Learning Program Behavior Profiles for Intrusion Detection*

Anup K. Ghosh, Aaron Schwartzbard & Michael Schatz
Reliable Software Technologies Corporation
21515 Ridgetop Circle, #250, Sterling, VA 20166
phone: (703) 404-9293, fax: (703) 404-9295
email: anup.ghosh@computer.org
www.rstcorp.com

Abstract

Profiling the behavior of programs can be a useful reference for detecting potential intrusions against systems. This paper presents three anomaly detection techniques for profiling program behavior that evolve from memorization to generalization. The goal of monitoring program behavior is to be able to detect potential intrusions by noting irregularities in program behavior. The techniques start from a simple equality matching algorithm for determining anomalous behavior, and evolve to a feed-forward backpropagation neural network for learning program behavior, and finally to an Elman network for recognizing recurrent features in program execution traces. In order to detect future attacks against systems, intrusion detection systems must be able to generalize from past observed behavior. The goal of this research is to employ machine learning techniques that can generalize from past observed behavior to the problem of intrusion detection. The performance of these systems is compared by testing them with data provided by the DARPA Intrusion Detection Evaluation program.

1 Introduction

Intrusion detection tools seek to detect attacks against computer systems by monitoring the behavior of users, networks, or computer systems. Intrusion detection techniques are the last line of defense against computer attacks behind secure network architecture design, secure program design, carefully configured network services, firewalls, penetration audits, and personnel screening. Attacks against computer systems are still largely successful despite the plethora of intrusion prevention techniques available. For instance, insider attacks and malicious mobile code have been able to penetrate most security defenses. Largely, however, most computer security attacks are made possible by poorly configured software or by buggy software.

Some of the first intrusion detection activities were performed by system administrators who examined audit logs of user and system events recorded by computer hosts. Activities such as super user login attempts, FTP transfers of sensitive files, or failed file accesses were flags for potential intrusive activity. Soon thereafter, expert systems were used to automatically detect potential attacks by scanning audit logs for signs of intrusive behavior or for departures from normal behavior. The Intrusion Detection Expert System (IDES) developed at SRI performed intrusion detection by creating statistical profiles for users and noting unusual departures from normal profiles [16]. IDES keeps statistics for each user according to specific intrusion detection measures, such as the number of files created and deleted each day. These statistics form the statistical profile of each user. The profiles are periodically updated to include the most recent changes to the user's profile. Therefore, this technique is adaptive with changing user profiles. However, it is also susceptible to a user slowly changing his or her profile to include possibly intrusive activities.

More recently, network-based intrusion detection tools have gained popularity among researchers and even in commercial tools. Network-based intrusion

*This work is sponsored under the Defense Advanced Research Projects Agency (DARPA) Contract DAAH01-98-C-R145. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

detection tools will typically search network data for signatures of known computer attacks. For example, network probing attacks, which map out the network topology of a site, can often be detected by their characteristic "pings" to the range of network services across many machines.

Today, there are generally two types of intrusion detection systems: anomaly detection and misuse detection. Anomaly detection approaches attempt to detect intrusions by noting significant departures from normal behavior [7, 5, 20, 18, 15, 17, 16]. Misuse detection techniques attempt to model attacks on a system as specific patterns, then systematically scan the system for occurrences of these patterns [22, 14, 10, 9, 19]. This process involves a specific encoding of previous behaviors and actions that were deemed intrusive or malicious.

It is important to establish the key differences between anomaly detection and misuse detection approaches. The most significant advantage of misuse detection approaches is that known attacks can be detected fairly reliably and with a low false positive rate. However, the key drawback of misuse detection approaches is that they cannot detect novel attacks against systems that leave different signatures. So while the false positive rate can be made extremely low, the rate of missed attacks (false negatives) can be extremely high depending on the ingenuity of the attackers. As a result, misuse detection approaches provide little defense against novel attacks, until they can learn to generalize from known signatures of attacks.

Anomaly detection techniques, on the other hand, directly address the problem of detecting novel attacks against systems. This is possible because anomaly detection techniques do not scan for specific patterns, but instead compare current activities against models of past behavior. One clear drawback of anomaly detection is its inability to identify the specific type of attack that is occurring. However, probably the most significant disadvantage of anomaly detection approaches is the high rates of false alarm. Because any significant deviation from the baseline can be flagged as an intrusion, it is likely that non-intrusive behavior that falls outside the normal range will also be labeled as an intrusion — resulting in a false positive. Another drawback of anomaly detection approaches is that if an attack occurs during the training period for establishing the baseline data, then this intrusive behavior will be established as part of the normal baseline. In spite of the potential drawbacks of anomaly detection, having the ability to detect novel attacks makes anomaly detection a requisite if future, unknown, and novel attacks against computer systems are to be detected.

In this paper, we consider three techniques for intrusion detection that are based on anomaly detection. Our primary goal in this work is to be able to detect novel attacks against systems, *i.e.*, attacks that have not been seen before by our intrusion detection system. Our secondary goal is to reduce the false positive rate, *i.e.*, the rate at which our system classifies normal behavior as intrusions. Our approach is to learn the normal behavior of programs (using different techniques) and then flag significant departures from normal behavior as possible intrusions. This approach is designed to achieve our primary goal of detecting novel attacks.

To achieve our secondary goal of reducing the false positive rate, our approach is to generalize from past observed behavior to inputs the system did not encounter during training. To this end, we have developed three algorithms that range in their ability from being able to simply memorize past events to being able to classify inputs previously unseen based on a similarity measure, to being able to recognize recurrent patterns. Before developing the three algorithms, we first present related work in program-based intrusion detection.

2 Analyzing Program Behavior for Anomaly Detection

Analyzing program behavior profiles for intrusion detection has recently emerged as a viable alternative to user-based approaches to intrusion detection (see [7, 21, 12, 5, 3, 6, 14] for other program-based approaches). Program behavior profiles are built by capturing system calls made by the program under analysis under normal operational conditions. If the captured behavior represents a compact and adequate signature of normal behavior, then the profile can be used to detect deviations from normal behavior such as those that occur when a program is being misused for intrusion.

One of the first groups to develop program-based intrusion detection was Stephanie Forrest's research group out of the University of New Mexico. Their work in [5, 6] established an analogy between the human immune system and intrusion detection. The approach consisted of using short sequences of system calls (called a string or N-gram) from the target program to the operating system to form a signature for normal behavior. A database of system calls is built for each monitored program by capturing system calls made by the program under normal usage conditions. The Linux program `strace` was used in their work to capture system calls.

Once constructed, the database essentially serves as the repository for self behavior against which all subsequent online behavior will be judged. If a string formed during the online operation of the program does not match a string in the normal database, a mismatch is recorded. If the number of mismatches detected are a significant percentage of all strings captured during the online session, then an intrusion is registered. The application of this technique was shown viable for Unix programs `sendmail`, `lpr`, and `ftpd`.

It was later recognized by a research group out of Columbia University [14] and by another research project at UNM [12] that program anomalies were temporally located in clusters. Thus, averaging the number of anomalies over the entire execution trace as performed in the UNM's earlier work could potentially "wash out" the intrusive behavior among normal variation in program behavior. Hence, the notion of fixed-length frames in which anomalies were to be counted was used in both groups' subsequent work.

The Columbia group applied a rule learning program (RIPPER [2]) to the data to extract rules for predicting whether a sequence of system calls is normal or abnormal. Because the rules made by RIPPER can be erroneous, a post-processing algorithm is used to scan the predictions made by RIPPER to determine if an intrusion has occurred or not. The post-processing algorithm uses the notion of temporal locality to filter spurious prediction errors from intrusions which should leave temporally co-located abnormal predictions. The results in [14] verified that system calls can be used to detect intrusions, even with different intrusion detection algorithms.

Subsequent work performed by the UNM group and reported in [12], applied fixed-length frames to the equality matching approach developed earlier in [6]. However, their work was further distinguished by their analysis of the structure of system calls made by the program. The empirical analysis found recurrent patterns of system calls in execution traces of any given program. For instance most programs have a prefix, a main portion, and a suffix. Within these portions, system calls tended to be repeated in a regular fashion. As a result, they hypothesized that a deterministic finite automaton (DFA) could be constructed to represent this behavior using a macro language. For each program, they manually selected macros that matched the pattern they believed to represent the normal behavior. Anomalies were then detected by applying the macros against the observed behavior and noting mismatches. However, because their technique involves creating DFAs heuristically and by hand, the technique will not scale well to real systems. Furthermore, an exact DFA representation of the program behavior could lead to a state explosion problem.

In a similar vein as the work of [12] in creating finite state automata, a group from Iowa State is implementing a program-based intrusion detection approach that analyzes system calls using state machine models of program behavior [21]. However, their approach is not concerned with detecting anomalies, as much as detecting violations of specified behavior. As a result, the approach of the Iowa State group requires the development of specification models for acceptable program behavior, where the work of [12, 14, 5, 6] used models of program behavior derived from empirical training. An auditing specification language (ASL) is used to develop a representation of expected or allowed program behavior based on specification models of programs; violations of this model are used to detect potential intrusions and isolate the program in question from privileged resources. This approach is similar to sandbox models of programs that constrain program behavior based on policies or models of acceptable program behavior [8, 11].

In this paper, we build upon the work of the UNM group in creating normal program behavior profiles from system calls and performing anomaly detection from these profiles. We present an evolution of techniques that begin from a table lookup equality matching approach (similar to the UNM work in [5]) to machine learning approaches that can generalize from past observed behavior. Our goal in applying the equality matching technique was to verify the feasibility and performance of the technique on a much larger scale than previously performed. Our approach was simply to improve on the equality matching technique where it was obvious improvements could be made.

In the equality matching approach, we use fixed-size frames to capture temporally co-located events similar to [14, 12]. However, unlike the approach in [12], our technique automatically builds profiles for programs and performs anomaly detection. No heuristics or hand coding of macros are necessary to do anomaly detection. We have been able to scale up our program-based anomaly detection approach significantly over previous studies [12, 14, 5] to monitor over 150 programs as part of the 1998 DARPA Intrusion Detection Evaluation program[1]. Hence the results presented here represent the first significant study of applying an equality matching technique for system calls to a realistic system in a comprehensive intrusion detection study.

One of the key drawbacks in using an equality matching approach in its current form is the inability to generalize from past observed behavior. Thus, if the normal program behavior is not adequately captured, future unseen normal behavior will be classified as anomalous, thus contributing to the false positive rate. Desiring the ability to reduce the false positive rate while still providing the ability to detect novel attacks consistently, we investigated machine learning approaches for learning program behavior. Neural networks were the best fit for learning associations between observed inputs and desired outputs. We implemented a standard backpropagation neural network (a feedforward multi-perceptron network) to be able to generalize from previously seen inputs to map future unseen inputs into normal or anomalous outputs. We tested our backpropagation networks against the same corpus of data provided by the DARPA evaluation program. The results show both the benefits and pitfalls of using backpropagation networks for this purpose.

While working with neural networks, we re-visited the input domain for our networks in order to develop a proper encoding function to the network. We noticed recurrent patterns of system calls in the execution traces of the programs similar to what Kosoresow et al. noted in [12]. Unlike the approach developed by Kosoresow et al., however, we were interested in automatically learning the behavior of the program that would be able to exploit the recurrent features in the data. Furthermore, we desired our learning algorithm to be able to generalize to recognize future, previously unseen behavior — unlike the equality matching algorithm. These requirements led us to the development of Elman networks. Elman networks use the sequential characteristics of the input data to learn to recognize sequentially related (or in our case temporally co-located) features of variable length. Hence, we applied the Elman networks to the DARPA evaluation data for anomaly detection.

The study presented in the rest of this paper is able to provide a side-by-side comparison of three different algorithms for anomaly detection that represent evolutions from pure memorization to generalization based on the recurrent characteristics of system calls made by programs. The results are significant because the data on which the algorithms are evaluated represents a significant corpus of scientifically controlled data by which the false positive rate of a given intrusion detection algorithm can be simultaneously measured against the correct detection rate. Hence, we are able to scientifically validate our approaches against a good set of data. In the rest of this paper, we describe the algorithms and the results from their implementation.

3 Equality Matching: A Simple Anomaly Detection Approach

The first approach we implemented built on the work of Forrest et al. [5, 12, 6]. But rather than using the `strace(1)` program on Linux for capturing system calls, we used Sun Microsystem's Basic Security Module (BSM) auditing facility for Solaris. This approach is practical because no special software need be written to capture system calls. The BSM events serve as an adequate representation of the behavior of the program for our purposes because any privileged calls that might be made by a program are captured by BSM. Furthermore, a program can only abuse system resources if it is making system calls. Our study also finds that out of approximately 200 different BSM events that can be recorded, programs typically make only 10 to 20 different BSM events. Therefore, capturing BSM events also serves as a compact representation of program behavior, while still leaving ample room to detect deviant behavior (through odd BSM events or odd sequences of BSM events). Finally, the BSM events we recorded for program executions showed

[1] See `www.ll.mit.edu/IST/ideval/index.html` for a summary of the program.

regular patterns of behavior such as a common beginning and ending sequence, as well as recurrent strings of system calls. Any anomaly detection algorithm will perform better when the entity it is monitoring has well-defined regular patterns of behavior. For all these reasons, in addition to the simplicity of the algorithm and the early success of the UNM group, we applied this algorithm with improvements to a large set of data to benchmark its success.

The equality matching algorithm is simple but effective. Sequences of BSM events are captured during online usage and compared against those stored in the database built from the normal program behavior profile. If the sequence of BSM events captured during online usage is not found in the database, then an anomaly counter is incremented. This technique is predicated on the ability to capture the normal behavior of a program in a database. If the normal behavior of a program is not adequately captured, then the false alarm rate is likely to be high. On the other hand, if the normal behavior profile built for a program includes intrusive behavior, then future instances of the intrusive behavior are likely to go undetected.

The data is partitioned into fixed-size windows in order to exploit a property of attacks that tends to leave its signature in temporally co-located events. That is, attacks tend to cause anomalous behavior to be recorded in clusters. Thus, rather than averaging the number of anomalous events recorded over the entire execution trace (which might wash out an attack in the noise), a much smaller size window of events is used for counting anomalous events.

Several counters are kept at varying levels of granularity ranging from a counter for each fixed window of system calls to a counter for the number of windows that are anomalous. Thresholds are applied at each level to determine at which point anomalous behavior is propagated up to the next level. Ultimately, if enough windows of system calls in a program are deemed anomalous, the program behavior during a particular session is deemed anomalous, and an intrusion detection flag is raised.

The equality matching algorithm was evaluated by MIT's Lincoln Laboratory under the DARPA 1998 Intrusion Detection Evaluation program. Unlabeled sessions were sent by Lincoln Labs and processed by our intrusion detection algorithm. These sessions had an unspecified number of attacks of the following four types: denial of service (DoS), probe, user to root (u2r), and remote to local (r2l). A user to root attack is defined as an attack that elevates the privilege of a user with local account privileges. Remote to local attacks grant a remote user with no account privileges to local user account privileges. Because this approach is mainly suited to u2r and r2l types of attacks, and because there were a statistically insignificant amount of DoS and probe attacks in the BSM data, we present results only from the u2r and r2l attacks.

Attack Type	**Instances**	**Detections**	**Percent Detected**
u2r	22	19	86.4
r2l	3	2	66.7
Total	25	21	84%

Table 1: **Performance of table look up intrusion detection algorithm against user to root (u2r) and remote to local (r2l) attacks.**

Table 1 shows the performance of the equality matching algorithm for detecting attacks at a particular threshold of sensitivity. If the threshold is set too low, then the false alarm rate will be low, but detection rate will be low, too. Similarly, a threshold set too high may end up detecting most intrusions, but suffer from a high false alarm rate. False alarm rates are not shown for these attacks because our algorithm will not label a particular attack — it only notes when an attack (any attack) is occurring. As a result, false positives cannot be tracked to particular attack types.

While the table is useful for quickly determining how many attacks of a particular type were detected, a more useful measure of the performance of the method can be obtained from Receiver Operating Characteristic (ROC) curves.

A measure of the overall effectiveness of a given intrusion detection system can be provided by the ROC curve. An ROC curve is a parametric curve that is generated by varying the threshold of the *intrusive measure*, which is a tunable parameter, and computing the probability of detection and the probability of false alarm at each operating point. The curve is a plot of the likelihood that an intrusion is detected, against the likelihood that a non-intrusion is misclassified (*i.e.*, a false positive) for a particular parameter, such as a tunable threshold. The ROC curve can be used to determine the performance of the system for different operating points

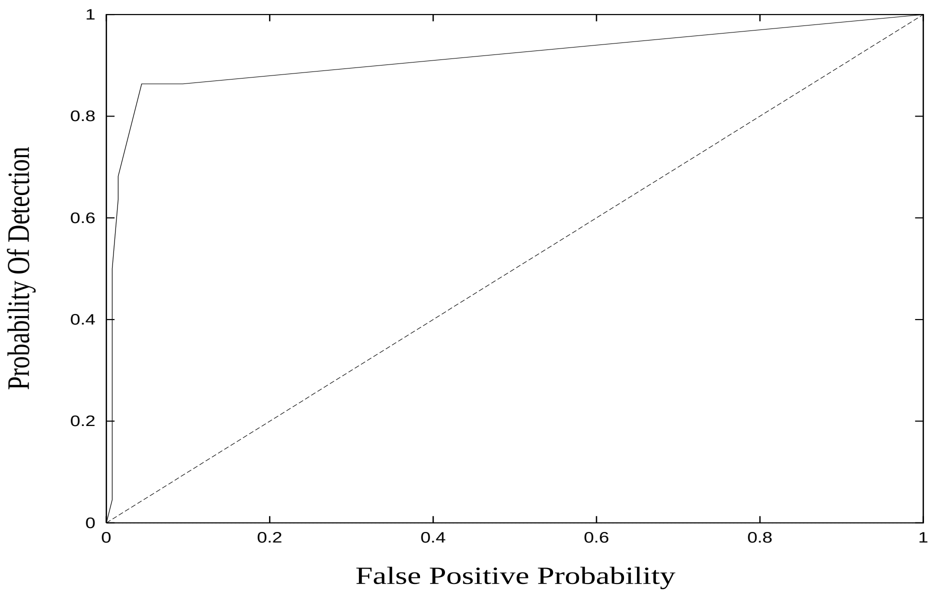

Figure 1: Performance of the equality matching technique as a function of false positive percentage (horizontal axis) and the correct detection percentage (vertical axis). This graph shows both the worst possible ROC curve (*i.e.*, $y = x$) as well as the ROC curve generated from actual data using the equality matching algorithm.

such as configurable thresholds, or for comparing the performance of different intrusion detection algorithms for given operating points.

Figure 1 shows performance of the equality matching algorithm as a ROC curve. To better understand this performance measure, consider an intrusion detection oracle that scores a session with a value of one if and only if it is an intrusion, and a value of zero otherwise. The resulting ROC curve would actually not be a curve, but rather, a single point at the location (0,1) since it is would detect intrusions with a likelihood of 1/1, and it would misclassify non-intrusions with a likelihood of 0/1. Further, as the threshold varied between zero and one (exclusive), there would be no change in the way sessions are classified, so the parametric value would remain at that one point. This can be called the *oracle point*. However, at the thresholds of 1 and 0 (inclusive), the (0,0) and (1,1) points remain fixed. Connecting these points and computing the area under the curve gives an area of 1, or a power of 100%.

At the other end of the spectrum, consider the curve that defines the worst possible intrusion detection system. The ROC curve for the worst case scenario is the $y = x$ line shown in Figure 1. Assume a system that randomly assigns a value between zero and one for every session. Starting from a threshold of zero, we derive the (1,1) point because all sessions would be classified as intrusions. As the session threshold increases, the likelihood of both correctly classifying an intrusion and incorrectly classifying a non-intrusion decrease at the same rate until the session threshold is 1 (corresponding to the point (0,0)). The power of this system is 50%, corresponding to the area under this curve of 0.5. If an intrusion detection system were to perform even worse than this curve, one would simply invert each classification to do better. Therefore, the $y = x$ plot represents the benchmark by which all intrusion detection systems should do better.

The results in Figure 1 for the equality matching algorithm represent an optimal tuning of the window (or frame) size to 20 and an N-gram size to six. These parameter values were found to be optimal through experimental analysis. The $y = x$ curve is shown as the benchmark for the worst case scenario. The equality matching method was able to detect 68.2% of all intrusions with a false positive rate of 1.4%. Higher detection rates could be achieved at the expense of more false positives. At a detection rate of 86.4%, the false positive rate rose to 4.3%. Similar curves are generated and compared for the two other intrusion detection approaches.

4 The Backpropagation Network

The goal in using neural networks for intrusion detection is to be able to generalize from incomplete data and to be able to classify online data as being normal or anomalous. Applying machine learning to intrusion detection has been developed elsewhere as well [4, 1, 13]. Lane and Brodley's work uses machine learning to distinguish between normal and anomalous behavior. However, their work is different from ours in that they build *user* profiles based on sequences of each individual's normal user commands and attempt to detect intruders based on deviations from the established user profile. Similarly, Endler's work [4] used neural networks to learn the behavior of users based on BSM events recorded from user actions. Rather than building profiles on a per-user basis, our work builds profiles of *software behavior* and attempts to distinguish between normal software behavior and malicious software behavior. The advantages of our approach are that vagaries of individual behavior are abstracted because program behavior rather than individual usage is studied. This can be of benefit for defeating a user who slowly changes his or her behavior to foil a user profiling system. It can also protect the privacy interests of users from a surveillance system that monitors a user's every move.

The goal in using artificial neural networks (ANNs) for intrusion detection is to be able to generalize from incomplete data and to be able to classify online data as being normal or intrusive. An artificial neural network is composed of simple processing units, or *nodes*, and connections between them. The connection between any two units has some *weight*, which is used to determine how much one unit will affect the other. A subset of the units of the network acts as *input nodes*, and another subset acts as *output nodes*. By assigning a value, or *activation*, to each input node, and allowing the activations to propagate through the network, a neural network performs a functional mapping from one set of values (assigned to the input nodes) to another set of values (retrieved from the output nodes). The mapping itself is stored in the weights of the network.

In this work, a classical feed-forward multi-layer perceptron network was implemented: a backpropagation neural network. The backpropagation network has been used successfully in other intrusion detection studies [7, 1]. The backpropagation network, or backprop, is a standard feed-forward network. Input is submitted to the network and the activations for each level of neurons are cascaded forward.

In order to train the networks, it is necessary to expose them to normal data and anomalous data. Randomly generated data were used to train the network to distinguish between normal and anomalous data. The randomly generated data, which were spread throughout the input space, caused the network to generalize that all data were anomalous by default. The normal data, which tended to be localized in the input space, caused the network to recognize a particular area of the input space as non-anomalous.

During training, many networks were trained for each program, and the network that performed the best was selected. The remaining networks were discarded. Training involved exposing the networks to four weeks of labeled data, and performing the backprop algorithm to adjust weights. An epoch of training consisted of one pass over the training data. For each network, the training proceeded until the total error made during an epoch stopped decreasing, or 1,000 epochs had been reached. Since the optimal number of hidden nodes for a program was not known before training, for each program, networks were trained with 10, 15, 20, 25, 30, 35, 40, 50, and 60 hidden nodes. Before training, network weights were initialized randomly. However, initial weights can have a large, but unpredictable, effect on the performance of a trained network. In order to avoid poor performance due to bad initial weights, for each program, for each number of hidden nodes, 10 networks were initialized differently, and trained. Therefore, for each program, 90 networks were trained. To select which of the 90 to keep, each was tested on two weeks of data which were not part of the four weeks of data used for training. The network that classified data most accurately was kept.

After training and selection, a set of neural networks was ready to be used. However, a neural network can only classify a single string (a sequence of BSM events) as anomalous or normal, and our intention was to classify entire sessions (which are usually composed of executions of multiple programs) as anomalous or normal. Furthermore, our previous experiments showed that it is important to capture the temporal locality of anomalous events in order to recognize intrusive behavior. As a result, we desired an algorithm that provides some memory of

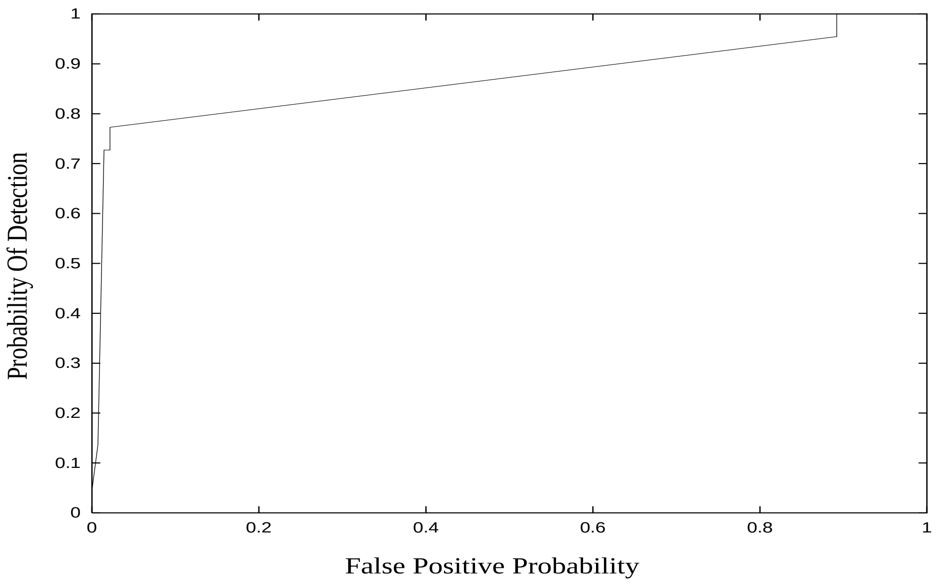

Figure 2: Performance of the backpropagation network expressed in a ROC curve. The horizontal axis represents the percentage of false positives while the vertical axis represents the percentage of correct detections for different operating thresholds of the technique.

recent events.

The leaky bucket algorithm fit this purpose well. The leaky bucket algorithm keeps a memory of recent events by accumulating the neural network's output, while slowly leaking its value. Thus, when the network computes closely related anomalies, the leaky bucket algorithm will quickly accumulate a large value in its counter. Similarly, as the network computes a normal output, the bucket will "leak" away its anomaly counter back down to zero. As a result, the leaky bucket emphasizes anomalies that are closely temporally co-located and diminishes the values of those that are sparsely located.

Strings of BSM events are passed to a neural network in the order they occurred during program execution. The output of a neural network—that is, the classification of the input string—is then placed into a leaky bucket. During each timestep, the level of the bucket is decreased by a fixed amount. If the level in the bucket rises above some threshold at any point during execution of the program, the program is flagged as anomalous. The advantage of using a leaky bucket algorithm is that it allows occasional anomalous behavior, which is to be expected during normal system operation, but it is quite sensitive to large numbers of temporally co-located anomalies, which one would expect if a program were really being misused. If a session contains a single anomalous program, the session is flagged as anomalous.

The performance of the IDS should by judged in terms of both the ability to detect intrusions, and by false positives—incorrect classification of normal behavior as intrusions. We used ROC curves to compare intrusion detection ability of the backpropagation network to false positives. The results from the backpropagation network are shown in Figure 2. The test data consisted of 139 non-intrusive sessions, and 22 intrusive sessions. Different leak rates from the leaky bucket algorithm produce different ROC curves. A leak rate of 0 results in all prior timesteps being retained in memory. A leak rate of 1 results in all timesteps but the current one being forgotten. We varied the leak rate from 0 to 1.

In Figure 2, the ROC curve is shown for a leak rate of 0.7. The curve and performance is similar to the equality matching algorithm results shown in Figure 1. A detection rate of 77.3% can be achieved with a false positive rate of 2.2%.

Purely feed-forward network topologies possess a major limiting characteristic. That characteristic is that the output produced by any input is independent of prior inputs. While this characteristic is appropriate for tasks which require processing of independent inputs, it is not optimal when the inputs are sequential elements of a *stream* of data. In the next section, we discuss an alternative network that can recognize recurrent features in the input.

5 Elman Networks

In this section, we motivate the reasons for using recurrent networks, then describe the Elman recurrent network used for anomaly detection. Results from applying the Elman network to the DARPA data are presented in comparison to the previous techniques.

The BSM events produced by a single program during a single execution can be considered to be a stream of events. That is, each event is part of an ordered series. A given portion of a program will typically generate similar sequences of BSM events during different executions. Since there is a limited number of ways in which a transition (or branch) from one portion of the program to another can occur, it is often possible to determine what sequence of events will follow the current sequence of events.

By using a feed-forward topology (with backpropagation learning rules), as described in the preceding section, we *train* ANNs to recognize whether small, fixed-sized sequences of events are characteristic of the programs in which they occur. For each sequence, an ANN produces an output value that represents how anomalous the sequence is (based on the training data). In addition, the leaky bucket algorithm used to classify the program behavior ensures that two highly anomalous sequences have a larger impact on the classification of a program if they are close together than if they are far apart. However, as determined by investigation of raw BSM data, the large-scale structure of a stream of BSM data has features that cannot be captured within individual sequences of lengths being used in our experiments.

In order to accommodate the large-scale structure of BSM features during a given execution trace, two options are apparent: 1) increase the size of individual sequences so that large-scale structures of the stream are represented within individual strings, or 2) use a system which maintains some degree of state between inputs. The first option will fail because in order to capture large-scale structures, individual sequences would necessarily be very large. As sequence sizes grow, so do the network and the difficulty in accurate classification.

The second alternative—to maintain state information between sequences—is more appealing. It allows the system to retain the generality of small sequences. It simply adds information concerning prior sequences. One possible way to maintain state information is through the use of a deterministic finite automaton (DFA). This approach was applied manually by a UNM group [12]. However, DFAs have several drawbacks. The primary drawback is the lack of flexibility. If the BSM stream briefly enters a state not represented in the DFA, the DFA cannot recover to recognize that the state was a slight aberration of the sort one would expect to encounter even during normal runs of a program. Thus, the DFA would need to be completely specified to represent all possible allowable sequences of BSM events, or a heuristic-based approach similar to the UNM approach would need to be adopted with its perils [12]. If the DFA is completely specified such that it represents enough states that no normal execution of a program produces states outside of the machine, then the machine will have represented so many of the target program's possible states that recognizing anomalous behavior may be difficult. Beyond the lack of flexibility of DFAs, it should be recognized that determining what constitutes a *state* of a program (and should be represented in the DFA) can be a difficult task. While neither of these issues is insurmountable, ANNs address each of them quite naturally.

We originally employed ANNs because of their ability to *learn* and *generalize*. Through the learning process, they develop the ability to classify inputs from exposure to a set of *training inputs* and application of well defined *learning rules*, rather than through an explicit human-supplied enumeration of classification rules. Because of their ability to generalize, ANNs can produce reasonable classifications for novel inputs (assuming the network has been trained well). Further, since the inputs to any node of the ANN used for this work could be any real-valued number, no sequence of BSM events could produce an encoding that would fall outside of the domain representable by the ANN.

In order to maintain state information between inputs, we required a recurrent ANN topology. A recurrent topology (as opposed to a purely feed-forward topology) is one in which cycles are formed by the connections. The cycles act as delay loops—causing information to be retained indefinitely. New

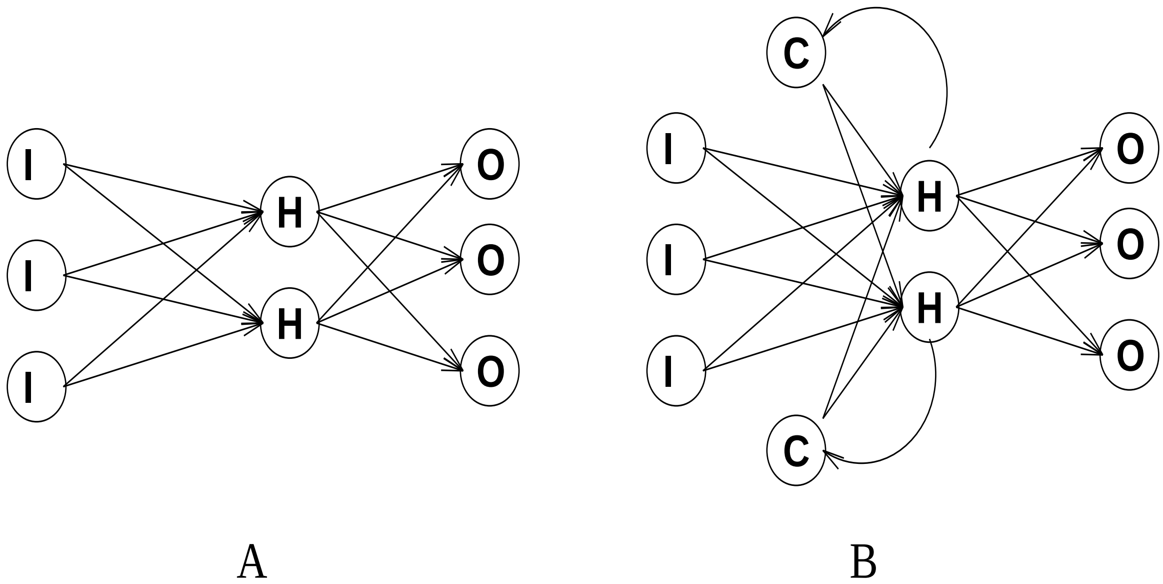

Figure 3: In each of the examples above, the nodes of the ANNs are labeled as input nodes (I), hidden nodes (H), output nodes (O), or context nodes (C). Each arc is unidirectional, with direction indicated by the arrow at the end of the arc. A) A standard feed-forward topology. B) An Elman network.

input interacts with the cycles, both the activations propagating through the network and the activations in the cycle are affected. Thus, the input can affect the state, and the state can affect the classification of any input.

One well known recurrent topology is that of an Elman network, developed by Jeffrey Elman. An Elman network is illustrated in Figure 3. The Elman topology is based on a feed-forward topology—it has an *input layer*, an *output layer*, and one or more *hidden layers*. Additionally, an Elman network has a set of *context nodes*. Each context node receives input from a single hidden node and sends its output to each node in the layer of its corresponding hidden node. Since the context nodes depend only on the activations of the hidden nodes from the previous input, the context nodes retain state information between inputs.

Because an Elman network retains information concerning previous inputs, the method used to train purely feed-forward ANNs to perform anomaly detection (see Section 4) will not suffice. We employ Elman nets to perform classification of short sequences of events as they occur in a larger stream of events. Therefore, we train our Elman networks to *predict* the next sequence that will occur at any point in time. The nth input, I_n, is presented to the network to produce some output, O_n. The output O_n is then compared to I_{n+1}. The difference between O_n and I_{n+1} (that is, the sum of the absolute values of the differences of the corresponding elements of O_n and I_{n+1}) is the measure of anomaly of each sequence of events. We continue to use the leaky bucket algorithm that causes anomalies to have a larger effect when they occur closer together than when they occur farther apart. However, the classification of a sequence of events will now be affected by events prior to the earliest event occurring within the sequence.

We implemented an Elman net and applied it for anomaly detection against the same set of DARPA evaluation data. Despite being the least extensively tuned of the three methods employed, the Elman nets produced the best results overall. The performance of the Elman nets in comparison to the equality matching (table lookup) technique and the backpropagation network is shown in Figure 4. The Elman ROC curve is the left-most curve that quickly reaches 100% detection. With a leak rate of 0.7, the Elman networks were able to detect 77.3% of all intrusions with no false positives — a very significant improvement over the other algorithms. Further, the Elman nets were able to detect 100.0% of all intrusions with significantly fewer false positives than either of the other two systems.

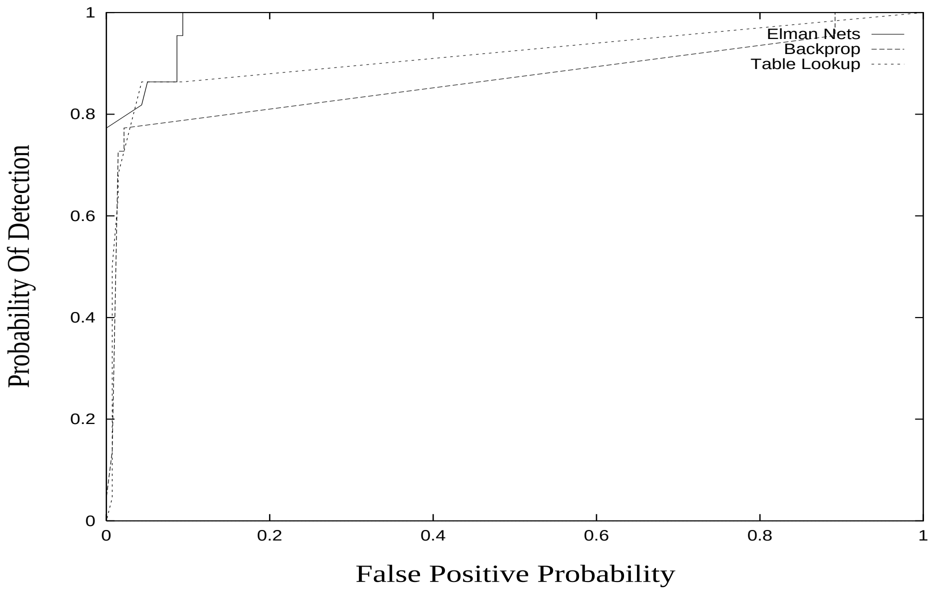

Figure 4: Performance of three anomaly detection algorithms expressed as ROC curves against the DARPA evaluation data. The horizontal axis represents the percentage of false positives while the vertical axis represents the percentage of correct detections for different operating thresholds of the technique.The Elman network performs the best overall.

6 Conclusions

This paper presented three different anomaly detection algorithms for detecting potential intrusions by using program behavior profiles. The algorithms range from pure memorization using an equality matching approach to the ability to generalize, to the ability to recognize recurrent features in the input. The results show that though the equality matching approach worked fairly well, the performance can be significantly improved (particularly in reducing the false positive rate) by using Elman networks.

References

[1] J. Cannady. Artificial neural networks for misuse detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98)*, pages 443–456, October 5-8 1998. Arlington, VA.

[2] W.W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann, 1995.

[3] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Security and Privacy*, 1996.

[4] D. Endler. Intrusion detection: Applying machine learning to solaris audit data. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98)*, pages 268–279, Los Alamitos, CA, December 1998. IEEE Computer Society, IEEE Computer Society Press. Scottsdale, AZ.

[5] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, October 1997.

[6] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE, May 1996.

[7] A.K. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98)*, December 1998.

[8] I. Goldberg, D. Wagner, R. Thomas, and E.A. Brewer. A secure environment for untrusted helper applications: Confining the

wiley hacker. In *Proceedings of the 1996 Usenix Security Symposium.* USENIX, July 22-25 1996.

[9] K. Ilgun. Ustat: A real-time intrusion detection system for unix. Master's thesis, Computer Science Dept, UCSB, July 1992.

[10] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State transition analysis: A rule-based intrusion detection system. *IEEE Transactions on Software Engineering*, 21(3), March 1995.

[11] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *10th Annual Computer Security Application Conference*, pages 134–144, December 1994. Orlando, FL.

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

[13] T. Lane and C.E. Brodley. An application of machine learning to anomaly detection. In *Proceedings of the 20th National Information Systems Security Conference*, pages 366–377, October 1997.

[14] W. Lee, S. Stolfo, and P.K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management*, 1997.

[15] T.F. Lunt. Ides: an intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*, November 1990. Rome, Italy.

[16] T.F. Lunt. A survey of intrusion detection techniques. *Computers and Security*, 12:405–418, 1993.

[17] T.F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[18] T.F. Lunt, A. Tamaru, F. Gilham, R. Jagannthan, C. Jalali, H.S. Javitz, A. Valdos, P.G. Neumann, and T.D. Garvey. A real-time intrusion-detection expert system (ides). Technical Report, Computer Science Laboratory, SRI Internationnal, February 1992.

[19] P.A. Porras and R.A. Kemmerer. Penetration state transition analysis - a rule-based intrusion detection approach. In *Eighth Annual Computer Security Applications Conference*, pages 220–229. IEEE Computer Society Press, November 1992.

[20] P.A. Porras and P.G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, October 1997.

[21] R. Sekar, Y. Cai, and M. Segal. A specification-based approach for building survivable systems. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98)*, pages 338–347, October 1998.

[22] G. Vigna and R.A. Kemmerer. Netstat: A network-based intrusion detection approach. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98)*, pages 25–34, Los Alamitos, CA, December 1998. IEEE Computer Society, IEEE Computer Society Press. Scottsdale, AZ.