A Machine Learning Evaluation of an Artificial Immune System

Matthew Glickman glickman@cs.unm.edu
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386, USA

Justin Balthrop judd@cs.unm.edu
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386, USA

Stephanie Forrest forrest@cs.unm.edu
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386, USA and

Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA

Abstract

ARTIS is an artificial immune system framework which contains several adaptive mechanisms. LISYS is a version of ARTIS specialized for the problem of network intrusion detection. The adaptive mechanisms of LISYS are characterized in terms of their machine-learning counterparts, and a series of experiments is described, each of which isolates a different mechanism of LISYS and studies its contribution to the system's overall performance. The experiments were conducted on a new data set, which is more recent and realistic than earlier data sets. The network intrusion detection problem is challenging because it requires one-class learning in an on-line setting with concept drift. The experiments confirm earlier experimental results with LISYS, and they study in detail how LISYS achieves success on the new data set.

Keywords

Anomaly detection, artificial immune systems, machine learning, immune system, network intrusion detection, computer security.

1 Introduction

The natural immune system uses a variety of evolutionary and adaptive mechanisms to protect organisms from foreign pathogens and misbehaving cells in the body. Artificial immune systems (AIS) seek to capture some aspects of the natural immune system in a computational framework, either for the purpose of modeling the natural immune system or for solving engineering problems. In either form, a fundamental problem solved by most AIS can be thought of as learning to discriminate between "self" (the normally occurring patterns in the system being protected, e.g., the body) and "non-self" (foreign pathogens, such as bacteria or viruses, or components of self that are no longer functioning normally). Almost any set of patterns that can be expressed as strings of symbols can be placed into this framework, for example, the set of normally occurring TCP connections in a local area network (LAN) and the set of TCP connections observed during a network attack. This is the example on which we will focus in this paper.

Hofmeyr introduced an artificial immune system framework called *ARTIS*, which he specialized for the problem of network intrusion detection in a system known as *LISYS* (Lightweight Intrusion detection SYStem) (Hofmeyr, 1999; Hofmeyr and Forrest, 2000). In that work, he focused on explaining the analogy between real immunology and the LISYS artifact and reported early experiments demonstrating how well the system performed.

Hofmeyr's results from this immune-inspired architecture were encouraging, and there are several related projects in in network intrusion detection based on AIS (Dasgupta, 1999; Williams et al., 2001; Kim and Bentley, 1999a). However, LISYS is a complex architecture, similar in complexity to learning classifier systems (Holland et al., 1986), and the task of understanding and predicting its behavior is challenging. The research reported in this paper is focused up three complementary goals: (1) to connect LISYS with the broader context of Machine Learning, (2) to further the empirical evaluation of LISYS' performance, and (3) to deepen understanding of the contributions made by LISYS' many components to its overall performance.

Machine Learning: LISYS sends an alert to an operator when it detects anomalous network packets, potentially indicating intrusion attempts. LISYS' detection mechanism does not rely on preprogrammed knowledge. Rather, it observes an initial sample of network traffic, builds a model of normal traffic patterns, and then compares the model against subsequent traffic. Because new connections frequently occur on a LAN, LISYS must generalize from previously observed traffic to assess whether newly observed packets are anomalous. This capacity—that of an artificial system to cope with novelty via generalization from experience—is the domain of Machine Learning (ML).

Examining LISYS in the context of machine learning is important for emerging fields such as artificial immune systems, so that a common language can be identified and results communicated between fields. Here, we are interested in discovering what might be novel about LISYS from an ML point of view, and in what insights ML can give us about LISYS. In particular, one objective of this paper is to isolate the LISYS mechanisms so that they could potentially be used in other more traditional ML frameworks.

LISYS introduces several novel learning mechanisms, and we show that some of them improve performance dramatically. At its core, LISYS resembles a simple memory-based (also called "instance-based") learning algorithm. We examine each of LISYS' mechanisms as extensions to this simple learning algorithm, and assess how each mechanism addresses issues of interest in the more general machine learning community and computer security domain.

Evaluation: A preliminary study was conducted in the context of a small and well controlled data set (Balthrop et al., 2002b; Balthrop et al., 2002a), which focused on a limited number of LISYS mechanisms. Here, we extend that work to a more realistic and larger data set, and and we consider the complete LISYS architecture.

Our new data set represents data collected over a 62-day period from a subnet behind a firewall with 17 active machines. The scale of this network and its placement behind a masquerading firewall are typical of small enterprise networks. This setting, which is more controlled than the original data collected by Hofmeyr, allows us to isolate specific network events which affect LISYS performance and to be more certain that our core training data are free of attacks

Experiments assessing individual mechanisms: Although the essential idea of LISYS is quite simple, there are a variety of mechanisms incorporated into the full sys-

tem, each of which was originally inspired by some aspect of the biological immune system. The contributions to overall system performance made by each mechanism have until now been unclear. We report a number of new experiments designed to assess the relative contribution of LISYS' many components.

We begin by comparing the performance of LISYS with that of a "null algorithm" that performs no generalization from experience. We then conduct experiments with a set of variant algorithms covering the spectrum between the null algorithm and LISYS. In some cases, we incrementally add individual features into the null algorithm, while in others we subtract features from full LISYS (known as ablation experiments).

As an ML algorithm, the performance of LISYS reflects the validity of its assumptions about inference in its domain of application. Thus, the aggregation and ablation experiments reported here reveal the validity of the assumptions represented by each of LISYS' individual mechanisms in the given network intrusion-detection domain.

The remainder of the paper is organized as follows. Section 2 gives some background material on intrusion-detection systems, machine learning used for intrusion detection, LISYS, and other immunologically inspired approaches to intrusion detection. Section 3 examines LISYS from the perspective of machine learning; Section 4 describes the data set used for our experiments, and section 5 presents the experiments themselves. In section 6, we discuss the experimental results, LISYS' current limitations, and ideas for extension and improvement. We summarize our conclusions in section 7.

2 Background

Several areas of research are relevant to the work reported here. In this section, we briefly review earlier work on intrusion detection, discuss machine-learning approaches to this application domain, give an overview of LISYS, and review other AIS approaches to network intrusion-detection.

2.1 Intrusion Detection

Intrusion detection systems (IDS) vary widely, but they all seek to protect an information system (e.g., a single computer, a database, a server, or a network of computers) from violations of the system's security policy. Debar et al. (Debar et al., 1999) defined a taxonomy of IDS, distinguishing between behavior-based (often called anomaly-based) and knowledge-based (also known as signature) IDS. LISYS is an example of an anomaly IDS in which a model is constructed of the normal (legal) operation of the system and discrepancies from the model are labeled anomalous. The model of normal behavior can be based on any observable behavior of the system. Audit logs, patterns of network traffic, user commands, and system-calls are all common choices. Such an approach can work well if the anomalies in normal operation are correlated with security violations. The extent to which this is true is a topic of debate in the intrusion-detection community.

A second relevant distinction is whether an IDS defines the set of allowable behaviors (known as positive detection) or the set of disallowed behaviors (known as negative detection) (Forrest et al., 1994; Esponda et al., 2004). Because knowledge-based systems use knowledge of specific attacks or classes of attacks to identify intrusion events, we refer to them as negative-detection schemes. The knowledge can be represented many different ways. One common approach is to define a set of rules in a production system which is used to deduce the state of the monitored system. Examples of this approach include EMERALD (Porras and Neumann, 1997) and NSM (Heberlein

et al., 1990; Mukherjee et al., 1994). Alternatively, knowledge of attacks is often compiled into simple "signatures" or syntactic patterns which can be readily recognized in a data stream. Signature detection is computationally efficient, and thus forms the basis of many commercial products, for example, ISS' RealSecure Network Sensor (ISS, 2000).

Behavior-based systems typically treat intrusions as deviations from a profile of normal behavior (i.e. they employ a positive detection scheme). The most common way to obtain such a profile is by constructing a statistical model based on observation. Examples of this approach include HAYSTACK (Smaha, 1988), NIDES (Anderson et al., 1995), Helman and Liepins (Helman and Liepins, 1993), and NSM (Heberlein et al., 1990; Mukherjee et al., 1994). Lane (Lane, 2000) used both instance-based models and hidden Markov models to model normal sequences of commands used by specific users. A normal profile can also be prespecified rather than constructed empirically through observation, as in AT&T's ComputerWatch (Dowell and Ramstedt, 1990).

LISYS is best characterized as a negative-detection, behavior-based system. However, the secondary response of LISYS implements a form of signature-based detection (see Section 2.4.4). Following the taxonomy of Debar et al., LISYS can be further classified as passive in terms of counter-measures, net-based (packet-level) rather than host-based; and as a passive observer (rather than actively generating queries).

2.1.1 Evaluating intrusion-detection systems

IDS performance can be evaluated in several ways, including efficiency, training time, ease of use, ability to adjust to changing conditions, and accuracy of discrimination between allowed and disallowed behaviors(NIST, 2003; Gaffney and Ulvila, 2001; McHugh, 2000). Here, we focus on the accuracy criterion.

There are two kinds of errors a detection system can make: false negatives (mistakenly identifying abnormal patterns as legitimate) and false positives (mistakenly identifying normal patterns as abnormal). These are known as Type I and Type II errors respectively in the statistical decision theory literature. It is trivial to minimize either Type I or Type II errors, e.g., by labeling everything as anomalous or everything as normal. Therefore, any measure of IDS performance needs to take both types of errors into account. The tradeoff between false- and true-positives is often displayed using the "Receiver Operating Characteristics" (ROC) curve (Egan, 1975).

An important property of an IDS is tunability, that is, how easy it is to choose where on a given ROC curve a particular IDS will be deployed. This property is important because in some applications (e.g., a secure military installation), it is so important to catch all true positives that a high rate of false positives can be tolerated, while in other applications (say a university department) the inconvenience and cost of managing false positives outweighs the risk of a security breach. Although these tradeoffs are important, IDSs have historically suffered unacceptably high false-positive rates, and much work in IDS focuses on the problem of reducing false positives without unduly compromising true-positive detection ability (Axelsson, 2000).

There are several factors which complicate the evaluation and comparison of anomaly IDSs: The asymmetry between normal and intrusive data sets, differences in the amount of training data needed for different learning systems, and inconsistencies in how different IDS systems define false- and true-positives. In most data sets, normal data are plentiful and intrusion data are rare. As a consequence, it can be difficult to estimate the sensitivity of an IDS, and there are inherent asymmetries in the commonly reported ROC curves (e.g., strong discontinuities on the true-positive axis).

A related complication is that true positives are often measured differently from false positives. To detect an intrusion, we require only that the anomaly signal exceed a preset threshold at one point during the intrusion event. However, in an on-line system, low false-positive rates can be obtained only if the system makes many correct decisions over the entire sample of normal behavior. Another complication arises because different methods require different amounts of normal data in order to construct an accurate model. Thus, a straight comparison of two different IDS methods on identical data sets is not always fair. Different anomaly-detection methods record anomalies differently. For example, a system that records anomalies on a per-packet basis will likely achieve different false- and true-positive rates than one which records anomalies on a session basis.

A final consideration in evaluating the accuracy of an IDS is what role a detection event plays in the overall system. In some "miss-nothing-pyramid" security architectures, all detection events are passed upstream to other decision processes which evaluate the likelihood that an event is a false or true positive, e.g. (Porras and Neumann, 1997; Balasubramaniyan et al., 1998). In these settings, a relatively high false-positive rate can be tolerated at the IDS level. If, on the other hand, the IDS is deployed in a non-hierarchical collection of mostly autonomous systems, then false positives are additive (and therefore costly), but false negatives are not so important, because another process might catch it. This is the situation with LISYS, and consequently, we emphasize the analysis of false-positive rates in the following sections. This appears to be the more biological scenario, where autoimmunity is costly and there are many redundant detection mechanisms in the body. This scenario also seems more practical for computer systems that are unprotected now, where many intrusions succeed already and there are few resources available for sifting through false positives.

2.2 Machine Learning and Network Intrusion Detection

The LISYS system learns to classify network connections (SYN and SYN_ACK packets) as either normal or anomalous. In this section we discuss the network intrusion-detection problem from a machine learning perspective. In Section 3 we use this perspective to analyze LISYS as a machine learning algorithm.

Within the domain of machine learning, learning such a binary classification task typically falls under the heading of *supervised learning*, although LISYS receives external feedback only intermittently. In supervised learning, the goal is to learn a mapping from inputs to outputs. For example, the inputs might be visual data from a camera and the outputs might be appropriate commands to a robot motion system. If LISYS were a classical supervised learner, it would learn from a set of training data made up of both normal and anomalous packet traffic. Each training example would pair a specific input value, the packet, with its associated target output value or *label*, i.e. *normal* or *anomalous*. After training, the system would be presented with new inputs (which might or might not have been observed during training) and asked to infer the target output values.

To infer appropriate outputs for input values not presented during training, a system must incorporate an *inductive bias* (Mitchell, 1980), i.e. some assumption about the mapping that is to be learned: "a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances." [(Mitchell, 1997), p. 42]. The need for inductive bias arises from the fact that there are many possible mappings from inputs to outputs (labels) that are consistent with the observed training examples, and some set of assumptions or bias is necessary

to choose among them.

In a binary classification problem, the range of possible mappings or target concepts consistent with the observed training examples is bounded by a maximally specific choice at one extreme and by a maximally general one at the other. In the example of LISYS as a classical supervised learner, the maximally specific choice would be to assume that normal traffic is limited only to the normal packets observed during training, and that all other packets—including all those not observed during training–are anomalous. The maximally general choice, on the other hand, would be to treat all packets as normal, excepting those observed during training that were explicitly labeled as anomalous.

In most classification problems, it is assumed that the learning system will be provided with at least one input value in each target class during the training period. In the intrusion-detection problem, however, we lack representatives of all the possible attacks (because new attacks are constantly being devised), and there is a risk of biasing the system against detecting novel attacks if we rely on the corpus of current attacks for training. In this case, it may be preferable to use only observations of normal packets (positive examples of the target concept) to infer the target class. This is known as *one-class learning* and is the approach used by LISYS. Although LISYS classifies packets into two classes, it assumes during learning that all exemplars are normal, and it must infer the anomalous class without examples. This represents a deviation from classical supervised learning.

One-class learning is an open problem in machine learning research. Learning in the absence of training examples from one class leaves the range of target concepts consistent with the data much more open, making the job of the learner more difficult. In the case of LISYS, seeing only normal packets leaves the range of concepts consistent with the data unbounded up to the maximally general hypothesis, i.e. the hypothesis that all packets are normal. In the absence of examples from one class in a binary classification problem, an algorithm such as back-propagation of error (Rumelhart et al., 1987) applied to artificial neural networks will often gravitate toward such extreme hypotheses. In general, the expanded range of possible target concepts that arises with one-class learning serves to amplify the importance of the proper inductive bias.

Language inference is an area of machine learning in which has studied one-class learning. Gold (Gold, 1967) showed that not even the restricted class of regular languages can be learned in the limit from an arbitrary, finite-length sequence of positive examples. Nonetheless, children are observed to learn language based almost entirely on positive examples. This testifies to the quality of their inductive bias (Pinker, 1984), a bias developed through evolutionary time.

Angluin (Angluin, 1980) later showed that learning from positive examples is possible within restricted classes of target concepts ("pattern languages" in Angluin's example). More recently, Muggleton (Muggleton, 1996) proposed a Bayesian framework within which many concept classes can be learned from positive examples with arbitrarily low expected error. Lane (Lane, 2000) addressed the problem of one-class learning in IDS. He developed a method in which the user adjusts the generality of the learned concept through a specified threshold value.

A notion related to one-class learning is that of *outlier detection* in statistics. Here, the goal is to build a description or model of data generated by some process. The point of outlier detection is to exclude from the model data points that are overly affected by phenomena other than the process being studied. Most often, such points are artifacts of the data collection process, e.g. occasional spurious points generated by a

faulty piece of equipment. Thus, the goal of outlier detection is similar to that of intrusion detection in that we seek to identify anomalies from unknown sources. However, in intrusion detection the goal of building the model—characterization of normal—is specifically to detect "outliers," whereas in outlier detection the point is to prevent contamination of a model designed for other purposes.

Another challenge of the intrusion-detection domain is that normal behavior often changes over time, known in machine learning as *concept drift*. In the presence of concept drift some amount of invariance in the target concept must be assumed; if the target concept can change arbitrarily quickly, then there would be no rational basis for training. Consequently, some assumptions need to be made about the rate at which a concept can drift. Mitchell et al. (Mitchell et al., 1994) take a practical approach, empirically determining the length of a "window" of past training examples to be used for periodic retraining. A more theoretical approach, taken by Helmbold and Long (Helmbold and Long, 1991), derives an upper bound for the rate of drift that can be tolerated by any learner observing a given window size of past examples. Klinkenberg and Joachims (Klinkenberg and Joachims, 2000) proposed a method for support vector machines that periodically adjusts its window size using analytical means to estimate what window size is most likely to maximize generalization performance. Lane (Lane, 2000) addressed concept drift by training only on normal examples that are near the abnormal boundary. LISYS' approach, described below, is known as "detector turnover," in which old detectors are deleted and replaced by newly trained detectors. The detector set at any given time is composed of detectors trained upon many fixed-size windows of varying age. The system's assumption about the rate of drift is thus parameterized by a single number. This number determines the probability of any given detector being recycled during any single time-step.

2.3 Other AIS Approaches to Network IDS

Kim and Bentley describe an architecture for an AIS-based network IDS and report the results of experiments conducted with some of the overall system's key components (Kim and Bentley, 1999a; Kim and Bentley, 1999b; Kim and Bentley, 2001). As in LISYS, anomalies are detected via a repertoire of negative detectors distributed among multiple hosts. Unlike LISYS, detector generation is centralized at a single node (more closely resembling the natural immune system), and the decision about whether to signal an anomaly relies on communication between nodes. Also similar to LISYS is the choice of monitoring TCP connections, although the Kim and Bentley system monitors all packets in a connection, rather than only SYN packets. This gives the system more potentially relevant data, including temporal information such as packet rate within a connection. However, the costs include preprocessing (aggregating packets into connections and clustering connections into "profiles") as well as an extensive encoding/decoding process for mapping feature values into dynamic categories. Kim and Bentley's detector-generation process is more complicated than that of LISYS, involving a genetic algorithm, niching, and feedback from matched detectors to boost the diversity and utility of detectors. Later work (Kim and Bentley, 2002a; Kim and Bentley, 2002b) introduces mechanisms for adapting to changing self sets, managing memory detectors, and using old memory detectors to seed the immature detector population.

A second example of an immune-inspired network IDS, known as CDIS (Computer Defense Immune System), is reported by Williams et al (Williams et al., 2001). Like LISYS, CDIS processes the data-stream at the packet level (rather than at the level of connections). CDIS extends LISYS by monitoring packets for three different pro-

tocols: TCP, UDP, and ICMP. Accordingly, the precise set of fields used in the packet representation employed by CDIS varies depending upon the protocol to which it pertains. In general, the number of fields represented is significantly larger than that used by LISYS (28 features for TCP, and 16 features for UDP and ICMP).

Detectors in CDIS are explicit templates for packets. As in LISYS, immature detectors are initially generated randomly and then filtered using negative selection. Subsequent to negative selection, however, detectors undergo *affinity maturation* in which a genetic algorithm is used to evolve detectors that cover as many potential packets as is possible without matching self (this process takes place off-line, prior to engaging the detection process). Another unique feature of CDIS is co-stimulation between detectors rather than via interaction with a human operator. Recently (Anchor et al., 2002), CDIS has been extended to use more sophisticated detectors in the form of finite state transducers that are initially trained using an evolutionary algorithm.

Dasgupta (Dasgupta, 1999) described an immune-inspired architecture for general intrusion detection. The system is composed of multiple, mobile agents which are specialized for functions such as monitoring system state and responding to intrusions, along with agents specialized to help other agents to communicate. A goal of the system is to synthesize information from multiple domains, such as the system level, the user level, and the network level. When an intrusion is detected, the system may undertake any number of actions, including blocking a particular IP address, changing users' action privileges and/or informing system administrators.

Dasgupta and Gonzales (Dasgupta and Gonzalez, 2002) described an immunologically inspired approach to detecting anomalies in network traffic by examining the volume of various types of network traffic over time. Successive time windows of traffic data are encoded as vectors of real-valued traits, and negative detectors in the form of hyper-rectangles are evolved to cover as much of non-self space as possible. Later work (Gomez et al., 2003) extends this approach to use fuzzy logic-based detectors.

2.4 LISYS

LISYS monitors traffic on a local-area network (LAN). At its core, LISYS consists of a set of *detectors*, which are analogous to lymphocytes in the immune system. The detectors are continually compared against network packets. When one of the comparisons results in a *match* between a detector and event, the match is interpreted as an anomaly, potentially signaling a network-based attack.

The current implementation of LISYS monitors TCP connections, specifically the two initial SYN packets in a TCP connection (an initial SYN packet followed by a SYN/ACK packet in response). Following the immune system analogy, *self* is the set of normal pairwise TCP/IP connections between computers, and *non-self* is the set of connections, potentially an enormous number, which are not normally observed on the LAN. A connection can occur between any two computers in the LAN as well as between a computer in the LAN and an external computer. A connection is defined in terms of its "data-path triple"—the source IP address, the destination IP address, and the port by which the computers communicate (Mukherjee et al., 1994; Heberlein et al., 1990). This definition of self is quite restricted, and alternative definitions are certainly possible in this domain, as discussed in section 6.

The data path triple is compressed in two ways to form a single 49-bit string (Hofmeyr, 1999), shown in figure 1. First, it is assumed that one of the IP addresses is always internal, so only the final byte of this address needs to be stored. The port number is also compressed from 16 bits to 8 bits by re-mapping the ports into several

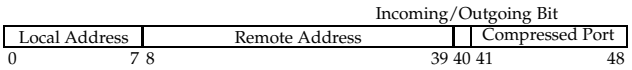

Figure 1: The LISYS 49-bit SYN packet compression scheme.

Figure 2: Using the r-contiguous bits matching rule with $r = 4$, the two strings on the left constitute a match. The two on the right, with a maximum of 3 matching contiguous bits, do not match.

different classes. 67 commonly assigned privileged ports are each allocated their own, unique ID number (from 0 to 66), while all other ports are each assigned to one of three classes: all other privileged ports (ID number 67), commonly assigned non-privileged ports (ports 6000-6063, ID 69), and all other non-privileged ports (ID 68).

The detectors are also 49-bit strings, along with a small amount of local state. A perfect match between a detector and a bit string representing a TCP connection means that at each location in the 49-bit string, the symbols are identical. However, LISYS uses *partial-match detection* so that it can generalize from its sample of observed normal traffic. LISYS uses a matching rule known as r-contiguous bits matching in which two strings match if they are identical in at least r contiguous locations, illustrated in figure 2 (Percus et al., 1993).

2.4.1 The Detector Lifecycle

Detectors in LISYS undergo a multi-stage lifecycle with the following stages: immature (analogous to immature lymphocytes undergoing negative selection in the thymus), mature (analogous to naive B-cells that have never been activated), activated (and awaiting co-stimulation), memory, and death. This lifecycle is illustrated in figure 3.

Detectors in LISYS are created randomly and immediately enter the immature phase where they remain for the *tolerization period*. The tolerization period lasts either for a fixed number of observed packets or a specific length of time, depending on a system parameter. If the detector matches a TCP connection during this phase, it is discarded and a new, random detector is generated in its place. An immature detector that fails to match any connections during its tolerization period becomes *mature*. This process, known as *negative selection*, is patterned after the tolerization process that occurs in the immune system (Forrest et al., 1994). A detector that fails to match any TCP connections during its tolerization period is presumed unlikely to misidentify normal packets as anomalous (that is, it has learned to tolerate normal traffic).

Detectors that survive negative selection and become mature have several possible fates. All mature detectors have a fixed probability of dying randomly on each time step. The finite lifetime of detectors, when combined with detector re-generation and tolerization, results in *rolling coverage* of the self set analogous to B-cell turnover in the body. Thus, a mature detector that fails to become activated eventually dies. Alternatively, a detector that matches a sufficient number of connections during its

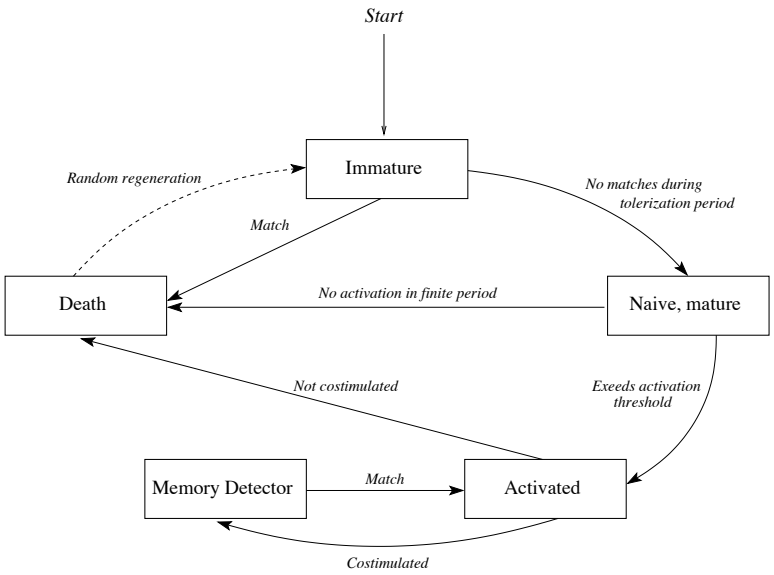

Figure 3: The LISYS detector lifecycle.

mature phase becomes activated. Unlike negative selection, where a single match is sufficient to kill the detector, a mature detector must match several connections before it becomes activated (see Section 2.4.3). Once a detector is activated, it must receive *co-stimulation* within a fixed period of time (see Section 2.4.4). A detector which receives co-stimulation becomes a memory detector, and a detector that fails to receive co-stimulation dies, analogous to second signaling systems in the immune system.

2.4.2 Detector Sets

Detectors in LISYS are typically grouped into autonomous detector sets, distributed among separate computers. This has several potential advantages. First, there is no central point of failure, as each node can operate independently. Second, individual nodes do not need to communicate among themselves or with a higher level controller in order to make a decision about an individual packet, potentially improving per-node efficiency. Distributing detectors among multiple nodes allows us to trade off coverage per node against the computational overhead per node. Finally, each node uses its own unique representation, allowing diversity of detection abilities throughout the system.

Diversity is achieved because each detector set uses a slightly different representation for its detectors, known as a secondary representation. Secondary representations approximate MHC diversity in the natural immune system. Secondary representations, combined with r-contiguous bits matching, allow each node to generalize differently from observed network traffic. As a result, some packets considered normal by one host will be flagged as anomalous by another. The overall effect of multiple secondary representations is to increase LISYS' discrimination abilities, decreasing the aggregate generalization of the entire system. The selection or adaptation of secondary representations to yield a more favorable inductive bias is the subject of ongoing research.

The secondary representation is a remapping applied to strings encoded in the base representation shown in figure 1. Although secondary representations may be generated many different ways, the experiments reported here follow (Hofmeyr, 1999),

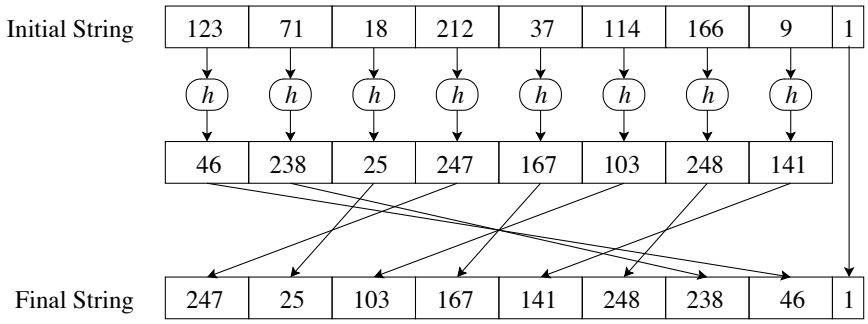

Figure 4: A substring-hash has two components: A random hash function, h, which maps each value from $[0..255]$ to another, unique value in $[0..255]$, and a random permutation of 6 elements. The 49-bit string is divided into 6 bytes, with one remaining bit. Each of the 6 bytes is remapped using h. The 6 bytes are then permuted, and the 49th bit is passed through unchanged.

assigning each host a unique randomly generated *substring hash* (see figure 4) to the base representation.

2.4.3 Activation Thresholds and Sensitivity Levels

Each detector records the number of packets it matches (the *match count*). A detector is activated when the number of matches exceeds the *activation threshold*. This mechanism also has a time horizon: Over time, in the absence of further matches, the match count probabilistically decays to zero.[1] Thus, only repeated occurrences of structurally similar and temporally clustered packets trigger the detection system. Activation thresholds are loosely analogous to lymphocyte activation in the immune system, which requires a sufficient number of receptors to be bound sufficiently tightly to become activated.

A related mechanism, *sensitivity levels*, crudely approximates the local sensitization signals used in the immune system. There is one sensitivity level for each detector set. Whenever the match count of a detector goes from zero to one, that detector set's sensitivity level is incremented. The higher the sensitivity level, the lower the threshold for activation: A node's effective activation threshold is equal to the nominal threshold value minus the sensitivity level, i.e. $\tau_{effective} = \tau - \Omega$, where τ is the activation threshold and Ω is the sensitivity level[2]. Sensitivity levels decay probabilistically toward zero.

2.4.4 Co-Stimulation and Memory Detectors

External feedback is supplied to LISYS by the co-stimulation signal, similar to co-stimulation in immunology. When a detector becomes activated, a message is sent automatically to a human operator. The operator has the option of providing confirmation that the detector has correctly identified an anomalous event of interest. If no confirmation is received (i.e. there is no "co-stimulation"), the activated detector is presumed

[1] In LISYS' original specification, when a detector raised an alarm its match count was reset to zero.

[2] Sensitivity levels in LISYS were originally capped to prevent the effective activation threshold from falling to zero or lower. Although this restriction is removed in the current implementation of LISYS, the fact that match counts are not reset to zero after signaling an alarm—and are therefore very unlikely to go from zero to one repeatedly within a short interval—prevents sensitivity levels from rising to high values in practice.

to be autoreactive and is deleted. If a detector does receive co-stimulation, the detector enters the memory state. Memory detectors are LISYS' analog of memory cells in the immune system. They are long-lived (no longer subject to random detector death), and they are more easily activated than mature detectors detectors (implemented with a lower activation threshold, $\tau_{mem} = 1$).

Co-stimulation helps eliminate auto-reactive detectors, thereby further reducing false positives. The co-stimulation signal also serves as a trigger for generating memory detectors. Memory detectors are analogous to the secondary response in immunology (providing a swifter and more aggressive response than the primary response), and they are analogous to signature-based detectors in intrusion detection. Co-stimulation allows LISYS to take advantage of external expert advice (such as that from a human operator or an independent intrusion-detection mechanism) when it is available.

3 LISYS as a Machine Learning Algorithm

LISYS resembles an instance-based (or memory-based) machine learning algorithm. In instance-based learning, training consists of storing training examples. After training, the output value associated with a given input is inferred using some function of the stored training examples and their respective distances from the given input value. For example, the k-nearest neighbor algorithm chooses an output value based upon the k stored training examples found to be nearest to the presented input value, i.e. the input value's k nearest "neighbors." For discrete valued outputs, as in a classification task, the algorithm selects the output value that is most common among the k nearest stored training examples. For continuous valued outputs, the output is typically calculated by taking the mean of these same k output values.

Any notion of distance can be used in an instance-based learner. For continuous valued inputs, Euclidean distance is a common choice, along with variants in which input attributes are weighted. For discrete inputs, such as we use in LISYS, rules like Hamming distance are common, in which the distance between two points is equal to the number of the corresponding attributes that have the same value.

A highly simplified version of LISYS could be viewed as a modified form of 1-nearest neighbor. The class of a 49-bit string representing a TCP SYN packet is inferred by first calculating the distance between the string and each (mature) detector, where the distance is the greatest number of contiguous bits the two strings have in common. If the detector (neighbor) nearest to the presented 49-bit string is at a distance of r or less, the string is classified as anomalous. Otherwise, it is considered to be normal. We call this minimal version of LISYS "minimal LISYS."

So described, there are two essential differences between minimal LISYS and 1-nearest neighbor: the use of an r threshold, and the negative-selection procedure. Other features of LISYS can be treated as extensions to the nearest neighbor framework. Some features exploit specific features of the domain to improve generalization, while others address challenges that any machine learning algorithm must cope with in this domain, such as one-class learning, on-line learning, and concept drift. Finally, the distribution of detectors among independent nodes makes the system more robust to attack. These differences are discussed in the remainder of this section.

3.1 Inductive Bias, Concept Drift, and One-Class Learning

The inductive bias of an instance-based learner depends on three factors: (1) the notion of distance, (2) how distance is used to weight the influence of stored examples when inferring an output value, and (3) which observed training examples are actually

retained for later use. All three of these factors are realized in novel ways by LISYS.

Distance has two components: the set of input features presented to the learner and the match rule that compares different instances based on the input features. In the first case, the use of data-path triples reflects a bias that the client address, server address, and service port are the relevant features for classifying TCP connections. In the second case, bias is introduced by using a bit-based representation and ordering the input features in a particular way (at least, for LISYS running without secondary representations). Both of these choices could be changed without affecting the underlying LISYS architecture.

The bit-based representation used in LISYS disrupts strict numerical distance, e.g. 7 is closer to 8 than to 15 numerically, but 7 and 15 have a greater number of bits in common (contiguously or otherwise) than do 7 and 8. This property is sometimes referred to as a "Hamming cliff" (Caruana and Schaffer, 1988). LISYS' contiguous-bits matching rule extends the bit-based bias in that features (or bits) that are grouped more closely together on the string are presumed to be more correlated for the purposes of classification. In the example shown in figure 2 (without the use of secondary representations) we can see that LISYS reflects the bias that the bits comprising the local address and those of the compressed port representation are unlikely to be correlated. Changing bit order, as occurs with secondary representations, can have a significant effect on inductive bias (Balthrop et al., 2002a). The r-contiguous bits match rules have several other unusual properties, which distinguish their generalization abilities from other rules such as Hamming Distance. For example, the class of languages recognized by r-contiguous bits match rules, augmented with permutations, properly contains that recognized by Hamming Distance (Esponda et al., 2004).

The effect of substring hashing (see figure 4) on distance is difficult to quantify. Although the remapping of individual strings can be drastic, there is some correlation between strings before substring-hashing and after. The bit-wise hash, however, obscures this binary similarity. Finally, substring hashing permutes bytes, which is important because of LISYS' r-contiguous bits matching rule.

The second factor affecting inductive bias, how distance is used to weight the influence of stored examples (detectors), is straightforward both in k-nearest neighbor and in LISYS. The k-nearest neighbors (one-nearest neighbor in the case of LISYS) have an effect on classification, and others do not. In LISYS, activation thresholds and sensitivity levels complicate this characterization, introducing a form of distance-based weighting related to the temporal distance between presented strings.

The effect of activation thresholds on inductive bias is subtle. The direct effect is to reduce the overall number of detected anomalies, thus increasing the size of the one-class generalization. That is, activation thresholds tend to reduce false positives and increase false negatives, reflecting the bias that false positives are more harmful than false negatives. When combined with probabilistic decay of per-detector match counts, activation thresholds introduce a bias toward temporally clustered anomalies, a well-known characteristic of intrusions. Activation thresholds introduce an additional bias. Because match counts are maintained on a per-detector basis, a temporally clustered set of strings that matches a single detector is more likely to trigger an anomaly signal than a temporally clustered set of strings that are heterogeneous, each string matching a different detector. Sensitivity levels, however, attenuate this third bias by temporarily lowering the effective activation threshold when multiple, distinct detectors (present at a single node) are matched within some interval.

Finally, we turn to the third factor determining the bias of an instance-based

learner, how examples are retained. In LISYS, this third form of bias is related to on-line learning and concept drift. In on-line learning, training is interleaved with testing (classification), for potentially two different reasons. First, in many real applications, a learner must act or respond while relevant training data continue to arrive. The second reason is to cope with concept drift. Learners are often operating in nonstationary environments, in which the concept being learned can change over time. In this case, continuing to train while new data arrive is essential for the learner to track the changing boundaries between classes. Intrusion-detection systems operate in environments where both constraints apply, but the concept-drift problem is generally the more pressing.

Approaches to concept drift include fixed- and adaptive-length window schemes, as well as weighting (see section 2.2). LISYS uses an adaptive window scheme, where the longer it has been since a normal pattern has been observed (i.e. the more rare the data string), the more likely it is that a mature detector in the population will match it. Thus, the more recently an example has been seen, the greater its effect on classification, and the heavier its weight. How quickly these implicit weights decay is related to the probability of random detector death (average lifespan of a detector), and possibly the tolerization period. Factors such as the value of r and the variation in observed traffic can affect the balance between immature and mature detectors, but we do not yet have a formal characterization of how these various factors are related.

The need for inductive bias is amplified in the case of one-class learning. As discussed in Section 2.2, learning with examples of a single class is problematic because increasingly general characterizations of the class are always possible. In the extreme, the class that includes all observable inputs is consistent with any training set. In LISYS, as in other instance-based approaches to one-class learning, e.g. (Lane, 2000), a solution is to establish a tunable threshold which restricts generality. The r threshold in LISYS plays this role. It determines how distant a match is required in order to be labeled a member of the normal class. In practice, nearly all the other parameters of LISYS interact with r to determine the effective threshold, but controlling r is the most direct way to affect the extent to which LISYS generalizes.

3.2 Resistance to Attack

One of the fundamental differences between minimal LISYS and 1-nearest neighbor is the negative-selection algorithm, used to produce negative detectors from a stream of positive (i.e. normal or self) examples. This modification to 1-nearest neighbor is not absolutely necessary to implement a simple form of LISYS. It is possible to store positive examples, as in the standard k-nearest neighbor, and then use the positive examples, together with an r threshold, to label strings that match the stored detectors as normal, and others as anomalous. This very simple positive detection was studied theoretically in Esponda et al. (Esponda et al., 2004); in Section 5 we report experimental results on its performance.

Negative detectors make LISYS resistant to attack because detectors can be distributed among many nodes (computers), eliminating single points of failure. If every node in the network has incomplete coverage (because it only has a small number of detectors), then negative detectors are more easily distributed than positive detectors because no communication is required to identify an anomaly, a property we call *distributed detection*. In the case of positive detectors distributed throughout a network, with the detector set on each node providing only partial coverage of self, an anomaly is detected when all the detectors on all the nodes fail to match the same event. This

requires that the nodes communicate with one another to determine if an anomaly has been observed. This extra communication represents an efficiency penalty. Further, if a node is disabled, the mostly likely result will be an increase in the number of false positives.

We consider an increase in false negatives to be a more desirable failure mode than an increase in false positives for several reasons. First, any intrusion-detection system with too many false positives quickly becomes unusable. Although false negatives are also undesirable, a system that can detect some attacks remains useful even if it is degraded. If the system is run in conjunction with other independent intrusion-detection systems, an increase in false negatives on one system will not interact with the performance of the other systems. Finally, in the intrusion-detection problem normal traffic is much much more prevalent than attack traffic. Thus, any increase in the false-positive rate will have a large impact.

4 The Data Set

LISYS was originally tested on data collected from a subnet in the Computer Science department at the University of New Mexico (UNM) (Hofmeyr, 1999; Hofmeyr and Forrest, 2000). 50 computers were active on this subnet and data were collected for a period of 50 days. More recent experiments with LISYS were also conducted using a second data set (Balthrop et al., 2002b). Data were collected for a period of two weeks from an internal broadcast network of six computers in our own research lab. Connectivity to the internet passed through a firewall using network address translation.

The experiments reported in this paper use a third data set. As with the second data set, our goal was to collect data in a context that was as controlled as possible, but still realistic. We again used an internal network in our research lab, but this time data were collected for a significantly longer period (62 consecutive days), on a larger network (12 static hosts plus five distinct dynamic IP addresses), and with a greater number of more active users (about ten).

The internal restricted network from which the data were collected is much more controlled than the external university or departmental networks. In this environment we can understand all of the connections that occur, and we can be relatively certain that there were no attacks during the normal training periods[3]. Moreover, this environment is realistic. Many corporations have intranets in which activity is somewhat restricted and external connections must pass through a firewall. This environment could also model the increasingly common home network (albeit with a large number of hosts) that connects to the Internet through a cable or DSL modem and has a single external IP address. Attacks are a reality in environments such as these, and the attack scenarios we tested correspond to plausible occurrences in this class of environment.

4.1 Normal Data

The data set contains a total of 415,274 TCP SYN packets (including SYN/ACK packets) and roughly 55% of this is web traffic. There were approximately 6,700 packets per day during the normal period.

In (Hofmeyr, 1999), network connections to web and ftp servers were removed from the data, as well as all traffic involving hosts with local, dynamically assigned IP addresses. Although such traffic is highly irregular (because web and ftp servers are intended to accept connections from widely varying locations), we were interested

[3]Rolling coverage and negative detection help LISYS withstand attacks during training. However, training on attack-free data makes the experimental results easier to interpret.

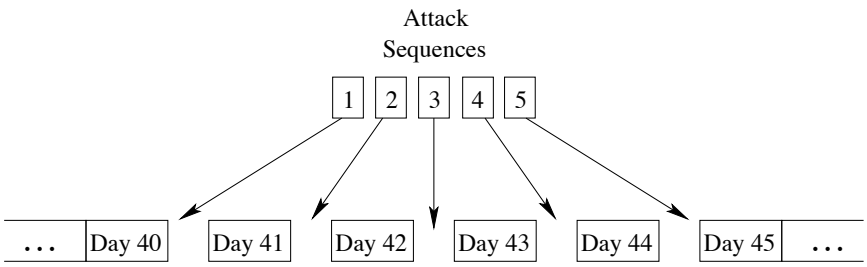

Figure 5: The five attacks were inserted into the normal data beginning after the 40th day and with 24 hours of real traffic separating each successive attack.

in assessing how well LISYS could function in the presence of such traffic. Instead of completely removing web connections, the current data set simulates the behavior of a proxy server. All outgoing connections to port 80 (http) or port 443 (https) are re-mapped to port 3128 on the proxy machine. This is very close to what the traffic would have been like if we were using the web proxy cache SQUID (Squid,). Traffic involving ftp and internal addresses assigned via DHCP was left undisturbed.

4.2 Attack Data

We followed Hofmeyr's method (Hofmeyr and Forrest, 2000) for incorporating attacks into the data set. By examining his 50 days worth of collected data, Hofmeyr identified several attacks that had taken place during the collection period. He then isolated and removed the continuous sequence of packets corresponding to each attack. Each sequence began and ended with the first and last packets associated with the attack, and contained all packets that appeared in between, including both normal and attack packets. These attacks were then reinserted into the normal data stream at well-controlled locations, as shown in figure 5.

We drew our attacks from the collection of eight individual attack events described in Ref. (Balthrop et al., 2002b). We chose these attacks because they were common attacks at the time and supported by Nessus, a free security scanner (`http://nessus.org`) which we used to perform the attacks. All of the attacks, with the exception of one denial-of-service attack, were performed using a laptop connected to the internal network. The laptop was able to acquire a dynamic IP address because it had a physical connection to the internal network. The eight attacks included a denial of service (from an internal computer to an external computer), a firewall attack, an ftp attack against an internal machine, an ssh probe against several internal machines, an attack probing for specific services, a TCP SYN scan, an nmap tcp connect() scan against several internal computers, and a full nmap port scan. Of these eight attacks, we chose to include five, discarding the TCP SYN scan and nmap attacks, which were each comprised of an extremely large volume of packets (about 75,000 total between the three) and could be easily detected by a packet-flood sensor.

Like Hofmeyr, each isolated attack sequence began with the first packet associated with the attack, ended with the last packet of the attack, and included all packets in between (many of which were legitimately normal). Also following Hofmeyr, we took care to label each individual packet that appeared within the attack sequence as either self or non-self. The packets in each attack sequence were further re-mapped to reflect the current network configuration. Finally, we inserted each of the attacks into our

data, with the first sequence beginning after the 40th day and each of the remaining sequences inserted after each successive day.

5 Experiments

In this section we describe a series of experiments conducted on the data set described above. Although we collected the data set on-line in a production network, the experiments themselves were conducted in an off-line set of simulations performed on a single computer. This made it possible to compare performance across many algorithmic variants as well as different parameter values. The programs used to generate the results in this paper are available from http://www.cs.unm.edu/∼immsec. The programs are part of the LISYS package and are found in the LisysSim directory.

After describing the parameter settings used in the experiments, we begin our experiments with a null model of learning and compare the performance of the null model to that of full LISYS. We then describe a series of experiments which incrementally modify these two extremes and test the performance of several intermediate variants.

5.1 Anomaly identification without generalization: "Packet Hash"

As we have seen, LISYS is a complex algorithm with many components. Before applying LISYS to the new data set, one might ask just how challenging the data set really is. How well might we detect the attacks in this data set with an approach that is a great deal simpler than LISYS?

To address this question, we study the performance of a null algorithm called *packet hash* (Chao, 2001). Packet hash is a form of the one-nearest neighbor algorithm, modified for one-class learning and with a matching threshold that requires an exact match. Packet hash provides a baseline for comparison from an ML point of view because it performs no generalization whatsoever. During the fixed-length training period, packet hash records every unique packet it observes. After training, packet hash labels any novel packet—that is, any packet it did not observe during training—as anomalous. Packet hash thus reflects an extreme inductive bias, namely, that all possible members of the class are observed during training. This choice represents the maximally specific bound on the set of all possible target concepts consistent with the observed data (see section 2.2).

To run packet hash, it is necessary to specify the length of the training period. For these experiments, we trained on the first 30 days of data. During training, packet hash examined every SYN packet (of which there were 225,470) and stored a copy of each unique packet (there were 860 unique packets in this case). After training, every observed packet was compared to the list of stored packets. All packets not exactly matching one of the previously stored packets were flagged as anomalies. In contrast to LISYS' negative detection scheme, packet hash uses positive detection, storing a representation of self rather than of non-self. It can be thought of as a 1-nearest neighbor learning algorithm with an exact matching requirement.

As expected, the packet-hash algorithm performed excellently in terms of true positives, identifying 97.7% of all attack packets[4]. The cost, however, is a mean false-positive rate of 101.7 packets per day. In most settings, a false positive rate of 101.7 per day would be an unacceptably high price to pay for such a good true-positive rate.

[4] 2.3% of the attack packets (packets observed during the time of an attack) were identical to normal packets observed during training.

Parameter	Description	Value	Hofmeyr & Forrest's Value
l	string length	49 bits	49 bits
r	match length	12 bits	12 bits
τ	activation threshold	100 matches	10 matches
$1/\gamma_{match}$	match decay period	1 day	1 day
ω	per-match sensitivity increment	1.5	1.5
$1/\gamma_{\omega}$	sensitivity decay period	0.1 days	0.1 days
T	tolerization period	20 days	4 days
T_s	co-stimulation delay	1 day	1 day
$1/p_{death}$	life expectancy	14 days	14 days
n_d	number detectors per node	100	100

Table 1: LISYS parameter values: Column 1 gives the parameter symbol, column 2 gives a brief description of the parameter, column 3 lists the value used for our experiments, and column 4 gives the values used in the experiments reported in (Hofmeyr and Forrest, 2000). Although many parameters are expressed in days, they are in practice specified in terms of the number of packets seen, where a day corresponds to 6,200 packets in the current experiments and 25,000 in the earlier study. Decay times are the expected time for the value to decay by one.

5.2 The performance of LISYS

A simulated run of LISYS consisted of 50 nodes, each node with its own randomly generated substring-hash and 100 detectors. Each simulation ran for the duration of the entire data set. Differences in performance between runs arose from random variations in certain operations, such as generating substring-hash functions and detectors.

In contrast to packet hash, LISYS does not have distinct training and testing phases. However, until the initial tolerization period has passed, no detectors will have yet become mature and able to label packets. Once some detectors mature, LISYS both trains and identifies anomalies continuously. As with packet hash, we began recording true and false positives after 30 days of simulated data.

LISYS relies on several parameters to control its behavior, and we do not yet have extensive knowledge about how to use these parameters. One reason for experimenting with yet another data set was to study how much parameter tuning is necessary to achieve acceptable results on new data sets. Beginning with Hofmeyr's (Hofmeyr and Forrest, 2000) original values, we found a set of parameters that worked well on the new data set with minimal experimentation (see Table 1 and Section 6.1). The parameter tuning consisted of only three runs of LISYS, in which tried small variations from Hofmeyr's original parameters scaled to our new data set (e.g. based upon a different average number of SYN packets per day). In addition, we conducted the parameter tests using only the first 42 days of data and not the full 68 days. Of the ten parameters listed, we changed only two values—activation level and tolerization period—to achieve reasonable performance. Of the remaining eight parameters, four values were unchanged and the other four, specified in days, were simply rescaled to account for the lower median number of packets per day in our data set (6,200) compared to that of Hofmeyr. The two values that were adjusted were directly related to controlling false

Attack	Total # of attack packets	Fraction of runs in which at least one attack packet was flagged
Firewall	12 (out of 12 total)	1/120
"Useless Services"	27 (out of 27 total)	99/120
ssh	56 out of 60 (total)	83/120
D.O.S.	284 (out of 301 total)	119/120
ftp	576 (out of 664 total)	109/120

Table 2: LISYS performance on the five attacks in the data set: Performance is reported as the fraction of runs in which one or more packets from each attack were flagged

positives.[5]

The resulting performance was quite good. LISYS detected the majority of attack packets while generating only a handful of false positives per day. Out of 120 runs, LISYS flagged on average 61.4% of all attack packets as anomalous while incurring 4.1 ± 0.7 (95% confidence interval) false positives per day averaged over the final 32 days of data, as shown in figure 6 and more fully explained in the next section. Packet hash's false-positive rate over this same period was 24 times that of LISYS.

A more important aspect of performance is the ability to detect some anomalous packets in each attack sequence. LISYS performed well in this respect as well, detecting at least one packet from the large majority of runs in four of the five attacks (see table 2). The one attack on which LISYS failed to flag almost any packets was the shortest, consisting of a total of only 12 packets.

The comparison between packet hash and LISYS suggests that generalization is a crucial mechanism for controlling false positives. Most of the architectural features of LISYS contribute to its generalization performance, including: activation thresholds and sensitivity levels, co-stimulation, approximate matching with r-contiguous bits, multiple secondary representations, rolling coverage, memory detectors, and negative selection/detection. In the remainder of section 5, we present a series of experiments designed to assess the contribution of these mechanisms. We begin by extending the original packet hash algorithm, progressively incorporating three of the simplest and most intuitively straightforward mechanisms: activation thresholds, co-stimulation, and approximate matching.

5.3 Adding activation thresholds to packet hash

We added an activation threshold to the packet-hash algorithm in the following way: A single global count is maintained during the testing phase which is incremented every time an anomalous packet is identified. As with match counts in LISYS, there is a decay parameter that controls how quickly the global match count decays. When the global count exceeds the specified threshold, an anomaly is signaled. The global activation count does not account for per-detector similarity between suspicious packets, and thus it is not exactly equivalent to the activation threshold used in LISYS. The packet-hash activation threshold is global because of positive detection. Because the algorithm only

[5]The co-stimulation delay also controls false positives, but it is properly a parameter of the experimental simulation rather than the LISYS system itself. Lowering this value would likely further reduce false positives, but it would also change our assumptions about how frequently the system can expect human intervention.

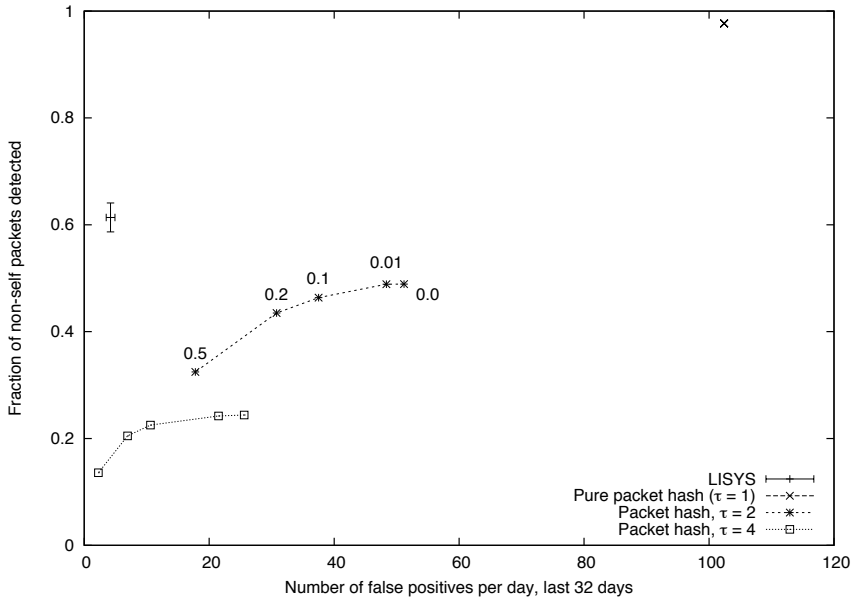

Figure 6: Pure Packet Hash and Activation Thresholds: Performance of LISYS (shown with a 95% confidence interval) compared with that of packet hash. The performance of packet hash is shown for activation threshold values of 1 (equivalent to pure packet hash), 2, and 4, each with match-count decay probabilities varying from 0.5 (farthest left) through 0.2, 0.1, 0.01, and ending with 0.0 (farthest right).

flags an anomaly when a test packet fails to match any cached packets, there is no detector to associate with the mismatch, as there is with a match in LISYS.

Figure 6 shows the performance of packet hash with activation thresholds for varying threshold values and match-count decay probabilities. Results are included for several threshold values and match count decay parameters. LISYS' performance using the parameter values from table 1 is also shown. All points reflect the mean performance over ten independent runs[6]. Error bars are given for LISYS which show the 95% confidence level for values measured on each axis. Because packet hash is more deterministic than LISYS, the associated confidence bounds are so small that they are difficult to display and were omitted.

In the case where the activation level is set to one (pure packet hash), only a single point was plotted, because the decay parameter has no effect. For activation levels two and four, however, as the match count decay probability is increased the match count decays more quickly, making it less and less likely that the activation threshold will be reached. The net result is that as the decay probability increases, fewer overall anomalies, either true or false positives, are signaled. This effect can be seen in the graph, as the plotted points are closer to the origin (0 false positives, 0 true positives) for higher decay values. A qualitatively similar result is observed as the activation threshold is increased, as can be seen by comparing the points for activation threshold 4 with those of activation thresholds of 2 and 1.

When the decay rate is set to zero, activation thresholds simply divide the overall

[6]Unless otherwise noted, all reported results for packet hash in this paper were averaged over ten runs.

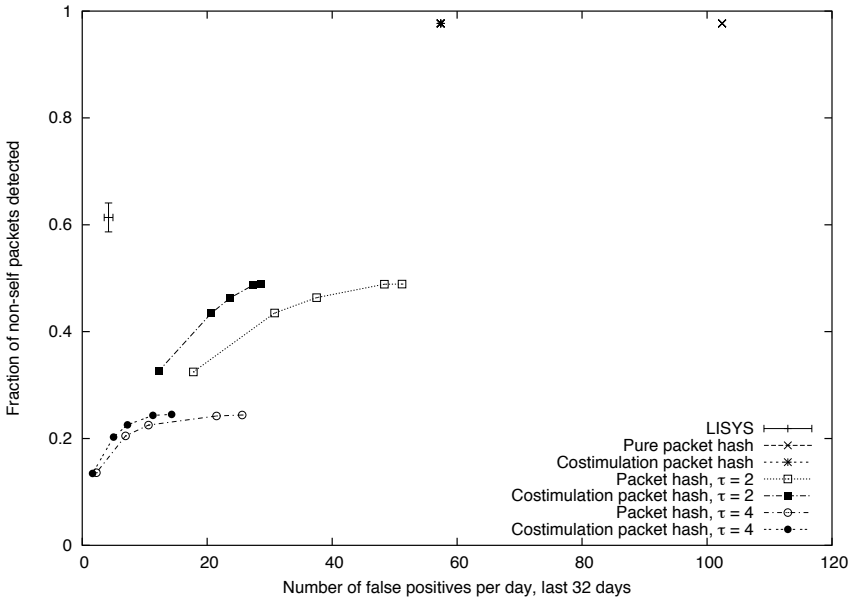

Figure 7: Co-stimulation: Performance of LISYS (shown with a 95% confidence interval) compared with that of Packet Hash with activation thresholds, both with and without co-stimulation.

number of anomalies signaled by some constant factor, and the system's discrimination ability remains unchanged. However, as the decay rate is increased, so is the bias that true attacks are correlated with temporally clumped anomalies. The extent to which this bias accurately reflects our data is revealed by observing the results of increasing the decay rate; as the rate increases from zero, the points representing performance move further to the left than they do down, indicating a greater reduction in false positives than in true positives. However, as the bias toward temporal clustering increases, the true-positive rate is reduced at a significant rate as well, indicating that there is a limit to the tight clustering of attack packets.

These experiments show that activation thresholds coupled with a well-chosen match decay rate significantly improve packet hash's discrimination ability. However, by the time the false-positive rate is reduced to a level commensurate with that of LISYS' (say, below ten per day), the true-positive rate is far lower than that of LISYS.

5.4 Adding co-stimulation

Next, we incorporated co-stimulation into packet hash. Co-stimulation uses an additional source of information (feedback from a human operator), making it an excellent candidate to improve performance. Furthermore, because feedback from the human operator is ongoing, incorporating co-stimulation into packet hash allows training to extend beyond the initial training period.

Similar to the case with activation thresholds, co-stimulation necessarily takes a different form in packet hash due to the differences between negative and positive detection. Activated detectors that correspond to false positives fail to receive co-stimulation, and they are eliminated from the detector pool within 6,200 packets of

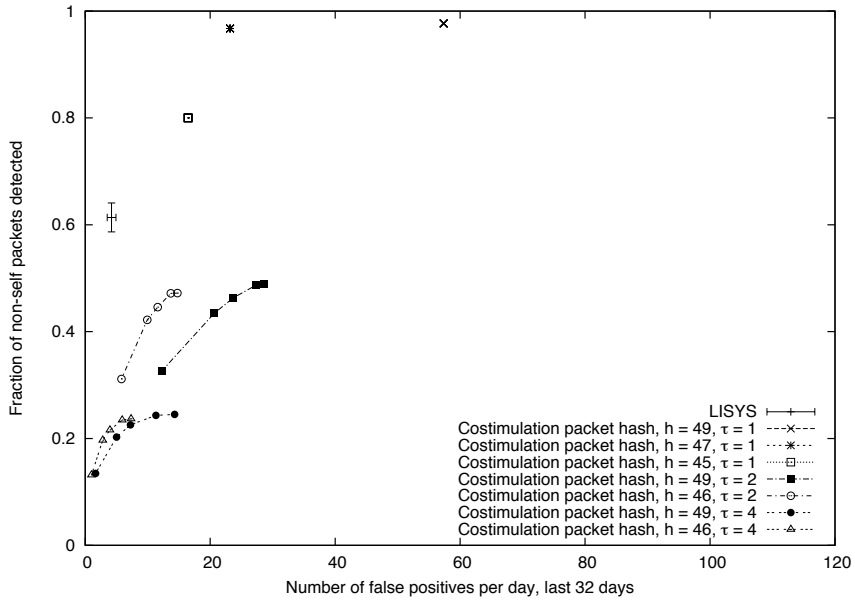

Figure 8: Approximate Matching: Performance of LISYS compared to that of packet hash with activation thresholds and co-stimulation, both with and without approximate matching using Hamming distance.

their "first offense", i.e. approximately one day later in simulation time. Because packet hash uses positive detection, the equivalent mechanism involves adding detectors (i.e. cached packets) after a false positive. Whenever a packet is flagged as anomalous, and is determined to be a false positive, that same packet is added to the list of cached packets 6,200 packets later.

The results of this extension are shown in figure 7. For comparison, we also show results for LISYS and for packet hash without co-stimulation (repeating some results from figure 6). As the figure shows, co-stimulation clearly improves performance; the false-positive rate is reduced without degrading performance on true positives. However, LISYS maintains superior performance on true positives and comparable performance on false positives.

5.5 Approximate Matching

We now consider the inductive bias arising from LISYS' approximate matching between detectors and packets. LISYS assumes that similar packets are likely to be similarly anomalous or normal. This is reflected in the match count, which is maintained on a per-detector basis. However, the exact nature of the bias is determined by r-contiguous bits matching. Because non-self is vastly larger than self for any interesting application, including the TCP SYN-packet domain, some kind of approximate matching is required to support negative detection. This is because it would be infeasible to store an exact detector for every single possible non-self packet.

Adding approximate matching to the packet-hash algorithm is straightforward. We tested two different matching rules for packet hash: Hamming distance and r-contiguous bits. Both rules allow packet hash to generalize because some packets other

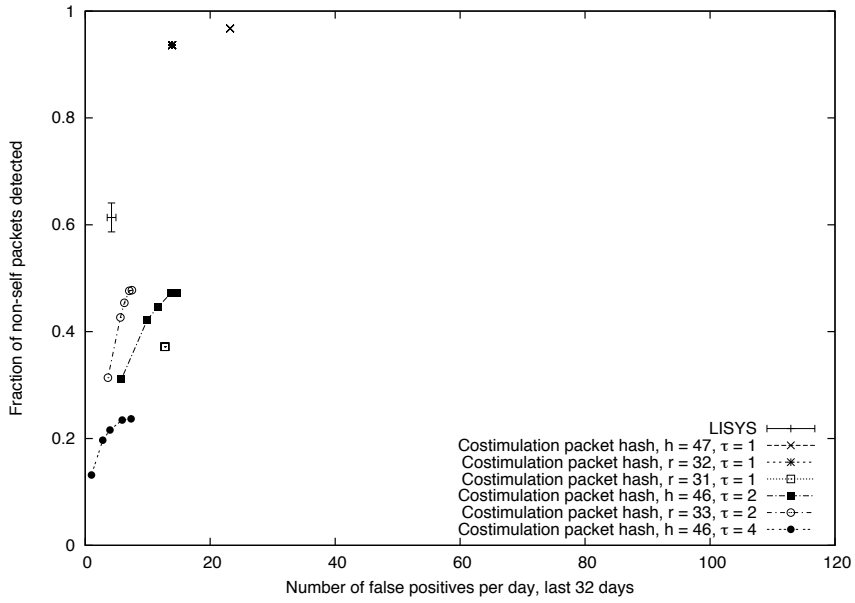

Figure 9: Hamming distance vs. r-contiguous bits matching: Performance of LISYS compared to that of packet hash with approximate matching, activation thresholds, and co-stimulation, both with Hamming distance and with r-contiguous bits matching.

than those observed during training (i.e. those which are similar to those observed) are considered self. In the testing phase, a packet is flagged as anomalous under the Hamming distance rule if it fails to match any of the cached self packets in at least the number of bits specified by the Hamming threshold, h. The behavior for r-contiguous bits is exactly as described for LISYS.

Figure 8 shows the result of adding approximate matching to packet hash. For comparison, we again include the results for LISYS. Figure 8 shows results for several Hamming thresholds as well as for exact matching, equivalent to a Hamming threshold of 49. The Hamming threshold results were calculated with an activation threshold of 1 (no activation thresholds). A false-positive rate around 20 per day is achieved with a true-positive rate of 80 - 98%. In a context where this false-positive rate were tolerable, the performance of packet hash with fuzzy matching and co-stimulation would be preferable to that of LISYS. However, if lower false-positives are required (say, less than ten per day), then LISYS' performance remains preferable.

Figure 9 compares approximate matching using Hamming distance to that using r-contiguous bits. The results show that r-contiguous bits further improves packet hash's generalization ability, but still does not achieve LISYS' rate of less than ten false positives per day.

5.6 Ablation Experiments with LISYS

We have now considered the effect of adding three out of the seven mechanisms listed in section 5.2 to packet hash, namely activation thresholds, co-stimulation, and approximate matching. At this point, to measure the contributions to performance of the remaining mechanisms it was more straightforward to perform ablation experiments,

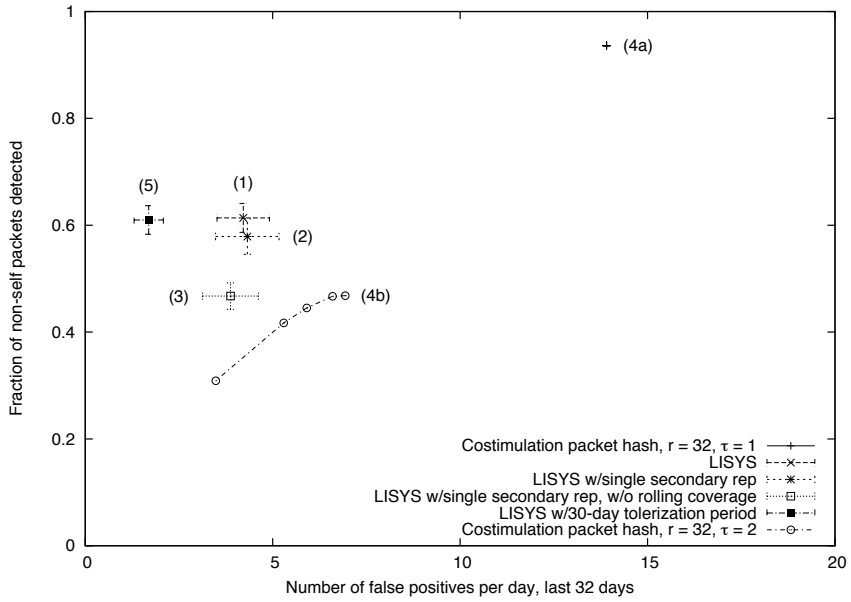

Figure 10: Representation Diversity and Detector Turnover: Performance of LISYS (1) compared with that of (2) LISYS without representational diversity (same representation at every node); (3) LISYS with neither representational diversity nor detector turnover; (4) packet hash with approximate matching, co-stimulation, r-contiguous bits matching ($r = 32$), and activation threshold values of 1 and 2 (4a and 4b, respectively); and (5) LISYS with a 30-day tolerization period. Note that the X-axis ranges from 0 to 20, instead of 0 to 120 as in figures 6-9.

removing mechanisms from LISYS, rather than to continue adding similar mechanisms into packet hash[7].

In the first such experiment, memory detectors were disabled. Upon co-stimulation, detectors were not promoted to be memory detectors; the only effect of co-stimulation was that the given detector was not killed. Disabling memory detectors had no significant effect upon performance. This finding appears to result from how attacks are distributed in the data. Recall that what distinguishes memory detectors is that they are long-lived and have a low activation threshold, signaling an anomaly as soon as they match. Note also that in these experiments the match count of a normal mature detector is not reset to zero once the detector is activated and flags an anomaly (section 2.4.3). With the match count expected to decay once per day (table 1) and attacks spaced apart by one day (figure 5), detectors activated by earlier attacks are likely to remain activated when they encounter subsequent attacks, regardless of whether they have been promoted to memory detectors. If the attacks were spread further apart in time, the memory detectors would likely provide a significant benefit.

In the second ablation experiment, we eliminated representational diversity from LISYS. In a normal run of LISYS, there are 50 independent nodes, each with its own secondary representation. For comparison, we conducted an additional 120 runs of LISYS in which, for each run, the same randomly generated substring-hash function was used at each of the 50 nodes. That is, a different randomly generated secondary representation was used in each of the 120 runs, but within a given run each of the 50 nodes used the same representation as all of the other nodes. As was the case with eliminating memory detectors, the results (shown in figure 10, labeled "(2)") do not reveal a significant change in mean performance.

Unlike the other mechanisms considered so far, the effect of multiple representations in LISYS is not so much to decrease the false-positive rate as it is to increase the overall number of matches, i.e. for both true and false positives. Because there are limits to what can be detected using any single representation and match rule, one expected effect of multiple secondary representations is to increase the overall number of matches, i.e. for both true and false positives. However, this assumes that there are real limits to what can be detected using a single representation. Because we do not observe significantly greater coverage with multiple representations, it is possible either that these limits are not reached by the given data set or that reaching them would require a greater number of detectors per node. In any case, there is one observable effect of multiple secondary representations that does not appear in figure 10. Employing a variety of representations within a single run rather than committing to a single one results in a lower variance about the mean between runs: With a single representation per node, the standard deviation in the number of false positives per day is 5.34, while that with multiple representations is 3.87 (interestingly, the variance in true positives remains essentially the same). Additional effects of different secondary representations in LISYS were reported in (Balthrop et al., 2002a).

A final ablation experiment was designed to assess the effect of detector turnover and rolling coverage in LISYS (see figure 10, label "(3)"). As in the previous ablation experiment, 120 runs of LISYS were conducted, each with one randomly chosen substring-hash function replicated over all 50 nodes. However, the detector lifecycle was "frozen" at the end of the first tolerization period. At the beginning of a run, 100 detectors were generated per node. These 100 detectors were subsequently subjected

[7] As with the unablated LISYS results, all results of ablation experiments reported reflect the average of 120 runs, displayed with a 95% confidence interval.

to negative selection during the tolerization period. Of the initial 100 detectors per node, those which survived negative selection became mature detectors at the end of the initial tolerization period. At this stage, the detector set was frozen, meaning that no further immature detectors could become mature and no mature detectors were subject to probabilistic death or death by failure to be co-stimulated. This configuration resembles packet hash with activation thresholds and approximate matching, only with negative instead of positive detectors. As LISYS in this configuration has a fixed training period, it was set to 30 days to match that of packet hash, for ease of comparison.

Several observations can be made about this final result. First, the performance of "static LISYS" and the maximally enhanced form of packet hash is quite close. At this point, by incrementally enhancing packet hash and degrading LISYS, we have almost closed the gap between them, both algorithmically and performance-wise. Nonetheless, there is still a statistically significant difference between the performance of the two algorithms (the performance of packet hash lies outside the 95% confidence interval on the mean of LISYS' performance). LISYS appears to maintain a better balance between true and false positives. Although this result might not carry over to other data sets, this final performance gap must be attributed to one significant remaining algorithmic difference: negative vs. positive detection.

This is a surprising result, because the principal advantage attributed to negative detection in LISYS is that it supports distributed detection (section 3.2). Moreover, theoretical work (Esponda et al., 2004) has shown that for a maximal number of positive and negative detectors, with full-length r-contiguous bits, coverage is nearly equivalent from a generalization standpoint. The difference possibly stems from the fact, discussed in section 3.1, that match counts are maintained on a per-detector basis with negative detection, introducing a slightly different inductive bias from the activation thresholds used with positive detection. The bias favors multiple related packets which match the same detector. Although it is interesting that negative detection provides this detection ability, it remains to be seen whether this bias would be helpful in other data sets. If this bias were found to be undesirable in some circumstance, it could be reduced in LISYS with sensitivity thresholds.

A second surprising result from this last experiment was that eliminating rolling coverage reduced the number of true positives without significantly changing the number of false positives. Normally, because the object of rolling coverage is to account for changing self, we would expect leaving it out to produce higher false-positive rate. In this case, the increase in tolerization period from 20 to 30 days may have affected the result. To test this hypothesis, we ran full LISYS with a 30-day tolerization period. In comparison to *full* LISYS with a 30-day tolerization period (figure 10, label "(5)"), LISYS without rolling coverage (figure 10, label "(3)") does indeed incur a higher false-positive rate, confirming the hypothesis.

6 Discussion

The empirical results presented in this paper are important for two reasons. First, by making the connection to machine learning explicit, we have shown how several of the mechanisms of LISYS could be incorporated into other machine-learning frameworks, in particular, to methods based on k-nearest neighbors. By quantifying the contribution of each individual mechanism and analyzing the nature of its contribution, it should be easier to make good decisions about how to combine the mechanisms of other ML mechanisms with those of LISYS. Secondly, we have set the stage for further mathemat-

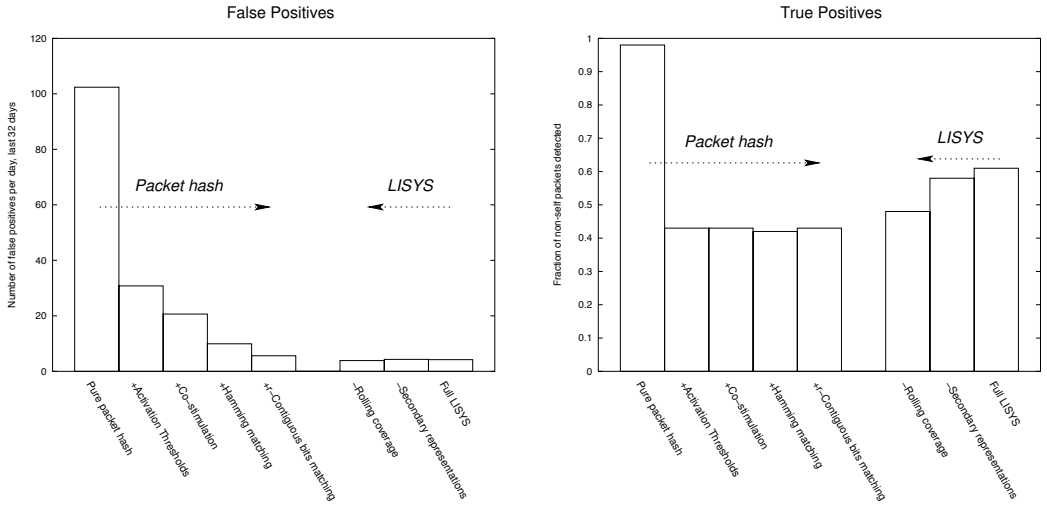

Figure 11: The differential contributions made by each mechanism, in terms of both false and true positives, for a chosen set of representative parameters.

ical investigations of the properties of the various LISYS mechanisms and their careful comparison with similar methods in machine learning. Although the theoretical properties of the r-contiguous bits match rule and negative selection have been studied extensively, the other mechanisms, such as activation thresholds, detector turnover, and co-stimulation, have not.

6.1 Experimental Results

Due to the great variety of mechanisms that comprise the LISYS architecture, together with their interactions, formal analysis of the complete system is a daunting task. Although LISYS has been tested experimentally on other data sets, there has been a need for more complete experimental testing. We tested the contribution of LISYS' components by progressively integrating them into a stripped-down version of LISYS or deleting (ablating) them from the full system. With the exception of multiple secondary representations, each mechanism contributes an observable performance benefit and all are necessary to maximize performance. The relative contributions made by each mechanism are shown in figure 11.

Measuring the performance of the entire system on our data set, which is extensive but still well-controlled and understood, is a further contribution to the empirical evaluation of LISYS. LISYS' performance on the new data set has significantly reinforced our confidence in the system's potential, relevance, and interest. At the most elementary level, having achieved good performance in a third, distinct network context is encouraging. Although this data set was collected at the same institution (UNM) and the same department (CS), it differs from Hofmeyr's original data set in several important ways. It was collected from a well-controlled but production subnet behind a firewall, and it was collected after four years of subsequent technological evolution. Finally, it was collected over a longer period of time, using data from active users and hosts on the network.

Experiments conducted with this data set were also more challenging than

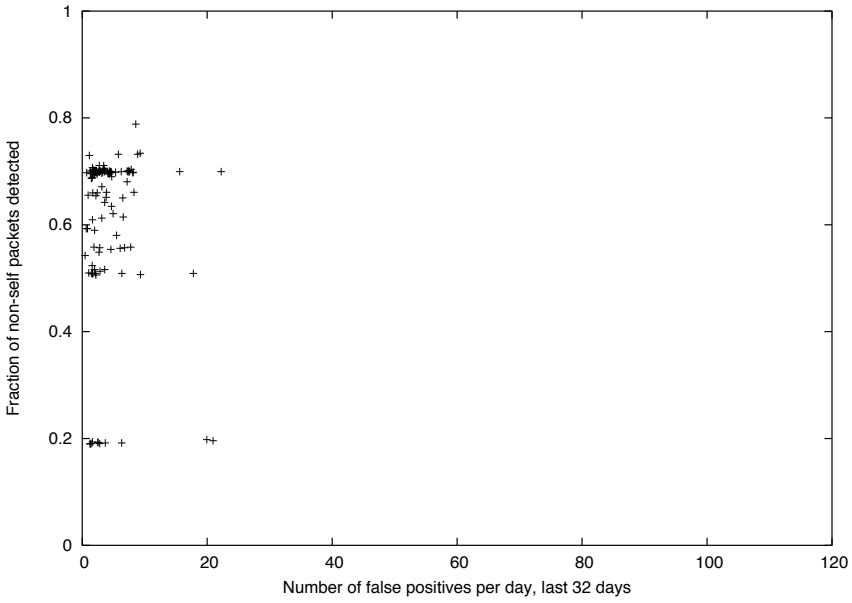

Figure 12: The performance of 120 independent runs of LISYS.

Hofmeyr's experiments in some ways. In order to simulate network traffic for longer than the 50-day collection period, Hofmeyr randomly resampled self (non-attack) packets from the original data set for all experiments on the full LISYS system. This method was necessary to fully test certain aspects of the dynamic detector lifecycle, and the packets in the resulting data stream all occurred with the same frequency at which they had occurred during training. However, this approach eliminated the temporal structure present in the original self data. Although attack packets generally appear closely clustered in time, we have observed that false-positive packets can also be temporally clustered. Thus, the experiments in this paper, which all use normal data in the original order of occurrence, present a more challenging test for LISYS. Also, we performed less filtering of highly variable data-paths such as incoming http connections.

Although the successive versions of packet-hash performed worse than LISYS' average score (to a statistically significant degree), LISYS' performance was much more variable (see scatterplot in figure 12). In cases where the performance level of packet hash is acceptable, it's greater consistency would be an advantage over LISYS. However, LISYS' consistency could likely be improved with further research. In particular, if a way were found to filter out lower performing points (e.g., using evolutionary methods to eliminate poor secondary representations) early on, LISYS' mean performance would increase significantly.

In assessing implications of the experiments, how parameters were tuned is an important issue. One one hand, it was encouraging was that LISYS yielded reasonable performance in this next context after very little adjustment to the parameters used by Hofmeyr. However, these experiments did not follow the strictest methodology in that results from using the full data set, attacks and all, were used to guide parameter adjustment. In machine-learning terms, full cross-validation was not performed. Were an IDS deployed in a production context, what would be available for parameter

tuning would be a sample of normal data along with, perhaps, some sample attacks. After tuning, the true value of the system would be calculated from its performance on subsequent normal data along with any attacks which occurred. Thus, the fact that the results reported here are for the same data set on which the parameters were tuned does limit our ability to predict how well LISYS might perform in a production context.

This shortcoming is mitigated by the fact that the parameter adjustments were minimal and only involved two out of the many parameters in LISYS. The tuning was directed towards false positives, and only those parameters most relevant to this issue were adjusted. However, a stricter cross-validation methodology would be an important feature in future experiments.

6.2 Extensions to LISYS

Since the original LISYS experiments were conducted, switched networks have become much more prevalent. There are many ways LISYS might be adapted to switched networks. Hosts could share SYN packet headers with each other, either through the switched network or through a secondary, dedicated broadcast network (loosely analogous to the biological lymph system). A second option would be to deploy LISYS in the switch, which would likely work well, although it would turn LISYS into a centralized system. Perhaps the simplest approach would be to do nothing, allowing the detectors on each host to see only the traffic relevant to that host. This would reduce redundancy of coverage provided by multiple hosts examining the same data, but could lead to an interesting form of diversity. Technology continues to change, and the spread of wireless networks now presents even greater challenges and opportunities for configuring LISYS in a network environment.

Another LISYS design decision carried over into this paper is the focus on TCP SYN packets, and in particular on the data-path triple of sender, receiver, and target port. Although there are certainly many attacks that cannot be caught at the SYN-packet level, there are the advantages of simplicity and efficiency, as well as allowing comparisons with earlier results. LISYS performs surprisingly well, even with such a limited data stream. The restriction to SYN packets in LISYS is not hard-wired, and the system could be easily extended to other aspects of network traffic. Although LISYS performs well in the SYN-packet domain, some of the attacks we studied would likely be caught by other, less adaptive mechanisms such as signature scanners.

LISYS models the adaptive immune system. In nature, the adaptive immune system works together with the innate immune system. The innate immune system copes with attacks that do not require adaptation in an organism's lifetime—that is, those that are stable enough that evolution has adapted to them. For instance, because there are particular polysaccharides bacteria are obliged to express, the innate immune system has hardwired detectors for them. It is impressive that LISYS can learn to catch attacks which make use of completely unused ports or exhibit other obvious attack signatures, but following the natural immune system, it would be interesting to integrate LISYS with an analog of the innate immune system, i.e. a signature-based IDS such as Snort (Caswell and Roesch,).

7 Conclusions

In conclusion, the work reported in this paper had three complementary goals: To extend the empirical evaluation of LISYS' performance to new and more challenging data sets, to deepen our understanding of the contributions made by each component of LISYS to its overall performance, and to connect LISYS with the broader context of

machine learning.

We described experiments with a new data set and found that LISYS performed well with minimal tuning of its original parameters. When we studied the relative contributions of the many components of LISYS' architecture, we found that nearly every component positively affects performance. We also examined the various LISYS components in a machine-learning context. We conclude that many of the components that help LISYS perform well in the intrusion-detection domain could prove valuable in other challenging ML domains which involve one-class learning, concept-drift, and/or on-line learning. These components include: co-stimulation and memory detectors, rolling coverage; activation thresholds and sensitivity levels, r-contiguous bits matching, and secondary representations.

Co-stimulation is a mechanism for incorporating feedback from an expert teacher during system operation, and memory detectors are a mechanism for storing this feedback for future use. Together, they enhance LISYS' capacity for on-line learning and extend LISYS beyond a strict one-class learning framework. Rolling coverage addresses concept drift. In contrast with fixed- and adaptive-window schemes, the probabilistic deletion and regeneration of detectors produces a window which is probabilistic, decaying gradually with time. Coupled with co-stimulation, the window size is also adaptive. Activation thresholds and sensitivity levels bias the system against occasional or sporadic alarms and towards high-frequency bursts of anomalies. The principal role of these mechanisms is to reduce false positives. Similar mechanisms may be useful in other anomaly-detection domains where some stream of data is being monitored, particularly when false positives are costly. The r-contiguous bits matching rule was found to be more powerful than Hamming distance matching in LISYS. r-contiguous bits matching permits more fine-grained distinctions to be made between self and non-self (Esponda et al., 2004), and may provide for a form of automatic feature selection (Esponda et al., 2004). Another attractive property of bit-based matching rules such as r-contiguous bits is that they are easily and efficiently implemented in digital logic circuits (Bradley and Tyrell, 2001). Multiple representations are currently used to increase overall coverage and reduce generalization. In the future, they may be used to tune inductive bias, i.e. learning to learn (Pratt and Jennings, 1997).

The LISYS system incorporates several different learning mechanisms, some of which we understand better than others. Negative vs. positive detection and r-contiguous matching have both been theoretically analyzed. In this paper we have begun the process of understanding the remaining mechanisms, although an important area of future work is to characterize formally activation thresholds, on-line learning, and the effect of permutations.

LISYS was designed to operate in an environment in which false positives are known to be a significant problem. As we showed in the analysis of LISYS' inductive bias, many of the learning algorithms in LISYS support this bias. Returning to the ARTIS framework mentioned in the introduction, it will be interesting to see how ARTIS performs in other settings where false positives may not be critical.

Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (grants ANIR-9986555, CCR-0311686, and DBI-0309147), Defense Advanced Projects Agency (grants AGR F30602-00-2-0584 and F30602-02-1-0146), the Intel Corporation, and the Santa Fe Institute.

Many people contributed to LISYS and made helpful comments and advice over

the past three years, including Dennis Chao, Fernando Esponda, Paul Helman, Hajime Inoue, Steven Hofmeyr, Todd Kaplan, Anil Somayaji, and Jason Stewart.

References

Anchor, K. P., Zydallis, J. B., Hunch, G. H., and Lamont, G. B. (2002). Extending the computer defense immune system: Network intrusion detection with a multiobjective evolutionary programming approach. In Timmis, J. and Bentley, P. J., editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 12–21, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

Anderson, D., Frivold, T., , and Valdes, A. (1995). Next-generation intrusion detection expert system (nides): A summary. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA.

Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135.

Axelsson, S. (2000). The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186–205.

Balasubramaniyan, J., Garcia-Fernandez, J. O., Spafford, E. H., and Zamboni, D. (1998). An architecture for intrusion detection using autonomous agents. Technical Report Coast TR 98-05, Purdue University.

Balthrop, J., Esponda, F., Forrest, S., and Glickman, M. (2002a). Coverage and generalization in an artificial immune system. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*.

Balthrop, J., Forrest, S., and Glickman, M. (2002b). Revisiting lisys: Parameters and normal behavior. In *CEC-2002: Proceedings of the Congress on Evolutionary Computing*.

Bradley, D. W. and Tyrell, A. M. (2001). The architecture for a hardware immmune system. In *Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*, pages 193–200.

Caruana, R. A. and Schaffer, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *MACHLERN5*, San Mateo, California. Morgan Kaufmann.

Caswell, B. and Roesch, M. Snort: The open source network intrusion detection system. `http://www.snort.org/`, Downloaded 2003.

Chao, D. (2001). Personal Communication.

Dasgupta, D. (1999). Immunity-based intrusion detection system: A general framework. In *Proceedings of the 22nd National Information Systems Security Conference*.

Dasgupta, D. and Gonzalez, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3).

Debar, H., Dacier, M., and Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):361–378.

Dowell, C. and Ramstedt, P. (1990). The computerwatch data reduction tool. In *Proceedings of the 13th National Computer Security Conference*.

Egan, J. P. (1975). *Signal Detection Theory and ROC Analysis*. Academic Press, New York.

Esponda, F., Forrest, S., and Helman, P. (2004). A formal framework for positive and negative detection schemes. *IEEE Transactions on System, Man, and Cybernetics*, 34(1):357–373.

Forrest, S., Perelson, A. S., Allen, L., and kuri, R. C. (1994). Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA. IEEE Computer Society Press.

Gaffney, J.E., J. and Ulvila, J. (2001). Evaluation of intrusion detectors: a decision theory approach. In *2001 IEEE Symposium on Security and Privacy. S&P 2001, 14–16 May 2001, Oakland, CA, USA*, pages 50–61, Los Alamitos, CA, USA. IEEE Computer Society.

Gold, E. (1967). Language identification in the limit. *Information and Control*, 10:447–474.

Gomez, J., Fonzalez, F., and Dasgupta, D. (2003). An immuno-fuzzy approach to anomaly detection. In *Proceedings of the 2003 IEEE International Conference on Fuzzy Systems*. IEEE Press.

Heberlein, L. T., Dias, G. V., Levitte, K. N., Mukherjee, B., Wood, J., and Wolber, D. (1990). A network security monitor. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEE Press.

Helman, P. and Liepins, G. (1993). Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19(9):886–901.

Helmbold, D. P. and Long, P. M. (1991). TRACKING DRIFTING CONCEPTS BY MINIMIZING DISAGREEMENTS. *Machine Learning*, 14:27–45.

Hofmeyr, S. (1999). *An immunological model of distributed detection and its application to computer security*. PhD thesis, University of New Mexico, Albuquerque, NM.

Hofmeyr, S. and Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473.

Holland, J., Holyoak, K., Nisbett, R., and Thagard, P. (1986). *Induction: Processes of Inference, Learning, and Discovery*. MIT Press.

ISS (2000). *RealSecure Product Datasheet*. Internet Security Systems, Atlanta, GA. Available at `http://www.iss.net/customer_care/resource_center/product_lit/`.

Kim, J. and Bentley, P. (1999a). The artificial immune model for network intrusion detection. In *7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany.*

Kim, J. and Bentley, P. (1999b). Negative selection and niching by an artificial immune system for network intrusion detection. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 149–158.

Kim, J. and Bentley, P. J. (2001). An evaluation of negative seelection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1330–1337, San Francisco, CA. Morgan-Kauffman.

Kim, J. and Bentley, P. J. (2002a). Immune memory in the dynamic clonal selection algorithm. In Timmis, J. and Bentley, P. J., editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 59–67, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

Kim, J. and Bentley, P. J. (2002b). A model of gene library evolution in the dynamic clonal selection algorithm. In Timmis, J. and Bentley, P. J., editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, pages 182–189, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

Klinkenberg, R. and Joachims, T. (2000). Detecting concept drift with support vector machines. In Langley, P., editor, *Proceedings the 17th International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann.

Lane, T. D. (2000). *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, West Lafayette, IN.

McHugh, J. (2000). The 1998 Lincoln Laboratory IDS evaluation—a critique. In Debar, H., Me, L., and Wu, S., editors, *Recent Advances in Intrusion Detection. Third International Workshop, RAID 2000, 2–4 Oct. 2000, Toulouse, France*, pages 145–61, Berlin, Germany. Springer-Verlag.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, New Brunswick, New Jersey.

Mitchell, T. M., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D. (1994). Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91.

Muggleton, S. (1996). Learning from positive data. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 358–376. Springer-Verlag.

Mukherjee, B., Heberlein, L. T., and Levitt, K. N. (1994). Network intrusion detection. *IEEE Network*, pages 26–41.

NIST (2003). An overview of issues in testing intrusion detectin systems. NIST IR 7007.

Percus, J. K., Percus, O., and Perelson, A. S. (1993). Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695.

Pinker, S. (1984). *Language learnability and language development*. Harvard University Press.

Porras, P. and Neumann, P. G. (1997). Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the National Information Systems Security Conference*.

Pratt, L. and Jennings, B. (1997). A survey of connectionist network reuse through transfer. In Thrun, S. and Pratt, L., editors, *Learning to Learn*, chapter 2. Kluwer Academic Publishers.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1987). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L., et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge.

Smaha, S. E. (1988). Haystack: An intrusion detection system. In *Proceedings, IEEE Fourth Aerospace Computer Security Applications Conference*.

Squid. Squid web proxy cache. `http://www.squid-cache.org/`, Downloaded 2003.

Williams, P., Anchor, K., Bebo, J., Gunsch, G., and Lamont, G. (2001). Cdis: Towards a computer immune system for detecting network intrusions. In Lee, W., Mé, L., and Wespi, A., editors, *RAID 2001*, volume 2212 of *Lecture Notes in Computer Science*, pages 117–133, Berlin. Springer-Verlag.