Intrusion Detection
Applying Machine Learning to Solaris Audit Data

David Endler
Department of Electrical Engineering and Computer Science
Tulane University
New Orleans, LA 70118
endler@eecs.tulane.edu

Abstract

An Intrusion Detection System (IDS) seeks to identify unauthorized access to computer systems' resources and data. The most common analysis tool that these modern systems apply is the operating system audit trail that provides a fingerprint of system events over time. In this research, the Basic Security Module auditing tool of Sun's Solaris operating environment was used in both an anomoly and misuse detection approach. The anomoly detector consisted of the statistical likelihood analysis of system calls, while the misuse detector was built with a neural network trained on groupings of system calls. This research demonstrates the potential benefits of combining both aspects of detection in future IDS's to decrease false positive and false negative errors.

1 Introduction

Over the past several years, computer attacks and break-ins have become commonplace. Numerous attacks have been successfully launched on government installations, company systems, and personal user accounts resulting in loss of privacy, research, or proprietary information. Almost all existing computer systems safeguard information through access controls such as passwords or file protections. However, from following high-profile stories in the popular press about computer and information theft, it is clear that these mechanisms are insufficient.

Today's Internet community has a significant level of access to and awareness of security holes and system vulnerabilities, propagated through mailing lists, web sites, and newsgroups. Intruders can often easily exploit and gain unauthorized access simply by executing a downloaded C program or UNIX shell script. Other intruders are somewhat more insidious and indirect, hiding their activities and rarely taking part in extended periods of strange or unusual behavior [21]. Simple network probing experiments [4] suggest that there are many more intrusions taking place than are being reported or noticed by administrators on the Internet.

1.1 Intrusion Detection

It is therefore imperative to detect these unseen system attacks in an automated monitoring environment. Intrusion detection quite simply seeks to detect violations in an organization's security policy. Sometimes only through careful analysis of a system's audit data can intrusive activity be detected. *Audit trails*, the chronological record of system activities logged to files, are usually generated in huge amounts each day. For humans to manually comb through these files is unfeasible and a waste of manpower. Thus, it is desirable to have a computerized intrusion detection system to effectively and efficiently

- expose intrusions in real-time to circumvent possible damage or theft resulting from the intruder
- cut down on the tediousness of human analysis

Research and applications of intrusion detection techniques has resulted in its classification into two main categories, *anomaly detection* and *misuse detection.*

Anomaly detection is based on the assumption that misuse or intrusive behavior deviates from normal system use [7]. In general, most anomaly detection systems *learn* a normal system activity profile, and then flag all system events that statistically deviate from this established profile. One of the strengths of anomaly detection is the ability to abstract information about the normal behavior of a system and detect attacks regardless of whether or not the system has seen them before. Most behavior models are built using metrics that are derived from system measures such as CPU usage, memory usage, number and time of logins, network activity, etc. The main weakness of anomaly detection systems is their vulnerability to an intruder who breaches the system during their learning phase. A savvy intruder can gradually train the anomaly detector to interpret intrusive events as normal system behavior.

Misuse detection seeks to discover intrusions by precisely defining them ahead of time and watching for their occurrence [13]. For example, many well known attacks can be discovered by searching for distinguishing patterns or events in the audit trails. The main shortcoming of misuse detection is that future attacks cannot be predicted or detected without hard-coding them into the IDS attack database.

1.2 The Solaris Basic Security Module

The Solaris SHIELD Basic Security Module (BSM) provides security features defined as C2 in the Trusted System Evaluation Criteria (TCSEC) [22]. The audit trails produced by the BSM include detailed information about the specific system events attributable to a user. BSM data has been successfully put to use in past intrusion detection systems [1][12]. BSM recognizes 243 built-in system signals that are individually recorded and timestamped in a main audit logfile. Two more signals were added to this system which were setuid `EXEC_VE` and setuid `EXEC`. Another alteration was that failed signals were given a signal number of 245 added to the original number for a total of 488 possible signals. For example, depending on a user's UNIX shell, typing `ls` may generate some 30 to 100 signals that are recorded and attributed to that user as a result of the command's execution. The Solaris operating system can record different types of signals that fall under the categories of audit options available [22]. All possible system events (signals) for each user were recorded for use in this research.

2 Machine Learning and Intrusion Detection

Intrusion detection systems that are trained on system usage metrics use inductive learning algorithms. To simulate this learning process using a computer model is otherwise known as machine learning. Machine learning can be viewed as the attempt to build computer programs that improve performance of some task through learning and experience.

Our goal of designing machine learning applications with regard to computer security is to reduce the tediousness and time consuming task of human audit analysis.

The most commonly applied theory in many machine learning models is *Pattern Classification*. If we conceive of a pattern as a pair of variables: Pattern = [v, w], then the goal of Pattern Classification is to infer w from v, in other words, giving names w to observations v [19]. These observations v are most commonly represented using vectors of measurements or *features*.

$$v = [v_1 \ v_2 \ ... \ v_n]^T$$

While it would seem beneficial to have as many descriptive features as possible, it has frequently been observed in practice [3] that beyond a certain point, the inclusion of additional features leads to worse, rather than better performance.

2.1 Representing the BSM Signals as Features

Recent work [14] has involved learning the normal UNIX commands of four students over the course of a semester. By converting audit data into overlapping sequences of tokens, the researchers were able to apply a similarity measure to compare with the normal training data. Using this method it was possible to capture the "casual nature" of users' actions and create normal usage profiles for each user.

Another approach to learning from sequence data involves developing feature vectors based on a sequence of events [8]. Such aggregate features that can drastically improve detection include the derived characteristics of the number of arguments to a program, the number of times a user logs in per day, or the type of permissions on files accessed and created by the user. Choosing the best features based on raw audit data is challenging, and involves an expert's knowledge of the domain being studied.

In each of the two methods that were developed in this research, the signals from BSM were extracted and then converted to vector pattern representations. This enabled the learning simulator to process the input, allowing for simple interpretations of the results.

3 Approach and Learning Algorithms

Misuse and anomaly detection each have their own drawbacks. The limitation of the known attacks database in misuse detectors makes it necessary to predict all manners of intrusion before they occur. Anomaly detectors are vulnerable to knowledgeable hackers who tamper with an account during the learning phase of normal behavior patters. This research implements working models of both misuse and anomaly detection, with the eventual goal of combating the drawbacks of each method by combining them into one real-time system. Both models use the same BSM audit data to predict intrusive behavior in the form of UNIX buffer overrun attacks to attain root access. By melding the end results of each detection method with each other, it is possible to improve false negative and positive rates.

3.1 Statistical Anomaly Detection

With a statistical approach to anomaly detection, a system learns the behavior of users, applying the type of metrics previously mentioned. As the system is running, the anomaly detector is constantly measuring the deviance of the present behavior profile from the original. *Only* normal training data is learned by the system, which then attempts to extrapolate anomalous behavior through low probabilistic outputs of the testing data (running phase) outputs. However, false positives and negatives can be generated due to the inadequacy or insensitivity of the statistical measures chosen. Whether enough normal training data was collected is usually a concern as well.

3.2 Neural Networks in Misuse Detection

Neural networks have the ability to learn from an environment by applying an iterative process of adjustments to their internal structure. A neural network is a mathematical mechanism modeled from human brain behavior, able to be applied to a wide range of purposes. A misuse detection approach consists of training the neural network on a sequence of information units (or commands), which are represented on a more abstract level than an audit record [6]. Input to the neural network consists of a group of features that describe one command. This input data can be easily represented using pattern classification techniques, whereby each pattern of features is classified as normal or abnormal system behavior. The neural network uses non-linear regression to abstract information from the abnormal training cases to predict future attacks. For this experiment, the specific type of neural net that was used was a Multi-Layer Perceptron(MLP). The MLP was chosen for its ease of simulation and its past success in pattern classification.

3.3 Simulated UNIX Users

The BSM data used in this experiment originated from a Sparc ELC with 32 megabytes of memory installed with the core Solaris 2.5 operating system. The system was completely unpatched to allow for the exploitation of vulnerabilities of the operating system. Four simulated users were used to generate BSM audit data on this system over the period of about six weeks. Each user over this period had his own profile of system usage which can best be generalized below.

USER1 logs in during regular intervals and uses a wide variety of computing intensive applications. Once this user logs in, she starts an X session, and may launch xterms, `netscape`, `ghostview`, or various other programs. USER1 uses the tcsh shell.

USER2 regularly logs in, checks his email, and telnets to other machines. USER2 uses the csh shell.

USER3 infrequently logs in, and often runs many setuid programs like `ps`, `w` and `passwd`. She also runs many administrative commands such as `df` and `mount`. USER3 uses the sh shell.

USER4 logs in intermittently only to check email, fingers for friends on the network, and sometimes initiates a talk session if a friend is logged in. USER4 uses the tcsh shell.

All of these users' actions were recorded using BSM and later extracted using `praudit` and piped through Perl preprocessing scripts.

4 Anomaly Detection Approach

The underlying idea behind computer security anomaly detection is the ability to extrapolate strange user behavior based just on the normal patterns that the detector learns. *Behavior* is an imprecise term and often difficult to define in terms of a computer user. A method was needed to encode the actions of a user from the BSM audit data format to a mathematical representation that can be statistically analyzed. We wanted to develop a normal profile for each user, and then try to detect anomalous actions (buffer overrun attacks) based on a statistical classifier.

4.1 Feature Extraction of the BSM Logs for the Anomoly Detector

BSM audit data is stored in a binary file which requires processing with the `praudit` program to translate into a readable format. Once the audit file was in a format that could be easily parsed, expert knowledge was used to determine how to extract meaningful features from this format. Drawing from both past sequence data [5] [17] [8] and user profile [17] [14] research, the following preprocessing technique was developed.

Given the original `praudit` translated file of BSM audit data (with *all* users' information), a Perl program separated each of the four users' audit signals into four respective BSM audit files. For each user file, the entire sequence of audit events was converted into a a file of correlated numbers that represent those signal events. For instance, when an `EXEC_VE` signal is received, this is represented as the number 24. If the signal received a failed return value, then 245 is added to the base signal value. Therefore, if a user tries to execute a program, the `EXEC_VE` call would be converted to a 24, or a 269 if permission is denied. Here is an example of how the conversion would look like for the following snippet of audit data for a user called endler. In the original data, each entry represents a token which begins with a `header` and ends with a `return`.

```
header,88,2,AUE_ACCESS,,Thu Apr 02 16:47:36 1998, + 479993000 msec
path,/home/endler/.hushlogin
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,failure: No such file or directory,-1
header,126,2,AUE_OPEN_R,,Thu Apr 02 16:47:36 1998, + 490003000 msec
path,/usr/share/lib/zoneinfo/US/Central
attribute,100644,bin,bin,8388616,267714,0
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,4
header,137,2,AUE_CLOSE,,Thu Apr 02 16:47:36 1998, + 490003000 msec
argument,1,0x4,fd
path,/usr/share/lib/zoneinfo/US/Central
attribute,100644,bin,bin,8388616,267714,0
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,0
header,115,2,AUE_CLOSE,,Thu Apr 02 16:47:36 1998, + 630005000 msec
path,/etc/.name_service_door
attribute,150444,root,root,42467328,2,41680896
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,0
header,106,2,AUE_EXECVE,,Thu Apr 02 16:47:36 1998, + 630005000 msec
path,/usr/bin/tcsh
attribute,100777,endler,user,8388616,120375,0
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,0
header,125,2,AUE_OPEN_R,,Thu Apr 02 16:47:36 1998, + 630005000 msec
path,/devices/pseudo/mm@0:zero
attribute,20666,root,sys,8388616,15061,3407884
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,3
header,110,2,AUE_OPEN_R,,Thu Apr 02 16:47:36 1998, + 650002000 msec
path,/usr/lib/libc.so.1
attribute,100755,bin,bin,8388616,72240,0
subject,endler,endler,user,endler,user,295,294,24 0 h42.s92.tulane.edu
return,success,4
```

This snippet would be converted into entries with timestamps and signal numbers.

```
16:47:36 260
16:47:36 73
16:47:36 113
16:47:36 113
16:47:36 24
16:47:36 73
```

When processing the `EXEC_VE` and `EXEC` system calls, and either invokes a setuid program (which can be detected in the audit data), the derived feature number is converted to 244 and 245 instead of 24 and 8 respectively. For purposes of the buffer overrun exploits that we were searching for, this made the most sense to further describe the `EXEC_VE` call since most of these attacks use setuid root programs. Deciding which aggregate feature(s) to select (e.g., setuid `EXEC_VE`, return code) is entirely dependent on the creators of the detection system, making experimentation sometimes the most practical solution for feature selection.

Once the timestamped signal files have been created for each of the users, the actual training data is produced. The training data is in the form of feature vectors with 488 members (for the 488 possible signals). Here is how the training data feature vectors are created from the first set of files.

The entire sequence of signals is stored in an array fashion. Given the entire sequence of signal numbers for a user, we create a sliding window which will determine how many signals to consider in one pattern. Following is an example of window length five ($w = 5$) incremented by a step of 1 to generate the follow patterns of signals. Using the audit trail snippet we generated above, the sequence 73 113 113 24 73 is converted with this sliding window concept to:

```
260 73 113 113 24
73 113 113 24 73
113 113 24 73 11
113 24 73 11 28
...
```

Then the final training data file is converted to an accumulation of feature vectors with 488 members that count the total number of signals appearing in each sequence. So the first vector pattern of the training data file will look this with the 113th element equal to 2, and the 260th, 73rd, and 24th elements equal to 1.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Thus for each window of w tokens, a feature vector of 488 elements is created. Each line was then prepended by a 0 to classify the pattern as a normal system event since all of our training data is of normal usage.

Dependent on the histogram function (discussed in the next section) being applied to our data, the best value for w must be determined that will optimize our detection of abnormal system events. Three values for w were tested to get a sense for the best length, 6, 10, and 15. For all four users, the lowest sequence length of 6 the was most successful in identifying anomalous behavior. Different data sizes were used for each user's normal profile, which must be taken into consideration before making any generalizations about optimal sequence length. It has been shown that optimal sequence window lengths vary per user and the respective normal profile size being used [14].

The testing data consisted of the users' normal sessions, interspersed with several simulated break-ins to the account. During each break-in, the intruder uploads a buffer-overun exploit binary, and executes it, obtaining root access. The audit data was collected from all of these sessions and converted in the same manner described above for the training file, with a 1 prepended to window vectors that represent the actual attack taking place.

Once the training data was assembled in the desired format, a statistical classifier was applied to the normal patterns. After the classifier had generalized the normal usage for each user, the testing patterns were evaluated and an output value was given to each, representing the likelihood of normal pattern usage. We then marked areas as anomalous by looking at output likelihood values below a certain threshold value.

4.2 The Histogram Classifier

There are many methods to handle pattern classification using statistical analysis. For the purposes of this research, the histogram classifier [3] was used to estimate the likelihood of normal and abnormal classes by creating a set of histograms for each input feature. The histogram classifier belongs to a larger group called likelihood classifiers. Using the data collected above as the training set, all of the patterns were classified as normal. The test data consisted of the same format as the training set, except actual UNIX buffer overrun exploits were executed by simulated intrusions into the accounts. The test data included both normal and abnormal classified examples, signified respectively by a 0 or 1 as the first number in the pattern.

Maximum likelihood classifiers estimate the scaled probability density function, or *likelihood*, for each class, $p(X|A)P(A)$ where A represents a class label, X is the input feature vector for a pattern, $p(X|A)$ is the likelihood of the input data for class A, and $P(A)$ is the prior probability for class A [18]. Using the histogram classifier, the range of possible output values was divided into *bins*. The likelihood value assigned to each bin is proportional to how many training patterns occur in that bin's region. During the testing phase of our system, the likelihood values for each pattern are determined with $p(v|A)$. It is important to have a large representative set of normal data since low likelihood values will occur only when the user exhibits behavior outside of his profile. So by training on normal profile patterns, we can detect abnormal behavior by looking at very low likelihood values for our testing patterns.

The problem with the histogram classifier, and with pattern classification in general, is the curse of dimensionality. The difficulty with using the histogram function is that the number of histogram bins grows exponentially with the dimension N of the measurement space [19]. Thus, the more features we have, the more impractical it becomes to apply the histogram classifier. The solution was to reduce the number of features used in each pattern vector. The initial attempts at feature reduction were attempted using the LNKnet simulator's built-in algorithms. More successful results were achieved using human handpicked features which were selected using experience with the buffer overrun attack domain.

4.3 LNKnet Simulator

The LNKnet software package was developed to simplify the application of the most important statistical, neural network, and machine learning pattern classifiers [18]. For the purposes of this research, all training, testing, and feature selection/reduction was performed using this tool. The LNKnet software has a graphical user interface which allows the user to apply the statistical histogram approach mentioned earlier. The software is also capable of UNIX shell scripting which executes the same algorithms in a batch mode. The LNKnet package was compiled and used on a Sun Sparc Ultra Server 3000, single CPU, with 256 Megabytes of RAM, running Solaris 2.5.1.

Once the Perl scripts generated the normal classified training file mentioned above, LNKnet was then used to build a histogram model which was representative of a user's normal behavior. The testing data was then evaluated using this learned classifier model and the outputs for each pattern were then extracted from the log file. Scatter plots were then graphed with respect to time, and anomalous behavior patterns could then be inferred from a very low output value for the normal classifier.

All input features in the testing and training data were normalized to maintain stable and low output ranges for analysis.

5 Misuse Detection Approach

The underlying idea behind the misuse detection approach is to train our detector on both normal *and* abnormal (intrusive) patterns. Instead of learning the behavior of users over time, this method learns an entire event and creates a feature vector from this. For example, instead of the sliding window used in the anomaly detection approach, a sequence of signals was grouped into an event. In describing how this method works, the *event* is used to mean the collection of signals associated with a particular command or action. For instance, if a user types `ls` and generates 45 associated signals in the audit data, all 45 signals are understood to be included in that event.

5.1 Feature Extraction of the BSM Logs for the Misuse Detector

Using the collected signals of both user and system level events from the same audit source as the training data, feature vectors were generated as before using all 488 measures. A separate file contained the time stamps for each pattern so that we can correlate the data for graphing purposes. An event is extracted from the BSM audit logs by combining all signals within several microseconds that have the same audit session id. Each event pattern is then classified as normal or abnormal system behavior with a 0 or 1 respectively prepended to the pattern. For instance, this is the pattern that records a user typing `ls`.

```
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 4 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

The prepended 0 signifies a normal system event. The test data was generated in the same format and from the same audit source as the anomaly detection test records. Each pattern's timestamp was stored for both training and testing to allow for future correlation with the anomaly detection results.

5.2 Neural Networks

Neural networks are powerful mathematical structures that lend themselves well to pattern classification problems. One goal of this research was to determine whether it was possible to generalize enough information about malicious activity by extracting BSM features, and learn by example using a neural network, to predict future malicious actions. Using the training data that was collected from the main file of all system-wide events mentioned in the previous section, buffer overrun exploits were recorded and added as patterns to the training data, classified as abnormal. The curse of dimensionality also affected our neural network model, which required either fewer features in our patterns or significantly more samples to be sucessful. Feature reduction was performed, and as with the anomaly detection approach, hand-picking the features based on knowledge of the problem domain provided the most successful results.

5.3 LNKnet Simulator

LNKnet was also used for the Multi-Layer perceptron neural network simulations. The only variables adjusted during each simulation were the number of epochs, the number of hidden-layer nodes, and the learning rate parameter. The types of abnormal training patterns were varied to observe the performance rate of the detector.

6 Experiment Results

The main goal of this research was to illustrate the benefits of combining the results of the two methods of intrusion detection. Another goal was to show that intrusion detection was indeed feasible using the BSM audit data converted into feature vectors using the aforementioned techniques.

The training audit data consisted of 6 weeks worth of the users' normal user audit data. The testing phase of this experiment consisted of auditing the four simulated users for 50 minutes, while they performed various buffer-overrun attacks on the system. The total number of signals collected for the training and testing phases respectively was 22444 and 25457.

6.1 The Anomaly Detector

The procedure behind the anomaly detection phase of this experiment involved

1. Initial training on normal user profiles of different sequence samples using a histogram function to develop a classifier.

2. Analysis of testing samples using the histogram classifier against the respective sequence lengths to produce output.

3. Visual analysis of the output and selection of the proper output threshold to determine anomalies.

4. Selection of the best sequence length *w* for each USER's anomalous detections. Plotting of this sequence length versus time, so that we could correlate the data with our misuse detection method which is sampled over time.

The total amount of audit signals that were collected in the training data was 22444 which was divided up for USER1, USER2, USER3, and USER4 respectively: 6535, 2369, 3311, and 4310. The signals not accounted for belonged to system accounts such as root and lp which performed crontab and administrative commands. The testing data consisted of 25457 audit signals generated over the 50 minute interval. The test audit data was split up into user files again respectively for USER1, USER2, USER3, and USER4 with the following number of signals each: 9844, 4765, 4606, and 5683.

The training and testing files were first parsed with Perl scripts to separate each into four files for the respective users. The next step was to transform the files into feature vectors using another Perl script. The main variable that existed in creating the vector patterns for each file was the sliding window *w*. Each user then had three sets of training and testing files that utilized a window size of 6, 10, and 15 when creating the feature vector patterns.

Once the files were created, LNKnet was loaded and the vector values were normalized for use in the histogram plotting. Initially, all 488 features were used in the training and testing data. The resulting likelihood output values were too large to make any conclusions about the testing data at all. This was to be expected based on the previous discussion on the curse of dimensionality and large feature space. Because all of our input patterns were classified as normal, and the test pattern classifications were unknown, normal feature selection algorithms did not suffice.

It has been generally accepted that in pattern classification tasks, features must be first selected and reduced initially using experience and prior domain knowledge [19]. A set of 13 features was eventually selected by applying expert knowledge of buffer overruns and the audit signals generated by their occurrence. The signal features that were eventually chosen were 17, 24, 28, 40, 73, 77, 81, 113, 142, 178, 245, 318, and 386 from the signal list. Note that the numbers chosen to represent each system call was an arbitrary choice, and based on the order they are listed in the filesystem.

Once the feature size had been reduced from 488 to 13, substantial improvements in the results were then achieved. Using the histogram modeling function in LNKnet, each user's training files were learned and then applied to the respective data patterns. The output from each of these experiments consisted of two columns of histogram likelihood values, one for each of the classifiers used, *normal* and *abnormal.* The outputs for the abnormal classifier were all equal to 0 since there were no abnormal pattern examples in the training data as per the concept behind anomaly detection. The outputs for the normal classifier were extracted and examined from the log files with more Perl scripting.

To detect anomalous patterns from the histogram simulation runs, a value was selected for each output to represent the anomaly threshold. This threshold value was visually selected based on the graph of the scatter plots. Any outputs less than this value represented anomalous activity according to our model. For example, Figure 1 illustrates the threshold value of $6 * 10^-28$, in which the buffer overrun instances (shown by the arrows) were isolated by the histogram outputs below this value. The detector was quite successful in this instance, having only two false positives near pattern 4100.

For all four users, the signal sequence window length of 6 seemed to be the most successful in terms of the best false negative rate. Increasing the sequence window length for each user did not improve the anomalous discovery rate, although in the case of USER2, USER3, and USER4 the false positive rate was reduced. For the remainder of the experiments, the sequence window samples of length 6 were used since the highest percentage of anomalies was detected using this data.

Figures 1-4 show the selected anomaly outputs graphed over seconds, allowing us to overlap the results with the misuse detector later. The entire 50 minute testing period can be viewed on these figures, which consolidates the output values and enables our future correlation with the misuse detector. The different types of buffer overrun attacks executed by each user for the testing phase included `passwd`, `rlogin`, `eject`, and `ping` for USER1; `fdformat`, `ping`, and `rlogin` for USER2; `ping`, `eject`, and `rlogin` for USER3; `fdformat` for USER4. The arrows mark the exact patterns at which the exploits occurred so we easily see how many false positives and negatives exist.

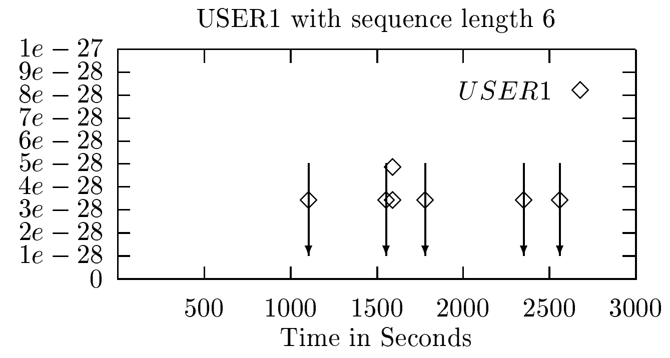

Figure 1: This is the Histogram plot for **USER1's** test data results with sequence size w equal to 6. The arrows represent when actual buffer overrun attacks were launched. The diamonds represent anomalous activity identified by the detector.

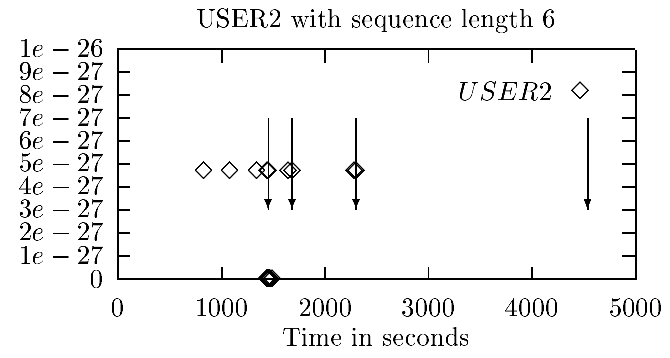

Figure 2: This is the Histogram plot for **USER2's** test data results with sequence size w equal to 6.

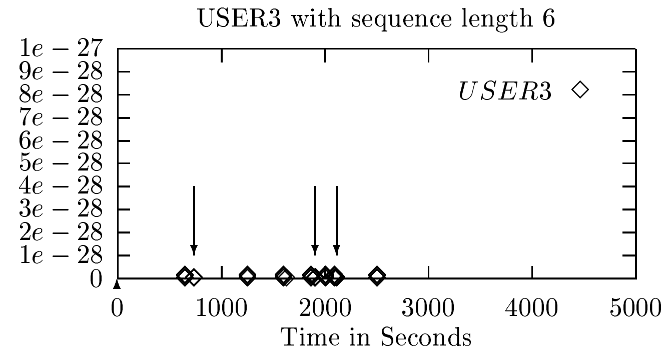

Figure 3: This is the Histogram plot for **USER3's** test data results with sequence size w equal to 6.

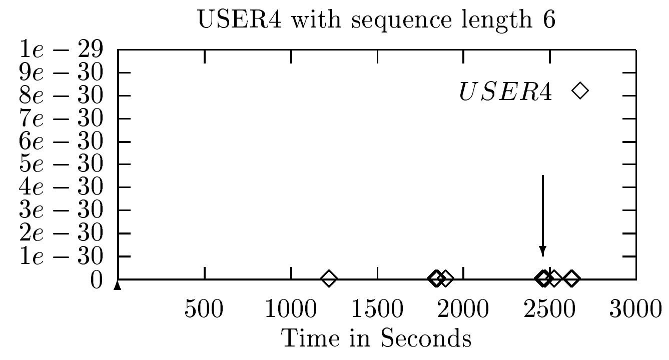

Figure 4: This is the Histogram plot for **USER4's** test data results with sequence size w equal to 6.

6.2 The Misuse Detector

For misuse detection, signals were consolidated into events so that each pattern represented a command and all the accompanying signals it generated. There were a total number of 362 and 569 events in the training and testing files respectively. The misuse detection procedure consisted of the following steps:

1. Initial learning on the *entire* normal training set with a Multi-Layer perceptron neural network simulator (LNKnet).

2. In addition to the normal data, buffer overrun attacks were recorded and specific patterns were individually added to the training set to see which other attacks could be extrapolated.

3. The entire testing data consisting of all users actions and system events was tested using each neural network and the results are shown in the extrapolation tables below.

4. Three combinations of two attacks in the training data were then used as the abnormal cases in the training data, creating three neural networks.

5. The entire testing data, consisting of all users actions and system events, was tested using these three neural networks and the results shown in the table below.

6. The best combination of two attacks was used to combine with the output of the anomaly detection methods discussed in the next section.

The central strategy behind misuse detection involves *training by example.* The main training file for the misuse detector initially contained all of the same normal usage data as the anomaly detection model. Pattern examples of buffer overrun attacks were then added to this training data in order to learn and extrapolate future attacks. Of the five buffer overrun attacks that were used in the testing phase, different combinations of attack patterns were added to the training data to see if the neural network could identify the other attacks in the test data.

The five different types of buffer overrun attacks that were executed in the test data include `passwd`, `rlogin`, `eject`, `fdformat`, and `ping`. Many combinations of the five attacks were included in the training set to see how successful the neural network was in generalizing and classifying future attack signatures.

The first approach was to use one buffer overun attack in the training data to see how many other attacks it could recognize. First, all of the attacks in the testing set were given a number to simplify identifying them. The numbers represent the order they are shown in the graphs with the arrows.

1. `ping` buffer overrun taking place at time 740 seconds by USER3.

2. `passwd` buffer overrun taking place at time 1110 seconds by USER1.

3. `fdformat` buffer overrun taking place at time 1460 seconds by USER2.

4. `rlogin` buffer overrun taking place at time 1556 seconds by USER1.

5. `ping` buffer overrun taking place at time 1685 seconds by USER2.

6. `rlogin` buffer overrun taking place at time 1785 seconds by USER1.

7. `eject` buffer overrun taking place at time 1910 seconds by USER3.

8. `rlogin` buffer overrun taking place at time 2119 seconds by USER3.

9. `rlogin` buffer overrun taking place at time 2305 seconds by USER2.

10. `eject` buffer overrun taking place at time 2353 seconds by USER1.

11. `fdformat` buffer overrun taking place at time 2461 seconds by USER4.

12. `ping` buffer overrun taking place at time 2564 seconds by USER1.

The best performing neural network variables were discovered after much testing and changing of parameters. For each of the simulations performed using the mulitlayer perceptron model, a hidden layer of 25 neurons, a learning parameter of .3 and an epoch length, of 2000 iterations were used.

With the following buffer overrun attacks, the Multi-Layer perceptron was able to extrapolate other attacks. The results of adding and trying each buffer overrun attack in the training data are shown in the table. The neural network was obviously able to extrapolate at least the one attack used in the training set for each instance shown below. The list of extrapolated attacks are the patterns that were classified as abnormal:

Attack Signature	Extrapolated Attacks
1	1, 3, 9
2	2, 10, 11, 12
3	1, 3, 9
4	2, 4, 5, 6, 8
5	2, 4, 5, 6, 8
6	4, 5, 6, 8
7	7, 11
8	2, 4, 5, 6, 8
9	1, 3, 9
10	3, 10, 12
11	7, 11
12	2, 10, 12

Using the information from these individual training instances fed into the neural network, groups of two attacks were then used in the training data. The following three sets of two attacks were used to train the neural network into better extraploation of more attacks.

Attack Combination	Extrapolated Attacks	Figure
2 and 7	2, 7, 10, 11, 12	5
1 and 5	1, 3, 4, 5, 7, 8, 9, 11	6
2 and 5	2, 4, 5, 6, 7, 8, 10, 11, 12	7

The combination of 2 and 5 seemed to be the most effective in extrapolating different types of attacks. This combination was used to combine with the anomoly detection results discussed in the next section. Notice that with both of the above neural net misuse detection runs, no false positives occured.

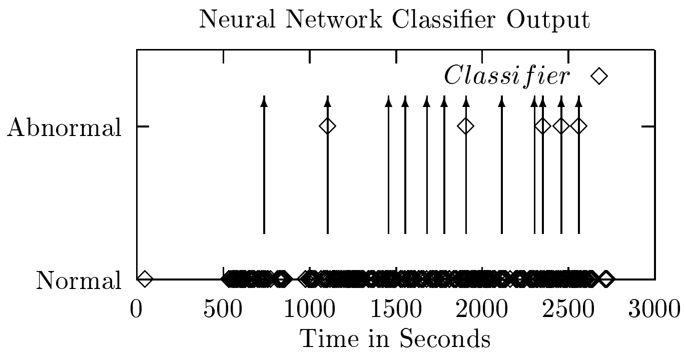

Figure 5: This is the classifier output for the two attacks 2 and 7 used in the training data. The arrows signify when attacks took place, and the diamonds show where the detector flagged an event as abnormal.

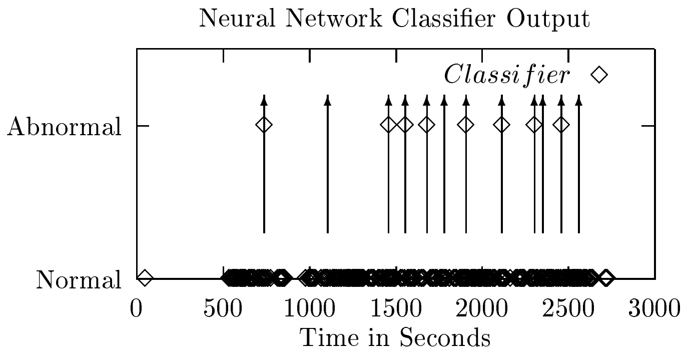

Figure 6: This is the classifier output for the two attacks 1 and 5 used in the training data.

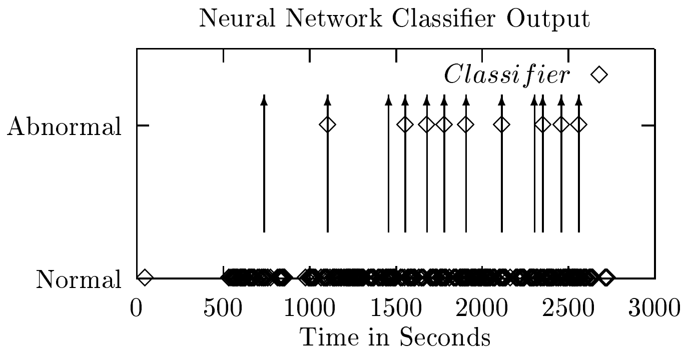

Figure 7: This is the classifier output for the two attacks 2 and 5 used in the training data.

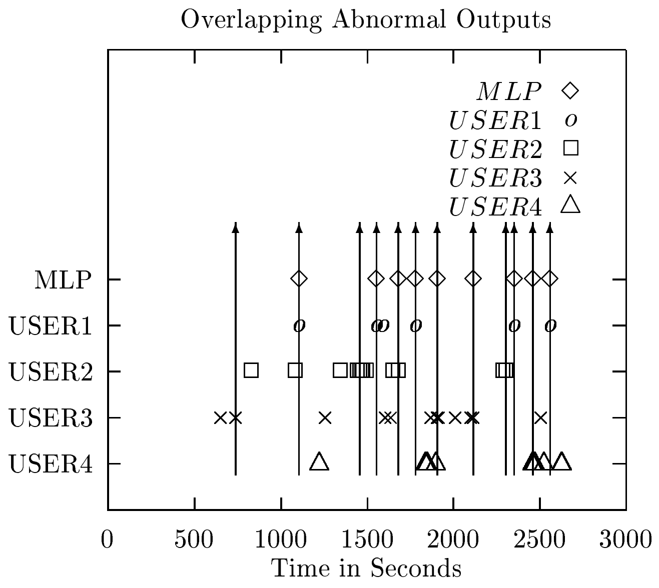

Figure 8: This is the combination of all anomalous output from the four users with the neural network Multi-Layer Perceptron results.

7 Putting it all Together

One of the major goals of this research was to combine both aspects of intrusion detection to improve the overall rate of detection. By combining the results of the anomaly and misuse detection models, it was possible to drastically reduce the false negative errors in detecting buffer overrun attacks. Looking at the output of all users' anomalies and the neural network output in figure 8 shows some overlapping in the outputs of both methods. If we look at just the overlapping features and consider all outputs equally abnormal, then figure 9 offers another perspective. By merging all output points, one can better see areas that require human audit analysis to determine if a real attack has taken place. In the way the abnormal data is presented in figure 9, it is possible to reduce the amount of false positive errors. By only considering regions of 30 or 60 seconds at a time, one such filtering process could consist of eliminating all regions with less than 2 or 3 abnormal outputs thereby cutting down on the false positive rate of detection.

8 Conclusions and Future Work

There were two main goals accomplished in this research. It was successfully shown that Solaris BSM audit data could be applied to both a misuse and

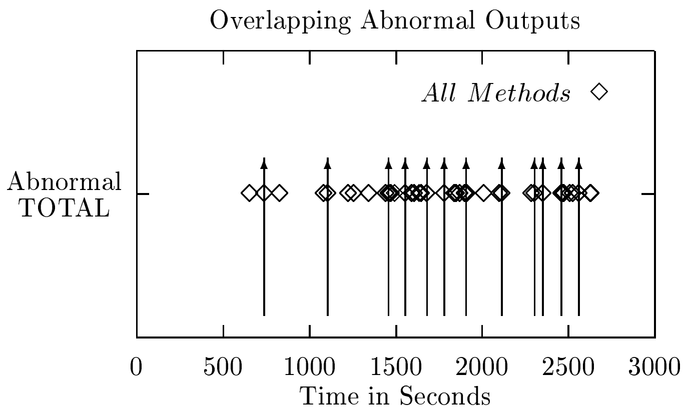

Figure 9: This is the overlapping of data points in figure 8. The graph illustrates an improved detection rate if both anomoly and misuse detection methods are combined.

anomaly detection model. While the NIDES system [1] has demonstrated this theory in practice drawing upon many data sources, this experiment was able to draw upon one source of data and utilize it differently for both types of detection. The two methods by which the BSM data was used to generate feature vectors proved to be fairly successful. The other goal of combining both detection models for more accurate detection results was also successful. By analyzing the final overlapping data points, it is possible to cull the output and consider only the time intervals that contain a certain amount of abnormally classified activity.

These preliminary experiments provide evidence that there is great promise in developing more advanced detection systems through BSM audit analysis and conversion. There is much room for the advancement of many aspects of this research. The signal profile size for each user's training data should be expanded and sampled randomly. The optimal window size w should also be more thoroughly analyzed, since it was shown [14] that optimal sequence lengths are user dependent. The feature selection done on both the anomaly and misuse detection models could be greatly improved from the hand-picked methods applied. More varied and real life user data samples would be desirable, in addition to more types of recorded attacks rather than just buffer overrun exploits.

Now that buffer overrun attacks have been shown to be capable of audit detection, other types of attacks and anomalous behavior should be further classified into the systems. One of the most important improvements that could be made to both models is the development of more detailed features such as arguments to `EXEC_VE` calls, host where the user originated from, and perhaps time between past x commands. Prior Probabilities were left untouched for these experiments, but adjusting the actual probability values of anomalies may have potential benefits.

The neural network aspect of the misuse detection model has vast room for exploration. The technique of clustering should be investigated with respect to the input space. While the Multi-Layer Perceptron is useful in many learning problems, other types of neural networks may be more successful when dealing with this type of pattern prediction and completion. The use of recurrent networks with BSM audit signals is an area that deserves more research, spurred on by experiments showing that predictive pattern matching could be achieved by using a neural network with feedback layers. The application of recurrent networks, as well as other neural networks such as Kohonen, Hoppfield, and RBF, has the potential of providing useful tools for audit sequence analysis.

The next logical step for this research will be to implement this model in real-time using actual user data instead of simulations. Audit reduction should be investigated to make this a performance feasible project. The sliding window of audit signals in the anomaly detector could "slide over" more than 1 signal at a time, reducing the number of patterns to be processed at once. Concept drift must be addressed by constantly retraining both detectors to keep up with the changing uses of a system. Also the neural network audit aspect could be reduced by filtering out audit signals that have little bearing on the detection process, thereby decreasing the preprocessing strain on the system.

9 Acknowledgements

I am very thankful to my advisor Dr. Mark Benard of Tulane University for his constant advice and encouragement. I am also grateful to Dr. Linda Lankewicz of Sewanee University for her useful comments and direction. I would especially like to thank Dr. Richard Lippmann of MIT Lincoln Laboratory for providing the LNKnet simulator software, offering helpful suggestions on how to handle audit data, and for his ongoing support.

References

[1] Debra Anderson, Thane Frivold, and Alfonso Valdes. "Next-generation Intrusion Detection Ex-

pert System (NIDES), A Summary." SRI Computer Science Laboratory, May 1995.

[2] Thomas G. Dietterich and Ryszard S. Michalski. "Learning to predict sequences." 1986. In Michalski, Carbonell, and Mitchell (Eds.). Machine Learning: An artificial intelligence approach. San Mateo, CA: Morgan Kaufman.

[3] Richard Duda and Peter Hart. *Pattern Classification and Scene Analysis.* New York: John Wiley and Sons, 1973.

[4] Dan Farmer. "Shall We Dust Moscow?" *http://www.trouble.org/survey.* December 18th, 1996.

[5] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. "A sense of self for unix processes." *Proceedings of the 1996 Symposium on Security and Privacy*, pages 120-128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[6] Kevin Fox, Ronda Henning, Jonathan Reed, Richard Simonian. "A Neural Network Approach Towards Intrusion Detection." *Proceedings of the 13th National Computer Security Conference*, pages 125-134, Washington, DC, October 1990.

[7] Jeremy Frank. "Artificial Intelligence and Intrusion Detection: Current and Future Directions." June 9, 1994.

[8] Haym Hirsh and Nathalie Japkowicz. "Bootstrapping Training-Data Representations for Inductive Learning: A Case Study in Molecular Biology." *Proceedings of the Twelfth National Conference on Artificial Intelligence.* (pp. 639-644). Seattle, WA.

[9] Todd Heberlien. "Haystack's Analysis: A brief Description." Internal Document, University of California, Davis 1991.

[10] K. Jackson, D. DuBoid, and C. Stallings. "An Expert System Application for Network Intrusion Detection." 1991

[11] H. Javitz and A. Valdes. "The SRI IDES Statistical Anomaly Detector." In *Proceedings, IEEE Symposium on Research in computer Security and Privacy*, 1991.

[12] Sandeep Kumar. *Classification and Detection of Computer Intrusions.* PhD thesis, Purdue University, August 1995.

[13] Sandeep Kumar and Eugene Spafford. "An Application of Pattern Matching in Intrusion Detection." Technical Report. June 17, 1994.

[14] Terran Lane and Carla Brodley. *An Application of Machine Learning to Anomaly Detection.* Purdue University, West Lafayette, IN, February 1997.

[15] Linda Lankewicz and Mark Benard. "Real Time Anomaly Detection Using a Nonparametric Pattern Recognition Approach." *Proceedings, IEEE Symposium on Research in Computer Security and Privacy*, 1990.

[16] Linda Lankewicz. *A Non-Parametric Pattern Recognition Approach to Anomaly Detection.* PhD. Thesis, Department of Computer Science, Tulane University, 1992.

[17] Wenke Lee and Salvator Stolfo, "Data Mining Approaches for Intrusion Detection." In, *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, January 26-29, 1998

[18] Richard Lippmann and Linda Kukolich. *LNKnet User's Guide.* MIT Lincoln Laboratory. 1993.

[19] Jürgen Schürmann. *Pattern Classification, A Unified View of Statistical and Neural Approaches.* John Wiley and Sons, Inc. New York: 1996.

[20] Michael Sebring, Eric Shellhouse, Mary Hanna, and R. Alan Whitehurst. "Expert Systems in intrusion detection: A case study." *Proceedings of the 11th National Computer Security Conference*, pages 74-81, October 1988

[21] Clifford Stoll. *Stalking the Wily Hacker. Communications of the ACM*, 31(5):484-497, May 1988

[22] Sun Microsystems. *SunShield Basic Security Module Guide.*