A Study in Using Neural Networks for Anomaly and Misuse Detection*

Anup K. Ghosh & Aaron Schwartzbard
Reliable Software Technologies
21351 Ridgetop Circle, Suite 400
Dulles, VA 20166
poc:aghosh@rstcorp.com
http://www.rstcorp.com

Abstract

Current intrusion detection systems lack the ability to generalize from previously observed attacks to detect even slight variations of known attacks. This paper describes new process-based intrusion detection approaches that provide the ability to generalize from previously observed behavior to recognize future unseen behavior. The approach employs artificial neural networks (ANNs), and can be used for both anomaly detection in order to detect novel attacks and misuse detection in order to detect known attacks and even variations of known attacks. These techniques were applied to a large corpus of data collected by Lincoln Labs at MIT for an intrusion detection system evaluation sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA). Results from applying these techniques for both anomaly and misuse detection against the DARPA evaluation data are presented.

1 Introduction

Results from a recent U.S. Defense Advanced Research Projects Agency (DARPA) study highlight the strengths and weaknesses of current research approaches to intrusion detection. The DARPA scientific study is the first of its kind to provide independent third party evaluation of intrusion detection tools against such a large corpus of data. The findings from this study indicate that a fundamental paradigm shift in intrusion detection research is necessary to provide reasonable levels of detection against novel attacks and even variations of known attacks. Central to this goal is the ability to generalize from previously observed behavior to recognize future unseen, but similar behavior. To this end, this paper describes a study in using neural networks for both anomaly detection and misuse detection.

Research in intrusion detection research has begun to shift from analyzing user behavior to analyzing process behavior. Initial work in analyzing process behavior has already shown promising results in providing very high levels of detection against certain classes of attacks. In particular, process-based anomaly detection approaches have shown very good performance against novel attacks that result in unauthorized local access and attacks that result in elevated privileges — a vulnerable area for most intrusion detection tools [6]. In spite of the good detection capability of process-based anomaly detection approaches, the results indicate high rates of false alarms that can make these tools unusable for the practical security administrator. Current wisdom is that false alarm rates must be reduced to the level of one to two false alarms per day in order to make the system usable by administrators.

One of the largest challenges for today's intrusion detection tools is being able to generalize from previously observed behavior (normal or malicious) to recognize similar future behavior. This problem is acute for signature-based misuse detection approaches, but also plagues anomaly detection tools that must be able to recognize future normal behavior that is not identical to past observed behavior,

*This work was funded by the Defense Advanced Research Projects Agency (DARPA) under Contract DAAH01-97-C-R095. 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.

in order to reduce false positive rates.

To address this shortcoming, we utilize a simple neural network that can generalize from past observed behavior to recognize similar future behavior. In the past, we have applied backpropagation networks in addition to other neural networks with good performance to the problem of anomaly detection [8]. Here we present using a neural network for both anomaly and misuse detection. The approach is evaluated against the DARPA intrusion detection evaluation data.

2 Prior Art in Intrusion Detection

Some of the earliest work in intrusion detection was performed by Jim Anderson in the early 1980s [1]. Anderson defines an intrusion as any unauthorized attempt to access, manipulate, modify, or destroy information, or to render a system unreliable or unusable. Intrusion detection attempts to detect these types of activities. In this section we establish the foundations of intrusion detection techniques in order to determine where they are strong and where they need improvement.

2.1 Anomaly detection vs. misuse detection

Intrusion detection techniques are generally classified into two categories: anomaly detection and misuse detection. Anomaly detection assumes that misuse or intrusions are highly correlated to abnormal behavior exhibited by either a user or the system. Anomaly detection approaches must first baseline the normal behavior of the object being monitored, then use deviations from this baseline to detect possible intrusions. The initial impetus for anomaly detection was suggested by Anderson in his 1980 technical report when he noted that intruders can be detected by observing departures from established patterns of use for individual users. Anomaly detection approaches have been implemented in expert systems that use rules for normal behavior to identify possible intrusions [15], in establishing statistical models for user or program profiles [6, 4, 22, 19, 16, 18, 17], and in using machine learning to recognize anomalous user or program behavior [10, 5, 2, 14].

Misuse detection techniques attempt to model attacks on a system as specific patterns, then systematically scan the system for occurrences of these patterns. This process involves a specific encoding of previous behaviors and actions that were deemed intrusive or malicious. The earliest misuse detection methods involved off-line analysis of audit trails normally recorded by host machines. For instance, a security officer would manually inspect audit trail log entries to determine if failed root login attempts were recorded. Manual inspection was quickly replaced by automated analysis tools that would scan these logs based on specific patterns of intrusion. Misuse detection approaches include expert systems [15, 3], model-based reasoning [13, 7], state transition analysis [23, 12, 11, 21], and keystroke dynamics monitoring [20, 13]. Today, the vast majority of commercial and research intrusion detection tools are misuse detection tools that identify attacks based on attack signatures.

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. Since specific attack sequences are encoded into misuse detection systems, it is very easy to determine exactly which attacks, or possible attacks, the system is currently experiencing. If the log data does not contain the attack signature, no alarm is raised. As a result, the false positive rate can be reduced very close to zero. 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 statistical models of past behavior. Any activity sufficiently deviant from the model will be flagged as anomalous, and hence considered as a possible attack. Furthermore, anomaly detection schemes are based on actual user histories and system data to create its internal models rather than pre-defined patterns. Though anomaly detection approaches are powerful in that they can detect novel attacks, they have their drawbacks as well. For instance, 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, 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.

2.2 Assessing the Performance of Current IDSs

In 1998, the U.S. Defense Advanced Research Projects Agency (DARPA) initiated an evaluation of its intrusion detection research projects.[1] To date, it is the most comprehensive scientific study known for comparing the performance of different intrusion detection systems (IDSs). MIT's Lincoln Laboratory set up a private controlled network environment for generating and distributing sniffed network data and audit data recorded on host machines. Network traffic was synthesized to replicate normal traffic as well as attacks seen on example military installations. Because all the data was generated, the laboratory has *a priori* knowledge of which data is normal and which is attack data. The simulated network represented thousands of internal Unix hosts and hundreds of users. Network traffic was generated to represent the following types of services: http, smtp, POP3, FTP, IRC, telnet, X, SQL/telnet, DNS, finger, SNMP, and time. This corpus of data is the most comprehensive set known to be generated for the purpose of evaluating intrusion detection systems and represents a significant advancement in the scientific community for independently and scientifically evaluating the performance of any given intrusion detection system.

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

TCP/IP data was collected using a network sniffer and host machine audit data was collected using Sun Microsystem's Solaris Basic Security Module (BSM). In addition, dumps of the file system from one of the Solaris hosts were provided. This data was distributed to participating project sites in two phases: training data and test data. The training data is data labeled as normal or attack and is used by the participating sites to train their respective intrusion detection systems. Once trained, the test data is distributed to participating sites in unlabeled form. That is, the participating sites do not know *a priori* which data in the test data is normal or attack. The data is analyzed off-line by the participating sites to determine which sessions are normal and which constitute intrusions. The results were sent back to MIT's Lincoln Labs for evaluation.

The attacks were divided into four categories: denial of service, probing/surveillance, remote to local, and user to root attacks. Denial of service attacks attempt to render a system or service unusable to legitimate users. Probing/surveillance attacks attempt to map out system vulnerabilities and usually serve as a launching point for future attacks. Remote to local attacks attempt to gain local account privilege from a remote and unauthorized account or system. User to root attacks attempt to elevate the privilege of a local user to root (or super user) privilege. There were a total of 114 attacks in 2 weeks of test data including 11 types of DoS attacks, 6 types of probing/surveillance attacks, 14 types of remote to local attacks, 7 types of user to root attacks, and multiple instances of all types of attacks.

The attacks in the test data were also categorized as old versus new and clear versus stealthy. An attack is labeled as old if it appeared in the training data and new if it did not. When an attempt was made to veil an attack, it was labeled as stealthy, otherwise it was labeled as clear.

The reason we present this evaluation study is because we believe it to represent the true state of the art in intrusion detection research. As such, it represents the foundation of more than 10 years of intrusion detection research upon which all future work in intrusion detection should improve. From this study, we can learn the strengths of current intrusion detection approaches, and more importantly, their weaknesses. Rather than identifying which systems performed well and which did not, we simply summarize the results of the overall best combination system.

Lincoln Laboratory reported that if the best performing systems against all four categories of attacks were combined into a single system, then roughly between 60 to 70 percent of the attacks would have been detected with a false positive rate of lower than 0.01%, or lower than 10 false alarms a day. This result summarizes the combination of best systems against all of the attacks simulated in the data. It shows that even in the best case scenario over 30% of the simulated attacks would go by undetected. However, the good news is that the false alarm rate is acceptably low — low enough that the techniques can scale well to large sites with lots of traffic. The bad news is that with over 30% of attacks going undetected in the best combination of current intrusion detection systems, the state of the art in intrusion detection does not adequately address the threat of computer-based attacks.

Further analysis showed that most of the systems reliably detected old attacks that occurred within the training data with low false alarm rates. These results apply primarily to the network-based intrusion detection systems that processed the TCP/IP data. This result is encouraging, but not too surprising since most of the evaluated systems were network-based misuse detection systems. The results were mixed in detecting new attacks. In two categories of attacks, probing/surveillance and user to root attacks, the performance in detecting new attacks was comparable to detecting old attacks. In the other two categories — denial of service and remote to local attacks — the performance of the top three network-based intrusion systems was roughly 20% detection for new denial of service attacks and less than 10% detection for new remote to local attacks. Thus, the results show that the best of today's network-based intrusion detection systems do not detect novel denial of service attacks nor novel remote to local attacks — arguably two of the most concerning types of attacks against computer systems today.

3 Monitoring Process Behavior for Intrusion Detection

In the preceding section, intrusion detection methods were categorized into either misuse detection or anomaly detection approaches. In addition, intrusion detection tools can be further divided into network-based or host-based intrusion detection. The distinction is useful because network-based intrusion detection tools usually process completely different data sets and features than host-based intrusion detection. As a result, the types of attacks that are detected with network-based intrusion detection tools are usually different than host-based intrusion detection tools. Some attacks can be detected by both network-based and host-based IDSs, however, the "sweet spots", or the types of attacks each is best at detecting, are usually distinct. As a result, it is difficult to make direct comparisons between the performance of a network-based IDS and a host-based IDS. A useful corollary of distinct sweetspots, though, is that in combination both techniques are more powerful than either one by itself.

Recent research in intrusion detection techniques has shifted the focus from user-based intrusion detection to process-based intrusion detection. Process-based monitoring intrusion detection tools analyze the behavior of executing processes for possible intrusive activity. The premise of process monitoring for intrusion detection is that most computer security violations are made possible by misusing programs. When a program is misused its behavior will differ from its normal usage. Therefore, if the behavior of a program can be adequately captured in a compact representation, then the behavioral features can be used for intrusion detection.

Two possible approaches to monitoring process behavior are: instrumenting programs to capture their internal states or monitoring the operating system to capture external system calls made by a program. The latter option is more attractive in general because it does not require access to source code for instrumentation. As a result, analyzing external system calls can be applied to commercial off the shelf (COTS) software directly. Most modern day operating systems provide built-in instrumentation hooks for capturing a particular process's system calls. On Linux and other variants of Unix, the `strace(1)` program allows one to observe system calls made by a monitored process as well as their return values. On Sun Microsystem's Solaris operating system, the Basic Security Module (BSM) produces an event record for individual processes. BSM recognizes 243 built-in system signals that can be made by a process. Thus, on Unix systems, there is good built-in support for tracing processes' externally observable behavior. Windows NT currently lacks a built-in auditing facility that provides such fine-grain resolution of program behavior.

Most process-based intrusion detection tools are based on anomaly detection. A normal profile for program behavior is built during the training phase of the IDS by capturing the program's system calls during normal usage. During the detection phase, the profile of system calls captured during on-line usage is compared against the normal profile. If a significant deviation from the normal profile is noted, then an intrusion flag is raised.

Early work in process monitoring was pioneered by Stephanie Forrest's research group out of the University of New Mexico. This group uses the analogy of the human immune system to develop intrusion detection models for computer programs. As in the human immune system, the problem of anomaly detection can be characterized as the problem of distinguishing between self and dangerous non-self [6]. Thus, the intrusion detection system needs to build an adequate profile of self behavior in order to detect dangerous behavior such as attacks. Using `strace(1)` on Linux, the UNM group analyzed short sequences of system calls made by programs to the operating system [6].

More recently, a similar approach was employed by the authors in analyzing BSM data provided under the DARPA 1998 Intrusion Detection Evaluation program [9]. The study compiled normal behavior profiles for approximately 150 programs. The profile for each program is stored in a table that consists of short sequences of system calls. During on-line testing, short sequences of system calls captured by the BSM auditing facility are looked up in the table. This approach is known as equality matching. That is, if an exact match of the sequence of system calls captured during on-line testing exists in the program's table, then the behavior is considered normal. Otherwise an anomaly counter is incremented.

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 groups. 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 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 results from the study showed a high rate of detection, if not a low false positive rate [9]. Despite the simplicity of the approach and the high levels of detection, there are two main drawbacks to the equality matching approach: (1) large tables of program behavior must be built for each program, and (2) the equality matching approach does not have the ability to recognize behavior that is similar, but not identical to past behavior. The first problem becomes an issue of storage requirements for program behavior profiles and is also a function of the number of programs that must be monitored. The second problem results from the inability of the algorithm to generalize from past observed behavior. The problem is that behavior that is normal, yet slightly different from past recorded behavior, will be recorded as anomalous. As a result, the false positive rate could be artificially elevated. Instead, it is desirable to be able to recognize behaviors that are similar to normal, but not necessarily identical to past normal behavior as normal. Likewise, the same can be said for a misuse detection system. Many misuse detection systems are trained to recognize attacks based on exact signatures. As a result, slight variations among a given attack can result in missed detections, leading to a lower detection rate. It is desirable for misuse detection systems to be able to generalize from past observed attacks to recognize future attacks that are similar.

To this end, the research described in the rest of the paper employs neural networks to generalize from previously observed behavior. We develop an anomaly detection system that uses neural networks to learn normal behavior for programs. The trained network is then used to detect possibly intrusive behavior by identifying significant anomalies. Similarly, we developed a misuse detection system to learn the behavior of programs under attack scenarios. This system is then used to detect future attacks against the system. The goal of these approaches is to be able to recognize known attacks and detect novel attacks in the future. By using the associative connections of the network, we can generalize from past observed behavior to recognize future similar behavior. A comparison of the two systems against the DARPA intrusion data is provided in Section 5.

4 Using Neural Networks for Intrusion Detection

Applying machine learning to intrusion detection has been developed elsewhere as well [5, 2, 14]. 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 [5] 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 [10, 2]. The backpropagation network, or backprop, is a standard feedforward network. Input is submitted to the network and the activations for each level of neurons are cascaded forward.

Our previous research in intrusion detection with BSM data used an equality matching technique to look up currently observed program behavior that had been previously stored in a table. While the results were encouraging, we also realized that the equality matching approach had no possibility of generalizing from previously observed behavior. As a result, we are pursuing research in using artificial neural networks to accomplish the same goals, albeit with better performance. Specifically, we are interested in the capability of ANNs to generalize from past observed behavior to detect novel attacks against systems. To this end, we constructed two different ANNs: one for anomaly detection and one for misuse detection.

To use the backprop networks, we had to address five major issues: how to encode the data for input to the network, what network topology should be used, how to train the networks, how to perform anomaly detection with a supervised training algorithm, and what to do with the data produced by the neural network.

Encoding the data to be used with the neural network is in general, a difficult problem. Previous experiments indicated that strings of six consecutive BSM events carried enough implicit information to be accurately distinguished as anomalous or normal for programs in general. One possible encoding technique was simply to enumerate all observed strings of six BSM events, and use the enumeration as an encoding. However, part of the motivation of using neural nets was their ability to classify novel inputs based on similarity to known inputs. A simple enumeration will fail to capture information about the strings. Therefore, a neural net will be less likely to be able to correctly classify novel inputs. In order to capture the necessary information in the encoding, we devised a distance metric for strings of events. The distance metric took into account the events common to two strings, as well as the difference in positions of common events. To encode a string of data, the distance metric was used to measure the distance from the data string to each of several "exemplar" strings. The encoding then consisted of a set of measured distances. A string could then be thought of as a point in a space where each dimension corresponded to one of the exemplar strings, and the point is mapped in the space by plotting the distance from each dimension.

Once an appropriate encoding method was developed, an appropriate network topology must be employed. We had to determine how many input and output nodes were necessary, and if a hidden layer was to be used, how many nodes should it contain. Because we seek to determine whether an input string is anomalous or normal, we use a single continuously valued output node to represent the extent to which the network believes the input is normal or anomalous. The more anomalous the input is, the closer to 1.0 the network computes its output. Conversely, the closer to normal the input is, the closer to 0.0, the output node computes.

The number of input nodes has to be equal to the number of exemplar strings (since each exemplar produced a distance for input to the network). With an input layer, a hidden layer, and an output layer, a neural network can be constructed to compute any arbitrarily complex function. Thus, a single hidden layer was used in our networks. A different network must be constructed, tuned, and trained for each program to be monitored, since what might have been quite normal behavior for one program might have been extremely rare in another. The number of hidden nodes varied based on the performance of each trained network.

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 that were not part of the four weeks of data used for training. The network that classified data most accurately was kept.

4.1 Anomaly detection

In order to train the networks, it is necessary to expose them to normal data and anomalous data. Randomly generated data was 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.

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 recent events.

The leaky bucket algorithm fits this purpose well. The leaky bucket algorithm keeps a memory of recent events by incrementing a counter of the neural network's output, while slowly leaking its value. Thus, as the network computes many 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 the 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 execution of a program, the session is flagged as anomalous.

4.2 Misuse detection

Having developed a system for anomaly detection, we chose to evaluate how well the same techniques could be applied to misuse detection. Our system is designed to recognize some *type* of behavior. Thus, it should not matter whether the behavior it is learning was normal system usage, or attack behavior. Aside from trivial changes to the way the leaky bucket is monitored, our system should not require any modification to perform misuse detection. Having made the trivial modification to the leaky bucket, we tested our system as a misuse detector.

Unfortunately, two issues particular to our data-set made misuse detection difficult. The first issue was a lack of data. In the DARPA data, there was between two to three orders in magnitude less intrusion data than normal data. This made it quite difficult to train networks to learn what constituted an attack. The second issue was related to the labeling of intrusions. Intrusion data were labeled on a session-by-session basis. Whereas several programs might be executed during an intrusive session, as few as one might be anomalous. Thus, while all data labeled non-intrusive could be assumed to be normal, not all data labeled intrusive could be assumed to be anomalous. Despite these stumbling blocks, we configured our neural network system for misuse detection.

5 Experimental Results

The anomaly and misuse detection systems were tested on the same test data. The test data consisted of 139 non-intrusive sessions, and 22 intrusive sessions. Although it would have been preferable to use a larger number of intrusive sessions for testing, there were so few intrusive sessions in the DARPA data that all other intrusion data were used to train the misuse detection system.

The performance of any intrusion detection system must account for both the detection ability and the false positive rate. We observed both of these factors while varying the leak rate used by the leaky bucket algorithm. 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.

The performance of the IDS should by judged in terms of both the ability to detect intrusions, and by false positives—incorrect classification of secure behavior as insecure. We used receiver operating characteristic (ROC) curves to compare intrusion detection ability to false positives. A ROC curve is a parametric plot, where the parameter is the sensitivity of the system to what it perceives to be insecure behavior. The curve is a plot of the likelihood that an intrusion is detected, against the likelihood that a non-intrusion is misclassified for a particular parameter, such as a threshold. The ROC curve can be used to determine the performance of the system for any possible operating point. The ROC curve allows the end user of an intrusion detection system to assess the trade-off between detection ability and false alarm rate in order to properly tune the system for acceptable tolerances.

Different leak rates produced different ROC curves. Figure 1 displays two ROC curves—one for a low leak rate, and one for a high leak rate. For the leak rate of .2, to achieve detection better than 77.3%, one must be willing to accept a dramatic increase in false positives. At 77.3% detection, the false positive rate is only 3.6%. When the leak rate is .7, a detection rate of 77.3% can be achieved with a false positive rate of only 2.2%.

ROC curves were also produced for the performance of our misuse detection system. While the performance was not nearly as good as the anomaly detection system in terms of false positives (which was a high as 5% for even low sensitivity rates), the misuse detection system displayed very high detection abilities—especially surprising due to the small number of sessions used to train the system. As illustrated in Figure 2, with a leak rate of 0.7, the sys-

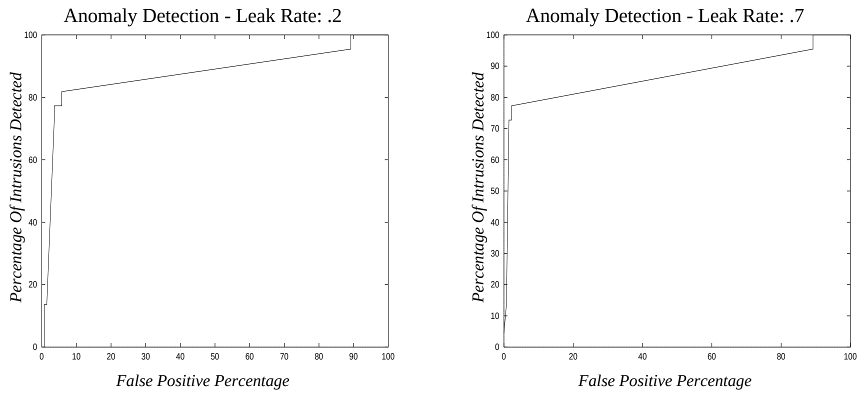

Figure 1: Anomaly detection results for two different leak rates.

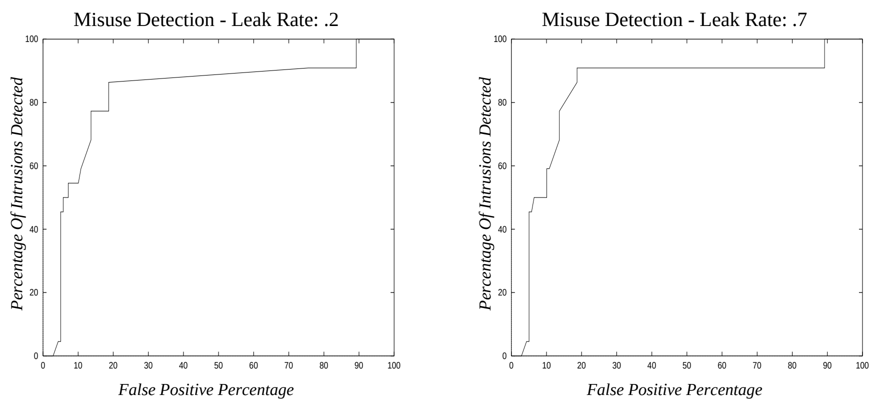

Figure 2: Misuse detection results for two different leak rates.

tem was able to detect as much as 90.9% of all intrusions with a false positive rate of 18.7%. Other host-based misuse detection systems can currently provide similar detection capabilities with lower false positive rates. Thus, this approach to misuse detection may not be suitable for detecting attacks in comparison to signature-based approaches. However, our technique demonstrated the ability of the system to detect novel attacks by generalizing from previously observed behavior.

While that false positive rate is clearly unacceptable, it should be remembered that the misuse detection system was trained on data which contained not only intrusion data, but also normal data. This would naturally lead this system to produce a large amount of false positives. By eliminating the non-intrusion data from the training data, it is believed that significantly lower false positive rates could be achieved, without lowering the detection ability.

The results of our experiments indicate that neural networks are suited to perform intrusion detection and can generalize from previously observed behavior. Currently, the false positive rates are too high to be practical for commercial users. In order to be a useful tool, false positive rates need to be between one and three orders of magnitude smaller. We continue to investigate how to improve the performance of our neural networks. Tests of variations on our techniques indicate that we have not yet achieved optimal performance.

6 Conclusions

This paper began with an examination of current intrusion detection systems. In particular, the DARPA 1998 intrusion detection evaluation study found that novel attacks against systems are rarely detected by most IDSs that use signatures for detection. On the other hand, well-known attacks for which signatures from network data or host-based intrusion data can be formed, perform very reliably with low rates of false alarms. However, even slight variations of known attacks escape detection by signature-based IDSs. Similarly, program-based anomaly detection systems have performed very well in detecting novel attacks, albeit with high false alarm rates. To overcome the problems in current misuse detection and anomaly detection approaches, a key necessity for IDSs is the ability to generalize from previously observed behavior to recognize future similar behavior. This capbility will permit detection of variations of known attacks as well as reduce false positive rates for anomaly-based IDSs.

In this paper, we presented the application of a simple neural network to learning previously observed behavior in order to detect future intrusions against systems. The results from our study show the viability of our approach for detecting intrusions. Future work will apply other neural networks more suited toward the problem domain of analyzing temporal characteristics of program traces. For instance, applying recurrent, time delay neural networks to program-based anomaly detection has proved to be more successful than using backpropagation networks for the same purpose [8]. Our next step is to apply these networks to misuse detection as well.

References

[1] J.P. Anderson. Computer security threat monitoring and surveillance. Technical Report Technical Report, James P. Anderson Co., Fort Washington, PA, April 1980.

[2] 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.

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

[4] 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.

[5] 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.

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

[7] T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, October 1991.

[8] A.K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*. USENIX Association, April 11-12 1999. To appear.

[9] A.K. Ghosh, A. Schwartzbard, and M. Schatz. Using program behavior profiles for intrusion detection. In *Proceedings of the SANS Intrusion Detection Workshop*, February 1999. To appear.

[10] 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.

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

[12] 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.

[13] S. Kumar and E.H. Spafford. A pattern matching model for misuse intrusion detection. The COAST Project, Purdue University, 1996.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] F. Monrose and A. Rubin. Authentication via keystroke dynamics. In *4th ACM Conference on Computer and Communications Security*, April 1997.

[21] 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.

[22] 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.

[23] 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.