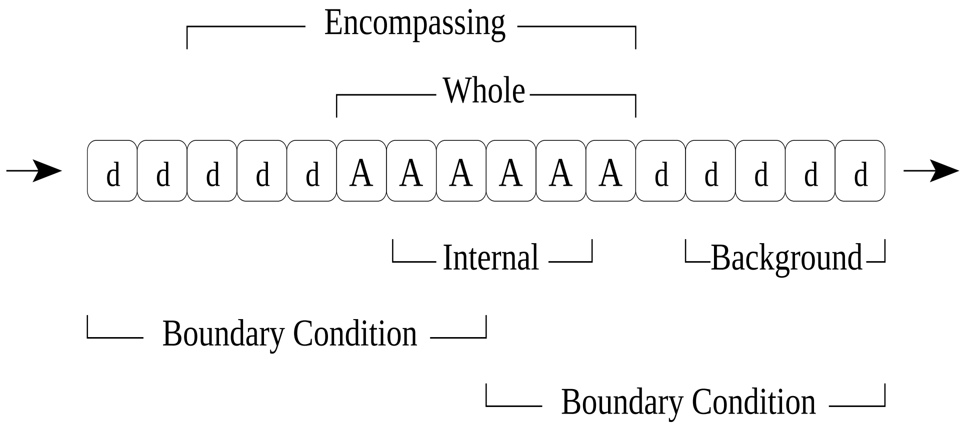

Figure 1: The ken of an anomaly detector: different views of an anomaly (depicted by AAAAAA) embedded in a sensor-data stream (depicted by ddddd) from the perspective of a sliding-window anomaly detector. Arrows indicate direction of data flow.

detector detects, it is useful to pursue the idea of a detector's ken, or range.

The word *ken* means the extent or range of one's recognition, comprehension, perception or understanding; one's horizon or purview. Thus it seems appropriate to ask, what is the ken of an anomaly detector? The univariate case is shown in Figure 1 which depicts a stream of sensor data (ddddd) into which an anomalous event (AAAAAA) has been injected. The right-directed arrows indicate that the data are moving to the right with respect to the detector window, as time and events pass.

The width of the window through which the detector apprehends the anomaly can take on any value, typically based on the constraints of the environment or situation in which the detector is being used. The extent to which the detector window overlaps the anomaly can be thought of as the detector's view of the anomaly. It is natural to focus on the case in which the window is the same size as the anomaly *and* the entire anomaly is captured exactly within the window. This is called the *whole view*. There are, however, a number of other cases, illustrated in the figure. When the size of the detector window is less than the length of the anomaly, the detector has what is called an *internal view*. For the case in which the detector window is larger than the anomaly, both anomalous and normal background data are seen - this is the *encompassing view*. Irrespective of the width of the window, as time passes and an anomalous event moves through the window, the event presents itself to the detector in different perspectives. Of particular interest are situations termed *external boundary conditions*, used interchangeably here with the term *boundary conditions*. These arise at both ends of an injected sequence embedded in normal data, when the leading or trailing element of the anomaly abuts the normal data. Boundary conditions occur independently of the relative sizes of the detector window and anomaly (except in the degenerate case of size one). In a boundary condition, the detector sees part of the anomaly and part of the background data. The *background view* sees only background data, and no anomalies.

It will be shown later that the detector views and conditions just discussed will be important in determining precisely what an anomaly detector is capable of detecting, as well as what may cause an anomaly detector to raise an alarm, even when it should not. Note that these conditions depend on the size of the injected event relative to the size of the detector window. Table 1 summarizes the conditions.

Conditions	$DW < AS$	$DW = AS$	$DW > AS$
Internal	x		
Boundary	x	x	x
Encompassing			x

Table 1: Conditions of interest that ensue with respect to detector-window size (DW) and anomaly size (AS).

6 Anomaly space

It is important to note that anomalies can be composed of subsequences of various types, three of which were identified in Section IV: foreign symbols, foreign n-grams and rare n-grams. A fourth type is a common n-gram, an n-gram that appears commonly (not rarely) in the normal data. Henceforth the terms n-gram and sequence will be used interchangeably, i.e., foreign n-gram and foreign sequence refer to the same thing.

That an anomalous sequence can be composed of several different kinds of subsequences, along with the concept of internal sequences and boundary sequences, gives rise to the idea of creating a map of the anomaly space for sliding-window detectors. Given such a map, it should be possible to determine the extent to which that map is covered by a particular anomaly detector. It is not the goal of this paper to do that, but rather to show that two detectors can have unexpectedly different coverages, even when encountering the same events embedded in the same data; that is, different detection capabilities will arise from the use of different metrics and different detectors.

An anomaly-space map is shown in Figure 2. The map is described in the figure caption and in the paragraph following it. The window size of the detector, relative to the size of the anomaly, is shown in the three columns of the figure: detector window size less than anomaly size, detector window size equal to the anomaly size, and detector window size greater than the anomaly size. For each of these conditions the figure addresses three kinds of anomalies: foreign-symbol-sequence anomalies (sequences comprising only foreign symbols); foreign-sequence anomalies (sequences comprising only foreign sequences); and rare-sequence anomalies (sequences comprising only rare sequences).

The following material expands the description of a cell by selecting as an example the anomaly type FF AI AB, depicted at the upper left of the figure. This is a sequence of foreign symbols (FF) composed of alien internal sequences (AI) and having alien external boundaries (AB). The term *alien* is an umbrella term used to refer to sequences that do not exist in the normal (training) data, irrespective of the characteristics that make them foreign, unlike the more closely-defined terms *foreign symbol* and *foreign sequence*.

FF is a foreign-symbol-sequence anomaly composed only of foreign symbols. In this specific case, when the anomalous sequence FF AI AB slides past a detector window whose size is less than the size of the anomaly, the detector will first encounter the leading edge of the anomaly. That leading edge will be alien, i.e., the sequence containing the first element of the anomaly and the normal element immediately preceding it is not a sequence that exists in the normal (training) data, and therefore will be anomalous. As the anomaly moves through the detector window, each internal, detector-window-sized subsequence of the anomaly will be alien. As the anomaly

Foreign-Symbol-Sequence Anomalies

$DW < AS$	$DW = AS$	$DW > AS$
FF AI AB	FF – AB	FF AE AB

Foreign-Sequence Anomalies

$DW < AS$	$DW = AS$	$DW > AS$
FS AI AB FS RI AB FS CI AB	FS – AB	FS AE AB ~~FS RE AB~~ ~~FS CE AB~~
FS AI RB FS RI RB FS CI RB	FS – RB	FS AE RB ~~FS RE RB~~ ~~FS CE RB~~
FS AI CB FS RI CB FS CI CB	FS – CB	FS AE CB ~~FS RE CB~~ ~~FS CE CB~~

Rare-Sequence Anomalies

$DW < AS$	$DW = AS$	$DW > AS$
RS AI AB RS RI AB RS CI AB	RS – AB	RS AE AB RS RE AB ~~RS CE AB~~
RS AI RB RS RI RB RS CI RB	RS – RB	RS AE RB RS RE RB ~~RS CE RB~~
RS AI CB RS RI CB RS CI CB	RS – CB	RS AE CB RS RE CB ~~RS CE CB~~

Figure 2: Anomaly space. The first two letters in each cell identify the type of anomalous sequence (FS: foreign-sequence anomaly; RS: rare-sequence anomaly; FF: foreign-symbol-sequence anomaly). The next two letters identify the type of condition (internal (alien, rare or common) or encompassing (alien, rare or common)); the last two letters refer to the boundary conditions (alien, rare or common). DW $<$ AS indicates detector window smaller than anomaly size; DW $>$ AS analogously indicates window larger; when DW $=$ AS there are no internal or encompassing conditions, indicated by dashes replacing the middle two letters. Impossible conditions are struck out.

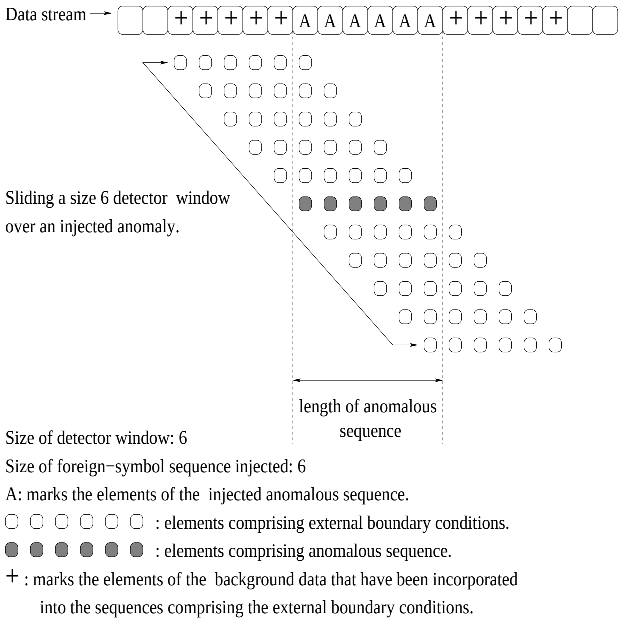

Figure 3: Foreign-symbol-sequence anomaly injected into background data. External boundary conditions are shown for detector window size of 6 and anomaly size of 6: $DW = AS$.

passes out of the window, its trailing edge will form another alien boundary.

Figure 3 illustrates a sliding-window detector moving over an anomaly injected (synthetically) into a data stream. The detector window and the anomaly size are the same: $DW = AS = 6$. The shaded boxes depict the injected FF anomaly, which raises an alarm as the detector window is positioned exactly over it. Ten sequences result from the interaction between the background and the injected anomaly. These ten sequences comprise the boundary conditions that may affect the response of the detector as its window slides over the injected anomaly, depending on whether or not additional anomalies are caused by the anomaly-background boundary interactions. The composition of an injected anomaly, as well as the position of the injection in the background data, must be carefully controlled to avoid the creation of additional anomalies at the boundaries of the injection; Section 7.4 provides details.

Note that the anomalies depicted in Figure 2 reflect the restricted needs of an experimental regime, and do not express all of the conditions that might be encountered in a real-world environment. In the FSRIRB anomaly, for example, *all* of the subsequences comprising the internal condition are rare, and *all* the subsequences that make up the external condition are rare. In the real world, the subsequences comprising these conditions might be a mixture of rare, foreign and common. The anomaly space is constructed as it is in order to effect experimental control and to reduce confounding in which a detector's response cannot be attributed to any single phenomenon, but rather is due to the interaction of several phenomena.

7 Mapping the detection regions

Different detectors may cover different parts of the anomaly space depicted in Figure 2. This section describes an experiment showing how well a selected portion of the anomaly space is covered by two different detectors, Markov and Stide, whose detection mechanisms will be explained below. The selected portion of the space is foreign-sequence anomalies as shown in the fifth row of the foreign-sequence section of the figure: FS RI RB, FS – RB, and FS RE RB. Because the last of these is not possible, focus is limited to the first two. These cells were chosen because they contain events that would unequivocally be termed anomalous by both of the detectors used in the experiment: both anomalies are foreign sequences with rare boundary conditions. For the case in which the detector window size is less than the anomaly size, the subsequences that make up the internal conditions are all rare, hence FS RI RB. For the case in which the detector window size is equal to the anomaly size, no internal conditions will be extant, hence FS – RB. The sequence FS RE RB is not possible, because the sequences that make up the encompassing condition will contain the foreign sequence FS. Sequences that contain foreign subsequences will themselves be foreign sequences; consequently it is not possible to have a rare sequence that contains a foreign subsequence.

The following subsections describe the detectors used in the coverage-mapping experiment, the methods for generating the data used to test detector coverage (background data, anomaly data and anomaly-injected testing data), and the regime for running the detectors in the experiment.

7.1 Detectors

To illustrate that different detectors may cover different parts of the anomaly space, two detectors were tested: Markov and Stide. Each of these is described below.

7.1.1 Markov detector

Most engineered processes, including ones used by or being driven by computers, consist of a series of events or states. While the process is running, the state of the process will change from time to time, typically in an orderly fashion that is dictated by some aspect of the process itself. Certain kinds of orderly behavior are plausible approximations to problems in real-world anomaly detection and, moreover, they facilitate rigorous statistical treatment and conclusions. The anomaly-detection work in this paper focuses on the kind of orderly behavior that corresponds to Markov models.

The Markov anomaly detector determines whether the states (events) in a sequential data stream, taken from a monitored process, are normal or anomalous. It calculates the probabilities of transitions between events in a training set, and uses these probabilities to assess the transitions between events in a testing set. These states and probabilities can be described by a Markov model. The key aspect of a Markov model is that the future state of the modeled process depends only on the current state, and not on any previous states [15, 16].

A Markov model consists of a collection of all possible states and a set of probabilities associated with transitioning from one state to another. A graphical depiction of a Markov model with four states is shown in Figure 4 in which the states are labeled with the letters A, B, C and D. Although the arcs are not explicitly labeled in the figure, they can be thought of as being labeled with the probabilities of transitioning from one state to another, e.g., from state A to state B. The transition probabilities can be written in a transition matrix, as shown in Figure 5, in which the letters indicate states and the numbers indicate transition probabilities. The probability of transitioning from D to A, for example, is 1; from D to any other state is 0.

The transition probabilities are based on a key property, called the Markov assumption, and can be written formally as follows. If X_t is the state of a system at time t, then:

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, ..., X_0 = x_0) = P(X_{t+1} = x_{t+1} | X_t = x_t)$$

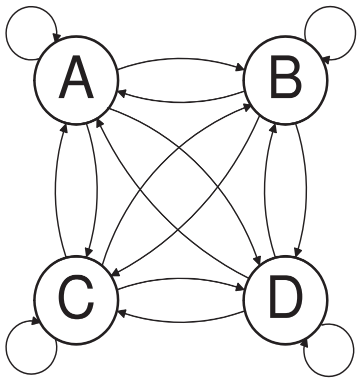

Figure 4: Four-state Markov model $alphabet comprised of four symbols$. Letters indicate states; arrows indicate transition probabilities. A transition can be made from any state to any other state, with a given probability.

Hence the probability of being in state $X_{t+1} = y$ at time $t+1$ depends *only* on the immediately preceding state $X_t = x$ at time t, and not on any previous state leading to the state at time t. Therefore the transition probability, P_{xy}, denoting the progression of the system from state x to state y, can be defined as:

$$P_{xy} = P(X_{t+1} = y | X_t = x)$$

Readers interested in further details are encouraged to consult the large literature on Markov models, e.g., [15].

Weather prediction provides a nice illustration of a Markov process. In general, one can usually predict tomorrow's weather based on how the weather is today. Over short time periods, tomorrow's noontime temperature depends only on today's noontime temperature. The previous day's temperature is correlated, but provides no additional information beyond that contained in today's measurement. So, there is some reasonably high probability that tomorrow will be like today. If tomorrow's weather is *not* like today's, then one's expectations would be violated, and one would feel surprised. The degree of surprise can be used in anomaly detection: the more surprised one is to observe a certain event or outcome, the more anomalous is the event, and the more it draws one's attention.

$$
\begin{array}{cc|cccc}
 & & & \text{State } X_{t+1} & & \\
 & & \text{A} & \text{B} & \text{C} & \text{D} \\
\hline
 & \text{A} & 0.00 & 1.00 & 0.00 & 0.00 \\
 & \text{B} & 0.00 & 0.00 & 1.00 & 0.00 \\
\text{State } X_t & & & & & \\
 & \text{C} & 0.00 & 0.00 & 0.00 & 1.00 \\
 & \text{D} & 1.00 & 0.00 & 0.00 & 0.00
\end{array}
$$

Transition sequence: ABCDABCD...

Figure 5: Transition matrix for four-state Markov model (alphabet comprised of four symbols: A, B, C and D). Letters indicate states; numbers indicate probabilities of transitioning from one state to another. Example: probability of transitioning from D to A is 1; probability of transitioning from D to any other state is 0.

Basing an anomaly detector on a discrete Markov process requires three steps. First, a state transition matrix is constructed, using the training data; the training data represent the conditions that are considered to be normal. For example, if sensor data are collected from an aircraft turbine that is running under normal operating conditions, these data would be used as training data. From these data would be constructed the state transition matrix that represents normal turbine behavior.

The second step is to establish a metric for surprise. This is generally a distance (or similarity) measure that determines how dissimilar from normal a process can be, while remaining within the bounds of what is considered to be normal operating behavior. If the threshold of dissimilarity is exceeded, then the observed behavior, as reflected in the sensor data, is judged to be abnormal, or anomalous. In the case of the Markov-model approach, if a transition is judged to be highly probable (e.g., has a probability of 0.9), then its surprise factor is $1.0 - 0.9 = 0.1$. If, in this example, the threshold were set at 0.9, then a surprise factor of 0.1 would not be anomalous. If the surprise factor had been 0.98, for example, then the transition would have been considered anomalous. The more the surprise factor exceeds the surprise threshold, the more anomalous the event will seem. Given a threshold of 0.9, a surprise factor of 0.91 would be deemed anomalous, but a surprise factor of 0.99 would be regarded as fractionally more anomalous.

The third step is to examine the test data to see if they fall within the expectations established by the training data. As each state transition in the test data is observed, its probability of occurring in normal data is retrieved from the transition matrix derived from the training data. If the transition under scrutiny has a surprise factor that exceeds the surprise threshold, then the event in the testing data is considered anomalous.

The Markov-based anomaly detector that is used in this paper is based on the ideas presented in this section. The states in the model do not necessarily need to correspond to single events or unigrams; a state can be composed of a series of events, too. In a case where multiple events comprise a state, the collection of states in the Markov model spans the combinations of unigrams of a specified length as present in the training data. Consider, for example, the sequence A B C D E F. A 3-element window is moved through the sequence one event at a time: the first window position would contain A B C; the second window position would contain B C D, and so forth. In using a Markov model to assess the surprise factor of the transition between the first and second windows, the states would comprise a series of three events or unigrams, i.e., equivalent to the window size. Notice that in the transition from the first window to the second window, the event A is eliminated, and the event D is added. Since the events B C are common to both windows, it is the addition of event D which drives the surprise factor. Therefore, the resulting surprise factor reflects on the event D following the series of events A B C. A formal description of the training and testing processes used in conjunction with such a Markov model is given next.

Markov training stage Primitives similar to those described in [17] are defined to facilitate the description of the training procedure. Let Σ denote the set of unique elements (i.e., the alphabet of symbols, or the set of states) in a sequential stream of data. A *state* in a Markov model is denoted by s and is associated with a sequence (window) of length N over the set Σ. A *transition* is a pair of states, (s, s') that denotes a transition from state s to state s'. The primitive operation $shift(\sigma, z)$ shifts a sequence σ left by one, and appends the element z, where $z \in \Sigma$, to the end. For instance, if the sequence $\sigma = abc$, then $shift(\sigma, z) = shift(abc, z)$ which is equal to the new sequence bcz. The primitive operation $next(\sigma)$ returns the first symbol of the sequence σ, then left shifts σ by one to the next symbol. This function is analogous to popping the first element from the top of a stack, where the top of the stack is the beginning of the sequence. For example, given a sequence $abcde$, $next(abcde)$ returns a and updates the sequence to $bcde$.

The construction of the Markov model for normal behavior based on training data can be described as follows:

Initialize:

- $current\_state$ = first N elements of training data and,
- σ = entire training-data stream minus first N elements.

Until all the sequences of size N have been scanned from the training data:
1. Let $c = next(\sigma)$.
2. Set $next\_state$ to $shift(current\_state, c)$.
3. Increment counter for the state $current\_state$ and for the transition $(current\_state, next\_state)$.
4. Set $current\_state$ to be $next\_state$.

After the entire stream of training data has been processed, the probability of the transition is computed as $(s, s') = \frac{F(s,s')}{F(s)}$, where $F(s, s')$ and $F(s)$ are the counters associated with the transition (s, s') and s respectively.

Markov testing stage Let 0.00 indicate normal, and let 1.00 indicate anomalous. The surprise factor (sometimes called an anomaly signal) can be calculated from test data as follows:

Initialize:

- $current\_state$ = first N elements of training data and,
- σ = entire training-data stream minus first N elements.

Until all the sequences of size N have been scanned from the test data:
1. Let $c = next(\sigma)$.
2. Surprise factor = 1 minus the transition probability of $(current\_state, next\_state)$.
3. Set $current\_state$ to be $next\_state$.

7.1.2 Stide detector

Stide is a sequence, time-delay, embedding anomaly detector inspired by natural immune systems that distinguish self (normal) from nonself (anomalous) [18] [19]. The reference to "time" recognizes the time-series nature of the categorical data on which the detector is typically deployed. Stide has been applied to streams of system kernel-call data in which the manifestations of maliciously altered code are regarded as anomalies [20]. Stide mimics natural immune systems by constructing templates of "self" and then matching them against instances of "nonself." It achieves this in several stages.

Stide training stage A database consisting of templates of "self" is constructed from a stream of data considered to be normal (self); these are the training data. The stream is broken into contiguous, n-element, overlapping subsequences, or n-grams. The value of n is typically determined empirically [21]. Duplicate n-grams are removed from the collection, leaving only the unique ones. These unique n-grams are stored for future fast access. This completes the training stage.

Stide testing stage Stide compares n-grams from an unknown dataset (testing data) to each of its unique "self" n-grams. Any unknown n-gram that does not match a "self" n-gram is termed a mismatch.

Finally, a score is calculated on the basis of the number of n-gram comparisons made within a temporally localized region (termed "locality frame") [21]. Each comparison in step two (testing) receives a score of either zero or one. If the comparison is an exact match, the score is zero; if the comparison is not a match, the score is one. These scores are summed within a local region to obtain an anomaly signal. An example illustrates. Within a local region of 20 comparisons made between "self" and unknown, if all 20 are mismatches, the score will be 20 for that particular region; if only 8 are mismatches, the score will be 8 for that region. There are many overlapping regions of 20 in any given data stream. Stide calculates which of these 20-element regions has the highest score, and concludes that that region is the locus of the anomaly.

The Stide algorithm can be described formally as follows. Let N be the length of a sequence. The similarity between the sequence $X = (x_0, x_1, x_2, .., x_{N-1})$ and the sequence $Y = (y_0, y_1, y_2, .., y_{N-1})$, is defined by the function:

$$Sim(X,Y) = \begin{cases} 0 & \text{if } x_i = y_i \text{ for all } i, 0 \leq i \leq (N-1) \\ 1 & \text{otherwise} \end{cases}$$

The expression above states that the function $Sim(X,Y)$ returns 0 if two sequences of the same length are element-by-element identical; otherwise the function returns 1.

Each sequence of size N in the test data is compared to every sequence of size N in the normal database. Let $Norm$ be the number of sequences of size N in the normal database. Given the set of sequences in the normal database, $\{Y_0, Y_1, Y_2, .., Y_{(Norm-1)}\}$, for the ordered set of sequences $\{X_0, X_1, X_2, .., X_{Z-(N-1)}\}$ in the test data, where $X_s = (x_s, .., x_{s+(N-1)})$ for $0 \leq s \leq (Z-(N-1))$, and where Z is the number of elements in the data sample, the final similarity measure assigned the sequence X_s is

$$Sim_f(X_s) = \begin{cases} 1 & \text{if } Sim(X_s, Y_j) = 1 \text{ for all } j, \\ & 0 \leq j \leq (Norm-1) \\ 0 & \text{otherwise} \end{cases}$$

The expression above states that when a sequence, X_s, from the test data is compared against all sequences in the normal database, the function $Sim_f(X_s)$ returns 1 if no identical sequence can be found (i.e., a mismatch); otherwise the function returns 0 to indicate the presence of an identical sequence (a match) in the normal database.

The locality frame count (LFC) for each size N sequence in the test data is described as follows. Let L be the size of the locality frame and let Z be the number of elements in a data sample. For the ordered set of sequences $\{X_0, X_1, X_2, .., X_{Z-(N-1)}\}$, where $X_s = (x_s, .., x_{s+(N-1)})$ for $0 \leq s \leq (Z-(N-1))$, the LFC can be described by:

$$LFC(X_s) = \begin{cases} \sum_{l=((s-L)+1)}^{s} Sim_f(X_l) & \text{for } s \geq L \\ \sum_{l=0}^{s} Sim_f(X_l) & \text{for } s < L \end{cases}$$

7.2 Constructing the synthetic training data

The training data serve as the "normal" data into which anomalous events are injected. The requirements for the training data are that a large proportion of the data be comprised of common sequences, that they contain a small proportion of rare sequences, and that there is a relatively high predictability from one symbol to another. The common sequences are required to facilitate the creation of background test data that will contain no noise in the form of naturally-occurring rare or foreign sequences. This is necessary so that a detector's response to the injected anomaly can be observed without confounding by such phenomena. The rare sequences in the training data are needed so that anomalous events composed of rare sequences can be drawn from the normal training data, and then injected into the test data; see Section 7.3 below for details. Finally, a modicum of predictability is convenient for emulating certain classes of real-world data (e.g., system kernel calls) for which detectors like Stide are said to be well suited [22].

The alphabet has eight symbols: A B C D E F G and H. A larger alphabet could have been used, but it would not have demonstrated anything that an 8-symbol alphabet could not demonstrate; increasing the alphabet size would not change the outcome. Moreover, substantially more computation time is required as the alphabet size goes up. It is noted that alphabet sizes in real-world data are typically much larger than 8; for example, the number of unique kernel-call commands in BSM [23] audit data exceeds 200. However, the current goal is to evaluate a detector in terms of its ability to detect anomalies as higher-level abstract concepts, and while alphabet size does influence the size of the set of foreign sequences and the set of possible sequences that populate the normal dataset, foreign sequences and rare sequences retain their character irrespective of alphabet size. Maintaining a relatively small alphabet size facilitates a more manageable experiment, yet permits direct study of detector response.

To accommodate the requirements for predictability and data content, the training data were generated from an eight-by-eight state transition matrix with probability 0.9672 in one cell of each row, and 0.004686 in every other cell, resulting in a sequence of conditional entropy 0.1 (see [24] for details). One million data elements (symbols) were generated so that there would be a sufficient variety of rare sequences in the sample to use them in the construction of anomalous sequences for the test data. Ninety-eight percent of the training data consisted of repetitions of the sequence A B C D E F G H, seeding the data set with common sequences. This is the data set used to train the two detectors used in this study, i.e., to establish a model of normalcy against which unknown data can be compared.

7.3 Constructing the synthetic test data

Test data, containing injected anomalies, are used to determine how well the detector can capture anomalous events and correctly reject events that are not anomalous. The test data consist of two components: a background, into which anomalies are injected, and the anomalies themselves. Each is generated separately, after which the anomalies are injected into the background under strict experimental control. The background consisted of repeated sequences of A B C D E F G H, the most common sequence in the training data. This was done so that the test data would not conflict with the training data, i.e., would not contain spurious rare or foreign sequences.

7.4 Constructing the anomalous injections

Once the background data are available, anomalous events must be injected into them to finalize the test data. The anomalies must be chosen carefully so that when they are injected into the test data they do not introduce unintended anomalous perturbations, such as external boundary conditions. If this were to happen, then a detector could react to those conditions, confounding the outcomes of interest. Hence, scrupulous control is necessary.

The goal is to map the detection capability of both the Stide and the Markov anomaly detectors, and to show that their detection capabilities may vary with respect to identical and unequivocally anomalous phenomena. Given this objective, a single anomaly type that both detectors must be able to detect is selected from the anomaly space in Figure 2 for the experiments. The anomaly type selected is a foreign sequence of length AS for which all subsequences of length less than AS that make up the internal sequences and the boundary sequences are rare. Rare is defined to be any sequence of detector-window length that occurs in the training data less than one percent of the time.

It is within the scope of this study to map out only one region or type in the anomaly space in order to illustrate what can be learned and gained by the effort. Once that anomaly type is determined, e.g., FS RI RB, as described in Section 7, the next step is to inject a foreign sequence composed of rare sequences into the test data. A catalog of rare n-grams is obtained from the training data. Rare n-grams are drawn from the catalog, and composed to form a foreign sequence of the appropriate size. For example, the bigrams BA, AF, FH, HE, EC, CC and CF each occurred less than 0.06% of the time in the training data; consequently, these are rare bigrams. Combining these seven bigrams produces one octagram (BAFHECCF) whose internal sequences are made up of rare sequences of size two. This octagram was injected into the background data.

Once the composed foreign sequence is injected into the background data, its boundary conditions must be checked to ensure that they are all rare (because rare boundary conditions are consistent with the anomaly class being examined). If the boundary conditions are satisfied, the procedure is finished; if not, an attempt is made to handcraft the boundary conditions with the help of semiautomated tools. If the handcrafting fails, then a different set of rare sequences is selected from the catalog, and a new foreign sequence is composed. The new foreign sequence is injected into the background data, and its boundary conditions are checked. This entire procedure is repeated until a successful injection is obtained. Note that when the size of the detector window is greater than the size of the injection, an encompassing condition ensues, not an internal condition; however, care is still required to ensure that the external boundary conditions remain pertinent, even though the focus has been moved from internal conditions to encompassing conditions.

Eight injection sizes and fourteen detector-window sizes were tested. The procedure outlined above for creating the anomalous events, and for injecting them, is repeated for each combination of injection size and window size, resulting in 112 total data sets.

7.5 Scoring detector performance

Anomaly detectors are capable of only two kinds of decisions: yes, an anomaly exists; or, no, an anomaly does not exist. Detectors are usually judged in terms of hits, misses and false alarms. A hit occurs when the detector determines that an anomaly is present, and an anomaly actually is present. A miss occurs when the detector determines that no anomaly is present when actually there is one present. A false alarm occurs when the detector determines that there is an anomaly present when in fact there is no anomaly present. A perfect detector would have a 100% hit rate, no misses and no false alarms.

To effect proper scoring, ground truth (a statement of undisputed fact regarding the test data) must be known. That is, it must be determined exactly where anomalies have been injected into the test data, so that when the detector issues a decision, the correctness of that decision can always be assured. The injector creates a key that indicates the exact location of every injection. Using this key, one can determine whether or not a detector's decisions are correct. The usual procedure for this is for the detector to create an output file containing its decisions: 0 for no, and 1 for yes. This file can be compared against the key, which contains a similar set of zeroes and ones. If the two files match perfectly, the detector's performance is a perfect 100%; otherwise, the percent of hits, misses and false alarms can be calculated easily.

Using the injection key file, however, is not as straightforward as it might first appear. Due to the interaction of the detector-window size and the particular composition of the injected event, the detector's responses may not always be aligned perfectly with the ones and zeroes in the key file. For example, if the key file contains a one at the leading edge of an injected event (which seems sensible), the detector's response might not match. In fact, the detector might make a variety of responses, some of them multiple, to an injected event, and the key file must be constructed to facilitate correct scoring for any set of responses the detector might make. For example, the detector might decide, incorrectly, that an anomaly exists at the leading-edge boundary of an injected event; in fact, at that boundary there is no anomaly, so the detector's decision would be wrong. Another example is that the detector, depending on its window size relative to the size of the injected event, may respond only to subsequences internal to the injected event, but not to the event *in toto*. There is a danger that a detector will respond several times to a single injected event (because it may encounter several different views of that event), and thereby be judged mistakenly to have false-alarmed - to have decided *yes* in error.

The problems with matching detector decisions against ground-truth keys can be addressed in a variety of ways. In the present work the primary concern is to determine whether or not a detector is completely blind to an injected event, so the most worrisome response would be no response at all - a miss. If the detector does not decide *yes* to any part of the injected event, whether internal, whole, encompassing or boundary, then it is blind to the event. Alternatively, if the detector does respond, there needs to be a way to mitigate the problem of key mismatch, as described above. This problem is addressed by requiring that the detector respond positively at least once within the span of the injected event and the elements comprising its boundary conditions (collectively called the incident span) in order to have judged the event to be a hit.

7.6 Procedure

Each of the two detectors, Markov and Stide, was provided with the same set of training data. From the training data, the detectors learned their models of normal behavior. Then each detector was tested, using each of the 112 test data sets described in Section 7.4. The size of the detector window was varied from two to fifteen, and the size of the injected events was varied from two to nine. The restrictions on these dimensions were due to resource limitations, since computation time and memory increase with an increase in either dimension. Moreover, nothing new would be learned from raising either parameter. Each detector, for each testing session, produced a decision file which was compared to the key files for each test data set. Comparisons that revealed detector blindness (no detection of injected anomalous events) were charted.

Note that Stide's locality-frame-count feature was ignored, because it operates only as an amplifier for detected anomalies (which Stide represents as mismatches). If a foreign-sequence anomaly is not detected by a mismatch, then applying the locality frame count will not make the anomaly visible. Since the task at hand is to determine whether or not the injected anomalies were detected, amplification, which can be viewed as a post-detection process, is not relevant for mismatch/anomaly detection under present experimental conditions. As a consequence of this, Stide's maximum anomalous response is 1.

8 Results

Detection and blind regions for the Markov and Stide detectors are depicted in Figures 6 and 7 respectively. These decision maps illustrate the detection capability of both the Markov and the Stide detector with respect to an injected foreign sequence composed of rare sequences.

The x-axis of each map marks the increasing size of the foreign sequence injected into the test data; the y-axis marks the size of the detector window required to detect a foreign sequence of a specified size. Each star indicates successful detection of the foreign-sequence anomaly whose size is indicated on the x-axis, using a detector window whose size is marked on the x-axis; detection specifically means that at least one positive response occurred in the incident span, where "positive" connotes the most-anomalous value possible in the detector's range of responses. In both detectors, the most anomalous response is 1. The areas that are bereft of stars indicate either detection blindness or an undefined region. Detection blindness means that the detector was unable to detect the injected foreign sequence whose corresponding size is marked on the x-axis, i.e., the maximum anomalous response recorded along the entire incident span was 0 – to the detector, the anomaly appears as being completely normal. Note that no false alarms occurred, because background data were constructed from common sequences which do not raise alarms.

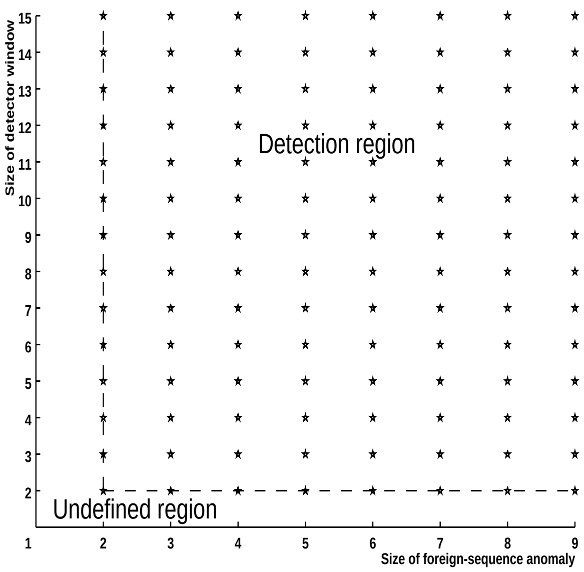

Figure 6: Markov sweet and blind regions.

The undefined region is an artifact of the Markov detector and anomaly type. Since the Markov detector is based on the Markov assumption, i.e., the next state is dependent only upon the current state, the smallest possible window size is 2, or a bigram. This means that the next expected, single, categorical element is dependent only on the current, single, categorical element. As a result, the y-axis marking the detector window sizes in Figure 6 begins at 2. Although it is possible to run Stide on a window size of 1, doing so would produce results that do not include sequential ordering of

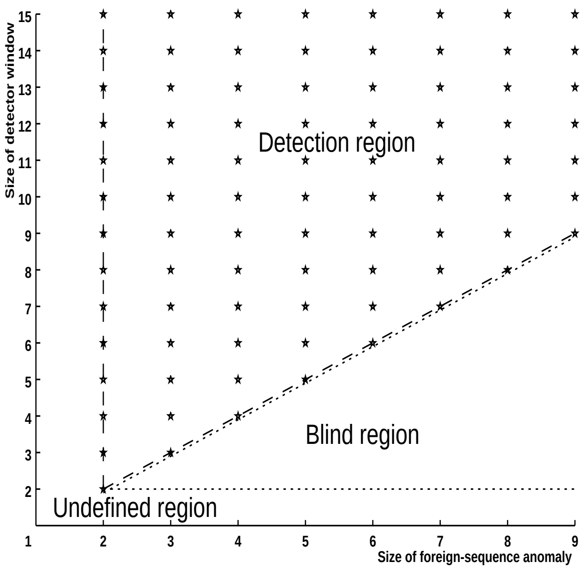

Figure 7: Stide sweet and blind regions.

events, a property that comes into play in all window sizes larger than 1. This, together with the fact that there is no equivalent window size of 1 on the side of the Markov detector, argued against running Stide with a window of 1.

The y-axis also begins at 2, because the type of anomalous event on which the detectors are being evaluated requires that a foreign sequence be composed of rare sequences. A foreign sequence of size one is an event that contains a single element that must be both foreign and rare at the same time; this is not possible. As a result, both Figures 6 and 7 show an undefined region corresponding to a detector window of size two and an anomaly of size one.

By charting the performance spaces of Stide and the Markov-based detector with respect to a foreign sequence composed of rare sequences, one is able to observe the nature of the gain achieved by employing the conditional probabilities of the Markov detector (that are absent in Stide). The significant gain in detection capability endowed by the use of conditional probabilities is illustrated by the blind region depicted in Figure 7. It is interesting to note that, for the exact same datasets, using a detector window of length 9, Stide's detection coverage is just 56% of Markov's. As the detector window size decreases to 2, Stide's coverage decreases to only 12.5% of that of the Markov detector. At this window size, however, the Markov detector still has 100% coverage of the space, which is a tremendous difference.

The results show that although the Markov and Stide detectors each use the concept of a sliding window, and are both expected to be able to detect foreign sequences, their differing similarity metrics significantly impact their detection capabilities. In the case of Stide, even though there is a foreign sequence present in the data stream, it is visible only if the size of the detector window is at least as large as the foreign sequence composed of rare subsequences – a requirement that the Markov detector does not have. Therefore, even if a fault does manifest as a foreign sequence in the data, it doesn't necessarily mean that Stide, which claims to be able to detect "unusual" sequences, will detect such a manifestation. It should be noted, therefore, that the selection of a similarity metric can have critical effects on the performance of a detector, and these choices should be made with care and with understanding of the metric.

9 Real-world data

The results shown in previous sections were based on synthetic data that were generated specifically to test the different anomaly detectors described. This section provides a link to real-world data, and shows that the manifestations of live anomalies in system kernel-call data are consistent with the anomaly-space map of Figure 2.

The live experiment consisted of a cyberattack on a RedHat 6.2 Linux system. The attack exploited a vulnerability in `glibc` (standard C library) through the `su` program (a program that switches a user from one account to another, given that the user provides appropriate credentials). The library allows a user to write an arbitrary format string. The exploiting program writes a carefully crafted string which interrupts the execution of `su`, allowing the user to run a script or program with root privileges. The exploit permits the user to switch accounts *without* providing credentials. Running `su` with and without the exploit should produce kernel-call data with and without anomalies due to the exploit itself. Kernel-call data on the victim machine were logged using the IMMSEC kernel patch, provided by the Computer

Immune Systems Research Group at the University of New Mexico.

The attack was scripted so that it could be repeated reliably and automatically. The following procedure was run three times, using standard `su` (with normal user interaction, providing credentials to switch from user to root) to obtain normal (training) data, and run three more times using the `su` exploit to obtain the anomalous (test) data:

1. Start a session as a regular user.
2. Turn on syscall logging for the `su` program.
3. Run the exploit as the regular user; verify that it was successful in giving the user a shell with root privileges.
4. Turn off syscall logging for the `su` program; move the log file to a permanent location.
5. Clean up the environment and log out.

The monitored kernel-call data were examined by both an experienced system programmer and a set of automated tools to find all the minimal foreign sequences that appeared in the attack data, but not in the normal data. A minimal foreign sequence is a foreign sequence in which no shorter foreign sequence is embedded. The programmer and tool results were compared and found to be mutually consistent. The system programmer confirmed, through systematic analysis, that all of the foreign sequences were direct manifestations of the attacks. Seventeen foreign-sequence anomalies were discovered in the `su` exploit; no foreign symbols were found, and there was no variability in the kernel-call data for the three attacks. The foreign-sequence anomalies ranged in length from 2 to 5, with one anomaly of length 5, five anomalies of length 3, and eleven anomalies of length 2. Some anomalies were unique; others were repeated in the data. Details are shown in Figure 8.

When using a detector window of size 2, which is the smallest possible size that covers an anomaly of length two, the real-world foreign-sequence anomalies in Figure 8 had compositional characteristics like the ones shown in the anomaly space in Figure 2. All of the anomaly descriptions are the same as the ones described in the anomaly space, except for the second one, which is eight characters instead of six or four, like the rest. The anomaly (FS RI RB FB) shown in the figure is from live data, and its composition reflects

Anomaly Size	Anomaly Contents	Anomaly Description
5	91,5,5,108,90	FS RI RB
3	4,5,5	FS RI RB AB
3	5,5,5	FS RI RB
3	5,5,5	FS RI RB
3	5,5,5	FS RI RB
3	5,5,5	FS RI RB
2	6,4	FS RB AB
2	4,4	FS – AB
2	4,4	FS – AB
2	4,4	FS – AB
2	4,4	FS – AB
2	4,4	FS – AB
2	4,4	FS – AB
2	4,4	FS – AB
2	5,4	FS RB AB
2	4,23	FS – AB
2	23,11	FS RB AB

Figure 8: Foreign-sequence anomalies, discovered in real-world information-warfare attack data, showing the size of each of the 17 anomalies, the events comprising each anomaly, and the anomaly description in accordance with the anomaly space of Figure 2. Anomaly contents are numerical encodings of kernel calls.

the broader set of conditions pertaining to uncontrolled, real-world data, as opposed to the more compact formulations in the anomaly space which are for well-behaved anomalies like the ones found in the synthetic data. The FS, as usual, indicates the base type of the anomaly: foreign sequence. The RI indicates that all of the internal conditions are rare. The RB indicates that all of the sequences comprising the left boundary are rare, and the AB indicates that all the sequences comprising the right-boundary are alien.

10 Discussion and conclusion

This paper has addressed fundamental issues in anomaly detection. The results are applicable in any domain in which anomaly detection in categorical data is conducted.

The paper has shown how to assess the coverage of an anomaly detector, and has also illustrated many subtleties involved in doing so. There are myriad factors to be considered carefully; the process is not straightforward. Meticulous attention needs to be paid to the interactions between an anomalous event and the normal environment in which it is embedded, i.e., external boundary conditions, internal conditions, encompassing conditions, and common, rare and foreign sequences that compose an anomalous event. Unless all of these factors are accounted for, error may be the biggest enemy of a correct mapping.

The coverage maps for two different anomaly detectors were shown to be strikingly different. This might come as a surprise to someone who believes that applying any anomaly detector to a stream of sensor data would be satisfactory, or that either of two detectors would detect the same events. One detector, Stide, which was specifically designed to detect foreign sequences, was shown to be blind to over half of a region it purports to cover. When used in its original role as a detector for information-warfare intrusions, Stide has been operated in a region of the detection space that is about six by six in terms of window size vs. anomaly size. It is interesting that in that region Stide is blind to 36% of the space, whereas the Markov detector covers 100% of that same region.

It is not necessarily bad for an anomaly detector to have less than perfect coverage, as long as the user knows the limitations of the detector. If a detector has suboptimal coverage, it may be possible to deploy the detector in situations where it doesn't need to operate in the part of the space in which it is blind. It will never be possible to assure this, however, if the detector's coverage is not mapped.

Can multiple anomaly detectors be composed to attain greater coverage than that achieved by a single anomaly detector used alone? It seems clear from the two maps produced here that one detector can be deployed to compensate for the deficiencies of another. In the present case it may appear that the Markov detector should simply replace Stide altogether, but because each detector has a different operational overhead, it may not be straightforward to determine the best mix for a compositional detection system. Also, one should be reminded that in the present work only one cell of the anomaly space depicted in Figure 2 has been examined; determination of overall coverage awaits examination of the rest of the cells as well. When deploying anomaly detectors in embedded or mission-critical systems, it is essential to understand the precise capabilities of the detectors, as well as the characteristics of the spaces in which they will operate.

Although the real-world experiment with live systems and data was limited in scope, it still illustrates two important things. First, the anomaly types depicted in Figure 2 were demonstrated to exist in real-world data; they are not mere artifacts of a contrived environment. Second, the response of a detector to a specified type of anomaly will not change, whether the anomaly is found in synthetic data or in real-world data; consequently, the results obtained from having evaluated an anomaly detector on synthetic data will be preserved faithfully when applied to real data; that is, predictions made with synthetic data will be sustained when transferred to real-world environments.

Some important lessons have been learned. A blind region in an anomaly-space map will always grow as the foreign sequence grows. This means that longer foreign sequences may constitute vulnerabilities for the detection algorithms considered here. Nevertheless, it is undoubtedly better to know the performance boundaries of a detector so that compensations can be made for whatever its weaknesses may be. Synthetic data have been effective in mapping anomaly spaces. Synthetic data may be the only avenue for creating such maps, because they permit running experiments in which all confounding conditions can be controlled, allowing absolute calibration of ground truth. Although real-world data is appealing for testing detection systems, real-world ground truth will always be difficult to obtain, and not all of the desired conditions for testing will occur in real data in a timely way. Finally, and most importantly, the anomaly-space framework provides a mechanism that bridges the gap between the synthetic and real worlds, allowing evaluation results to transfer to any domain through the anomaly-space abstraction.

11 Acknowledgements

The work herein was supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under contracts F30602-99-2-0537 and F30602-00-2-0528. Many other people contributed in various ways; the authors are grateful to Kevin Killourhy, Pat Loring, Bob Olszewski, Sami Saydjari and Tahlia Townsend for their help. This paper draws on Kymie Tan's forthcoming dissertation [25].

References

[1] Stephen Cass, "Little Linuxes," *IEEE Spectrum*, vol. 38, no. 3, pp. 23–25, March 2001.

[2] Peter A. Lee and Tom Anderson, *Fault Tolerance: Principles and Practice*, Springer–Verlag, Vienna, Austria, second edition, 1990.

[3] Brian Randell, "System structure for software fault tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, no. 2, pp. 220–232, June 1975.

[4] Roy A. Maxion and Frank E. Feather, "A case study of ethernet anomalies in a distributed computing environment," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 433–443, October 1990.

[5] Michael M. Tsao, *Trend Analysis and Fault Prediction*, Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, May 1983.

[6] Roy A. Maxion and Daniel P. Siewiorek, "Symptom based diagnosis," in *IEEE International Conference on Computer Design (ICCD-85)*, 1985, pp. 294–297, 07-10 October, Port Chester, NY.

[7] Michael F. Buckley, *Computer Event Monitoring and Analysis*, Ph.D. thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, Pittsburgh, PA, May 1992.

[8] Teresa F. Lunt, "A survey of intrusion-detection techniques," *Computers & Security*, vol. 12, no. 4, pp. 405–418, June 1993.

[9] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, June 1946.

[10] Beth A. Schroeder, "On-line monitoring: A tutorial," *IEEE Computer*, vol. 28, no. 6, pp. 72–78, June 1995.

[11] Thomas F. Arnold, "The concept of coverage and its effect on the reliability model of a repairable system," *IEEE Transactions on Computers*, vol. C-22, no. 3, pp. 251–254, March 1973.

[12] Daniel P. Siewiorek and Robert S. Swartz, *Reliable Computer Systems*, Digital Press, Burlington, MA, second edition, 1992.

[13] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer, "Fault injection techniques and tools," *IEEE Computer*, vol. 30, no. 4, pp. 75–82, April 1997.

[14] Jeffrey A. Clark and Dhiraj K. Pradhan, "Fault injection - a method for validating computer-system dependability," *IEEE Computer*, vol. 28, no. 6, pp. 47–56, June 1995.

[15] David R. Cox and Hilton D. Miller, *The Theory of Stochastic Processes*, Wiley, New York, 1965.

[16] James D. Hamilton, *Time Series Analysis*, Princeton University Press, Princeton, New Jersey, 1994.

[17] Somesh Jha, Kymie M. C. Tan, and Roy A. Maxion, "Markov chains, classifiers, and intrusion detection," in *14th IEEE Computer Security Foundations Workshop*, Los Alamitos, California, 2001, pp. 206–219, IEEE Computer Society Press, 11-13 June, Cape Breton, Novia Scotia, Canada.

[18] Stephanie Forrest, Steven A. Hofmeyer, and Anil Somayaji, "Computer immunology," *Communications of the ACM*, vol. 40, no. 10, pp. 88–96, October 1997.

[19] Stephanie Forrest, Steven A. Hofmeyer, Anil Somayaji, and Thomas A. Longstaff, "A sense of self for unix processes," in *IEEE Symposium on Security and Privacy*, Los Alamitos, CA, 1996, pp. 120–128, IEEE Computer Society Press, 06-08 May, Oakland, CA.

[20] Steven Hofmeyr, Stephanie Forrest, and Anil Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[21] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *1999 IEEE Symposium on Security and Privacy*, Los Alamitos, CA, 1999, pp. 133–145, IEEE Computer Society Press, 09-12 May, Oakland, CA.

[22] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri, "Self-nonself discrimination in a computer," in *IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA, 1994, pp. 202–212, IEEE Computer Society Press, 16-18 May, Oakland, CA.

[23] Sun Microsystems, "Sunshield basic security module guide," Technical report 805-2635-10, Sun Microsystems, Inc., Palo Alto, California, October 1998.

[24] Roy A. Maxion and Kymie M. C. Tan, "Benchmarking anomaly-based detection systems," in *International Conference on Dependable Systems and Networks*, Los Alamitos, California, 2001, pp. 623 – 630, IEEE Computer Society Press, 25-28 June, New York, New York.

[25] Kymie M. C. Tan, *Defining the operational limits of anomaly-based intrusion detectors*, Ph.D. thesis, Melbourne University, Department of Computer Science, Melbourne, Victoria, Australia, 2001.