01 Jan 2001

Experiments on Adaptive Techniques for Host-Based Intrusion Detection

Timothy Draelos

Michael Collins

David Duggan

Edward Thomas

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/ele_comeng_facwork/1040

Recommended Citation

T. Draelos et al., "Experiments on Adaptive Techniques for Host-Based Intrusion Detection,", pp. 1-66 United States. Department of Energy, Jan 2001.

SANDIA REPORT

SAND2001-3065
Unlimited Release
Printed September 2001

Experiments on Adaptive Techniques for Host-Based Intrusion Detection

Timothy Draelos, Michael Collins, David Duggan, Edward Thomas, and Donald Wunsch

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2001-3065
Unlimited Release
Printed September 2001

Experiments on Adaptive Techniques for Host-Based Intrusion Detection

Timothy Draelos and Michael Collins
Cryptography and Information Systems Surety Department

David Duggan
Networked Systems Survivability and Assurance Department

Edward Thomas
Independent Surveillance Assessment & Statistics Department

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-0785

Donald Wunsch
Department of Electrical & Computer Engineering
University of Missouri/Rolla
131 Emerson Electric Co. Hall
131 Miner Circle
Rolla, MO 65409-0040

ABSTRACT

This research explores four experiments of adaptive host-based intrusion detection (ID) techniques in an attempt to develop systems that can detect novel exploits. The technique considered to have the most potential is adaptive critic designs (ACDs) because of their utilization of reinforcement learning, which allows learning exploits that are difficult to pinpoint in sensor data. Preliminary results of ID using an ACD, an Elman recurrent neural network, and a statistical anomaly detection technique demonstrate an ability to learn to distinguish between clean and exploit data. We used the Solaris Basic Security Module (BSM) as a data source and performed considerable preprocessing on the raw data. A detection approach called *generalized signature-based ID* is recommended as a middle ground between signature-based ID, which has an inability to detect novel exploits, and anomaly detection, which detects too many events including events that are not exploits. The primary results of the ID experiments demonstrate the use of custom data for generalized signature-based intrusion detection and the ability of neural network-based systems to learn in this application environment.

CONTENTS

FIGURES

TABLES

NOMENCLATURE

ACD	adaptive critic design
BSM	Basic Security Module
CMAC	cerebellar model articulation controller
DHP	dual heuristic programming
DOS	denial of service
EWMA	exponentially weighted moving average
HDP	heuristic dynamic programming
ID	intrusion detection
IDS	intrusion detection system
IP	Internet protocol
MLP	multilayer perceptron
PCA	principal components analysis
R&D	research and development
STAT	state transition analysis technique
uid	user identification

ACSAC	Annual Computer Security Applications Conference
DARPA	Defense Advanced Research Projects Agency
DISCEX	DARPA Information Survivability Conference and Exposition
MIT	Massachusetts Institute of Technology
NISSC	National Information Systems Security Conference
Sandia	Sandia National Laboratories/New Mexico
UMR	University of Missouri/Rolla

MATLAB™ A high performance language for technical computing, MATLAB is developed and marketed by The MathWorks, Inc.

NT™ NT is a trademark of Microsoft Corporation for its Windows microcomputer operating system.

Solaris™ Solaris is a trademark of Sun Microsystems for its microcomputer operating system.

SPARC™ SPARC is a trademark of Sparc International for a computer architecture developed by Sun Microsystems.

UNIX™ UNIX is a trademark of AT&T Bell Laboratories for its microcomputer operating system.

SNORT SNORT shareware is a small, highly configurable, portable network-based IDS.

RealSecure™ RealSecure is a trademark of Internet Security Systems (ISS) for its intrusion detection system.

NFR™ NFR is a trademark of Network Flight Recorder, Inc. for its intrusion detection system.

Cisco Secure™ Cisco Secure is a trademark of Cisco Systems, Inc., for its intrusion detection system.

eXpert-BSM™ eXpert-BSM is a trademark of SRI International for its intrusion detection system.

Experiments on Adaptive Techniques for Host-Based Intrusion Detection

1. Introduction

The potential of intrusions into computer networks has created a sense of urgency in developing systems capable of detecting attacks before damage results. As computing has become more and more distributed, the number of legitimate network transactions has increased dramatically, setting up a classic information surety dilemma of preventing unauthorized use of network resources while allowing authorized use.

Firewalls are intended to provide a form of electronic protection. However, even with the advent of firewalls, adversaries can attack computer systems by exploiting errors in firewall configuration and ambiguities in security policies, finding ways around firewalls, and attacking network services allowed through the firewall. The failure of firewalls to fully protect computer systems from unauthorized use demands additional defenses in the form of intrusion detection and response.

Current network intrusion detection systems have many shortcomings including

- The inability to analyze large amounts of network traffic
- The propensity to generate huge quantities of false alarms
- The inability to identify new or evolving adversarial behaviors.

Intrusion detection is the process of monitoring computer networks and systems for violations of security policy [**B00**]. The components of an intrusion detection system (IDS) are as follows:

- **Information source**—the data utilized by the IDS
- **Analysis engine**—the process by which the intrusion decision is made
- **Response**—the action taken when an intrusion is detected.

The research project discussed in this report utilized a computer system's audit logs as the information source. We focused on the development of new adaptive analysis engines rather than on responses to intrusions. Therefore, our work falls under the genre of host-based intrusion detection. The data source for this project was limited to audit logs on Sun Microsystems workstations running the Solaris™ Basic Security Module (BSM). Host-based intrusion detection is considered the best way to discover exploits on the contents of computer systems and its applications. This is especially true as network data becomes increasingly protected with encryption.

Intrusions are generally categorized as *misuse* or as *anomalous behavior*. Misuse refers to known unauthorized attacks, while anomalous behavior refers to behaviors or activities other than those normally observed. Misuse detection is found in current intrusion detection systems where patterns of use are scanned for known attacks. These systems are limited to protection against known and identifiable attacks, and are vulnerable to slight variations on known authorized attacks as well as to new attacks. Anomaly detection is a much more difficult problem involving recognition of abnormal behaviors of users or applications, and a judgment about the authorization of the activity. Current anomaly detection systems can suffer a high false positive rate if trained too tightly and a high missed exploit rate if trained too loosely. In general, an optimal threshold exists which minimizes these error rates, but that threshold can be difficult to find and may change over time and with differing traffic patterns.

Multiple approaches exist for the analysis engine, which is designed to make the actual intrusion decision. In this report, we present experiments on three different analysis approaches:

1. Adaptive critic designs (ACDs)
2. An Elman recurrent neural network[1]
3. Statistical anomaly detection.

The original goal was the development of a complete ACD-based intrusion detection system to detect unknown adversarial behaviors. The reason for this focus is that ACDs are capable of two critical components important to the IDS application:

1. **ACDs have the ability to learn in a very murky training environment.**

   One of the difficulties of training an analysis engine to perform intrusion detection is that the exact moment of intrusion is often very unclear. Moreover, how should one treat the events leading up to, but not technically crossing the line of, what is called an intrusion? ACDs utilize a reinforcement-based learning approach, which assesses a penalty or reward to a system based on the job just completed. In other words, it communicates to the system that it should have detected an intrusion somewhere in the previous time period, without specifying exactly where. The intrusion detection problem does not offer the pristine supervised learning environment that one finds in an application like character recognition, where exact outcomes are precisely known in advance and samples abound for each training character.

2. **ACDs have the ability to assign costs in a realistic and complex manner.**

   In general, IDSs apply a binary cost structure of "proper detection is good" and "false alarm is bad." ACDs can allow different quantitative costs to various kinds of detection and different kinds of false alarms. For example, **Table 1** might be used in the training of an ACD-based IDS.

[1] Neural network, inspired by biological neural processing, is the name for a distributed organization of simple processing units that is able to acquire, store, and predict information through a learning process.

Table 1. Possible cost assignments for intrusion detection decisions.

Activity	Cost (Negative = Reward)
No detection	0
False positive	100
Missed detection of low-impact exploit	0
Missed detection of medium-impact exploit	100
Missed detection of high-impact exploit	1,000
Detection of failed low-impact exploit	–10
Detection of failed medium-impact exploit	–100
Detection of failed high-impact exploit	–1,000
Detection of low-impact exploit	–10
Detection of medium-impact exploit	–1,000
Detection of high-impact exploit	–10,000

1.1. Related Work

The work documented in this report presents new approaches to a challenging information surety problem. The goal is to develop a method capable of detecting a wide variety of old and new exploits. Other researchers are exploring ways of developing intrusion detection systems to adapt to new, unseen attacks. Ghosh et al. [**GSS99, GWC98**] have employed neural networks for both anomaly and misuse detection. They used a feed-forward, single hidden layer perceptron neural network. To capture a sense of time in the data, they used a leaky-bucket enhancement to the back-propagation training algorithm. Cannady [**C00**] applied a reinforcement-learning algorithm, a cerebellar model articulation controller (CMAC), to network-based anomaly detection in order to detect denial-of-service (DOS) attacks. Another neural network approach is aimed at learning a legitimate user behavior and detecting anomalies when an intruder poses as legitimate [**RLM98**]. Hoglund et al. [**HHS00**] used self-organizing maps to detect anomalies in user profiling information. Encouraging results on the use of BSM data were reported by Endler [**E98**], who used a neural network as a misuse detector.

Approaches based on state or cost are two methods of intrusion detection that are relevant to the use of adaptive critics. ACDs employ both of those methodologies. A system that models the state of each computer process according to its current privileges and detects unauthorized transitions of privilege is described by Nuansri et al. [**NSD99**]. Another state-based approach [**HCDB99**] allows adaptive security levels based on the state of the system. In yet another approach, the state transition analysis technique (STAT), intrusions are specified as sequences of actions that cause transitions in the security state of a system [**VEK00**]. Taking a state-based approach to intrusion detection in which the number of states is limited makes sense because the dimensionality of the space is reduced to a finite set of states and the approach looks for unauthorized transitions between the states.

Stolfo et al. [**SFLPC00**] describe a cost-based approach to fraud and intrusion detection. This approach recognizes the various costs associated with both detecting and responding to intrusions. These costs are real but are sometimes difficult to quantify. Identified costs include the following:

- **Damage costs** caused by an undetected attack
- **Challenge costs** incurred when responding to a detected attack
- **Operational costs** needed to operate and maintain an IDS.

The ACD approach makes decisions based on both the state of the environment and the cost of each action, and is therefore both state-based and cost-based. Exploring the potential of various cost schemes would be worth an entire research project. The work presented in this report is a preliminary evaluation of the use of ACDs for intrusion detection. From first principles, the potential of ACDs for this security problem is enormous, and the goal is best approached in a series of small steps.

1.2. Organization of Report

Section 2 of this report describes the data source issues relevant to this research, including a description of exploits, the specifics of the audit data used, the need for preprocessing of the data, and the collection of custom training and test data for our analysis engines. **Section 3** presents the standard approaches to intrusion detection, specifically signature-based and anomaly detection, as well as a new approach called *generalized signature-based intrusion detection*. **Section 4** presents the intrusion detection experiments conducted during this project involving IDS control and detection using ACDs, recurrent neural nets, and statistical anomalies. **Section 5** provides concluding remarks.

2. Data Sources

Just as a burglar alarm system uses sensors to acquire the information necessary to trigger an alarm, an IDS must utilize information from one or more sources to arrive at an intrusion decision. In this context, the sensor or data source does not discriminate between normal and abnormal activity but merely collects data. Of course, high-quality data is crucial to automated intrusion detection, as the following poem points out:

A computer, to print out a fact,
Will divide, multiply, and subtract.
But this output can be
No more than debris,
If the input was short of exact.

IDSs are currently hamstrung by available data sources (sensors), which typically include auditing programs and tcpdump for network traffic. Both of these are incomplete in their coverage of potential avenues into a computer, and neither was designed with automated intrusion detection in mind, making them less effective, especially for IDSs that require training. For these reasons, research and development (R&D) of new intrusion detection sensors is advised. The point of view of IDS-directed sensor R&D will impact the completeness and representation of data provided to IDSs, and will advance the underlying science. New sensors as well as improvements in existing sensors will greatly benefit the effectiveness of ID.

The following list includes standard intrusion detection data sources:

- Audit data
  - BSM
- System logs
  - File system statistics
- Application information
  - ftp, telnet, lpr, etc.
- Network packets
  - tcpdump, libpcap
- Other security products
  - Firewall log files
- Out-of-band information
  - Economic and geopolitical indicators, computer-security news bulletins, etc.

To summarize, the difficulties with intrusion detection begin with the actual data. The designer of an IDS must handle a subset of the following difficulties for any current data source:

- Inappropriate data representation for automated intrusion detection
- Temporal nature of data
- Variable-length information
- Massive amounts of data
- Incomplete coverage of the computer system.

Another issue critical to effective ID that is not well understood is exploit characterization, i.e., providing explanations of the nature of exploits and their manifestations. Malicious intrusions involve a series of exploits that are important to detect even if the overall outcome of an attack is unsuccessful. The problem of how best to characterize exploits (e.g., by outcome, manifestation, technique, state, or statistically) is not clear. In addition, the complexity and sheer volume of data is a challenge to any IDS. Visualization and analysis techniques such as cluster/pattern analysis and exploit/attack taxonomies (similar to those applied in the biological sciences) may offer some insight into these problems.

2.1. Exploit Classes

A security-significant event is one that violates a system security policy by attempting to compromise confidentiality or integrity of data within the system, or by reducing the availability of the data or the system itself. These events are called *intrusions* or *attacks*. An attack consists of one or more steps called *exploits*. An exploit could be a buffer overflow that raises a user's privilege level, or a simple database command that erases the entire database after the correct privilege has been attained. Host-based intrusion detection often uses the audit information created and collected by the operating system. Unless these audit data were designed to cover every security-significant event possible, exploits can be designed that can go undetected, resulting in no manifestation in the audit file.

Determination of a taxonomy for exploits is a research topic that has received little attention in the research community. Work done in this area is closely associated with the development of a specific IDS. Each of the several available taxonomies has used a different approach in its definition. The taxonomy developed in the thesis "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems" [**K99**] is an example of an incomplete taxonomy that illustrates the problems with taxonomies created thus far. An individual exploit can be placed within several of the five categories of the author's taxonomy. Each of the taxonomies uses a different characteristic of exploits as its basis, and, therefore, the taxonomies are incompatible.

The following classes of exploits are defined for professional-level adversaries, but may also be operated by script-kiddie-level adversaries:[2]

- DOS
- Buffer overflow
- Direct exploits (i.e., directed at known bugs in applications)
- Placing of trojan horses
- Data exfiltration (i.e., getting data out of a machine).

An overall taxonomy for exploit classification has yet to be developed. The approach for this classification will need to be determined at an initial level under one of the existing organizational strategies, which include:

[2] A script-kiddie-level adversary is a person, normally not someone who is technologically sophisticated, who randomly seeks out a specific weakness over the Internet in order to gain root access to a system without really understanding what it is they are exploiting because the weakness was discovered by someone else. A script kiddie is not looking to target specific information or a specific company, but rather uses knowledge of a vulnerability to scan the Internet for a victim who possesses that vulnerability. (Source: http://webopedia.internet.com.)

- Technical implementation (e.g., buffer overflow)
- Intended outcome (e.g., privilege escalation)
- Detectable manifestation (e.g., file checksum modified).

Each of these strategies has advantages and disadvantages.

Detecting exploit attempts is just as important as detecting successful exploits. For various reasons such as complexity and unintended interactions, exploits that should work for a particular version of software may not work. Detecting these attempts will permit an earlier response than will waiting for a successful exploit to occur. Since a number of exploits usually are available to take advantage of some security vulnerability, any information that can be used to prevent a successful attack will deprive the adversary of some advantage.

2.2. Solaris Basic Security Module

The Solaris™ operating system contains an audit capability called the Basic Security Module (BSM). This facility allows successes and failures for certain actions to be logged for later use. While many actions can be logged, not all the communications paths into the system are covered, leaving the uncovered paths available for use by an adversary.

BSM was developed originally as a response to needs of the government community and, in particular, the U.S. Department of Defense for use in classified processing. Certain audit capabilities were required and BSM fulfilled those requirements. BSM technology was created for use in computer accounting. Later, for security reasons, it was expanded to incorporate more areas. Auditing the access to files and services such as print service was expanded to include access to system calls, privilege escalation, and network access. Both success and failure in the attempted access to these resources also became an audit item.

BSM audit files contain audit records that are comprised of audit tokens each having a different set of token fields. Generally, BSM records are variable in length because each record can have a variable number of tokens (not all tokens appear in each record), and several audit tokens (especially those with textual fields) have variable length.

See **Appendix A** for a more detailed description of the BSM audit log structure. Appendix A also includes some statistical analysis of the contents of BSM files, the frequency of different tokens, and the frequency of patterns of tokens.

2.3. Data Collection for Intrusion Detection System Training and Testing

Proper data collection for training an intrusion detection system poses a considerable challenge. The sheer volume of data generated by most systems, the inherently temporal nature of this data, the great diversity of what constitutes normal operating conditions for networked systems, and the lack of any clearly superior method of representing the data, required us to give considerable thought to the question of just what our training and test sets would look like. Before describing the particular choices we finally made, we review these difficulties.

Over the last three years, a large quantity of data has been gathered at Massachusetts Institute of Technology's (MIT's) Lincoln Laboratory under contract to the Defense Advanced Research Projects Agency (DARPA) for the purpose of testing and comparing IDSs. This is the most well-known and widely used data set for IDS research. The Lincoln archive includes BSM audit files and tcpdump files from a variety of UNIX™ systems, as well as Microsoft Windows NT™ audit files. Audit information was collected from live networks, with certain exploits deliberately introduced into the network. When this data was used for benchmarking various IDSs, some exploits other than those deliberately introduced were detected. It turned out that these detections were not false alarms but rather actual exploits being run against the network from the outside. Of course, without careful analysis, they could easily have been marked as false alarms—a problem inherent in live data used for studies in network intrusion detection. Real networks come under attack quite frequently, but many attacks go unrecognized.

Another difficulty of using live data is the high level of background noise—the large number of daemons and other operating system services which cause numerous kernel events to be happening continually, more or less independently of what the users of the computer system are doing. Of course, an IDS must be robust in the presence of a high noise level, but it may be difficult for a network architecture to learn anything in such a noisy environment. A plausible bootstrapping approach is to do preliminary training on less noisy (and therefore artificial) data, then do further training to teach the network to tolerate more noise. Also note that a persistent problem in neural network research is the lack of any good general method for optimizing the number of hidden layers and hidden nodes; these parameters are most often determined by trial and error. The level of complexity needed to learn in a less noisy environment provides a lower bound on the resources needed to function in a more noisy environment.

Another issue is the difficulty or even the impossibility of pinpointing the time of an attack. This poses a problem for both live and artificial data. In MIT's DARPA data, sessions are labeled as *clean* or *exploit*, but no indication is given of the location of an exploit within a session. This greatly reduces the value of this data for our purposes. In general, a session that contains exploit activity may begin with a perfectly normal series of operations such as logging on as an ordinary user with a valid password. As another example, one machine being pinged by another is neither abnormal nor suspicious; so how many pings should occur before the behavior is identified as a possible port scan?

Given the difficulties with live data in general and MIT's data in particular, we chose to generate our own artificial data. We also chose to use BSM audit files exclusively. The data produced by tcpdump or other similar tools is qualitatively different from BSM data, so that using both would have entailed two largely separate lines of work. This was not a viable option for us. Furthermore, many interesting exploits occur entirely on one machine, making no use of network connections. Tcpdump would not be relevant for detecting such exploits. In contrast, some evidence of network-based attacks should be present in BSM audit trails.

To generate intrusion data, a variety of exploits were compiled and run on a SPARC™ Ultra running Solaris 2.5. Most of the exploits were obtained through SecurityFocus.com.[3] Audit

[3] Source: http://www.securityfocus.com. SecurityFocus.com, Inc., San Mateo, CA.

information was gathered with BSM configured to record all events. Our machines were on an isolated local network. They had no connection to the outside world, and no one using them for any work other than our intrusion detection experiments. This reduces the amount of background noise in the BSM audit data. However, the operating system was still running a variety of nonessential services, so there was a fair amount of other activity going on during the exploits. See **Table 2** for a list of collected data sets of both clean (normal) and exploit activity.

Most of the exploits were local elevation-of-privilege attacks. In such an attack, an ordinary user, after logging in normally, tries to obtain a root shell without knowledge of the root password. This is generally accomplished through exploiting buffer overflows or race conditions. Such attacks are possible because of bugs or careless coding practices in various system libraries and services. We included both successful and unsuccessful attempts at gaining root access; when root access was obtained, or the attempt failed, we terminated the exploit. An actual attacker, after becoming root, would proceed to use this elevated privilege to do something illicit on the system such as deleting or modifying files, installing trojan horses, or reading confidential data. But simply by becoming root without the root password, or by even attempting to do so, the user has already behaved in an abnormal manner, and we can attempt to detect this behavior. The sooner an attack is detected, the better. Since many different attacks make use of fundamentally similar mechanisms, it should be possible to detect attacks in their early stages by learning to recognize these common features.

The data also include some attacks against remote machines; the target for these attacks was another SPARC machine running Solaris 2.5. We ran a port scan, which is a way for an attacker to probe for possible vulnerabilities on a remote machine; and we also ran a DOS attack, in which the attacker causes the remote machine to crash. Audit trails from both the attacker and target were used for training.

Our clean data sets contained a variety of normal activities for a UNIX system, with users running compilers and text editors, modifying files, and using various standard system services. Since false alarms are a major concern, it is necessary for the clean data to contain activities that are superficially similar to activities commonly seen in exploits. Thus, the clean data also included users pinging or connecting to remote machines and sometimes trying unsuccessfully to access files or services for which they did not have permission.

One potential concern with such artificially generated data (and, possibly, with live data as well) is that the network may learn to use simple but irrelevant details to distinguish the clean and exploit cases. The presence of a particular filename or port number, or the use of a particular command-line argument, could be enough to separate the two classes of training data. To address this issue we wrote scripts that would replicate essentially identical behavior (both normal and exploit) while varying these details. Note that preprocessing the data (see **Section 2.4**) may eliminate most of these problems by stripping out filenames or port numbers and leaving only information deemed to be relevant.

We have noted the importance of perturbing the data by including in the training set different versions of the same basic attack and by varying unimportant details like file names and user IDs. This should help generalization and prevent an IDS from learning to distinguish attacks from normal behavior by using such clues. The BSM audit file identified as Variant in **Table 2** contains data collected in an attempt at carry out this plan. Three successful buffer-overflow exploits collected in other data sets (Eject, Fd, and T4) are separated by short periods of normal activity. Nonsense filenames are used for the exploit executables and then, during the periods of normal activity, files that have the same name but don't do anything unusual are executed. Also included are some command-line arguments that are the same in the normal and exploit cases, such as doing something superficially similar to the "eject" exploit. In general, the idea is to generate normal activity that has some superficial similarity to exploit data even though an exploit is not really occurring.

Table 2. Clean and exploit BSM files for training and testing ID systems.

Name	Type	Description
Eject	Buffer overflow	Uses 'eject' utility
Fail.Blat	Buffer overflow	Uses 'passwd'
Fail.Dtprint	Race condition	Uses print utility
Fail.Eject	Buffer overflow	Uses 'eject' utility
Fail.Rpc	Buffer overflow	Uses remote procedure call daemon
Fd	Buffer overflow	Uses floppy disk formatting utility
Rdist	Buffer overflow	Uses remote file distribution utility
Sdtcm	Buffer overflow	Uses desktop calendar manager
Sunkill	Denial-of-service	Open a telnet connection, flood victim with garbage
T4	Buffer overflow	Slight modification of Rdist
ufsrestore	Buffer overflow	Uses file backup utility
Clean	Normal activity	User manipulates files and executes programs
Clean2	Normal activity	Contains activity that is similar to exploit activity: pings and attempts to read files without permission
Clean3	Normal activity	User manipulates files and executes programs
Variant	Buffer overflow Normal activity	Includes Eject, Fd, and T4 exploits, and normal activity similar to the exploits

2.4. Data Representation and Preprocessing

Given the essentially temporal nature of the data, the choice of some type of recurrent network architecture was clear. In recurrent network architecture, there are connections from some internal nodes to the input nodes. Thus, the raw input vector is combined with a memory of previous states, so that the response of the network to an input depends on the context in which this vector is seen. Such a network can learn to respond to temporal patterns in the data. The ability to find temporal patterns is essential for our application, since an exploit necessarily consists of a sequence of interrelated actions; no single BSM event is likely to be suspicious in isolation.

Since BSM data are divided into records, it was natural to feed one input vector into the network for each BSM record. The difficult problem is to determine the contents of that vector. Neural networks require fixed-length input vectors, so variable-length fields such as file names and command-line arguments need to be mapped to fixed-length, reasonably short inputs. Many of the fields in a typical BSM record are not likely to be useful for intrusion detection and can be discarded. Fields with a fixed but large range, such as port numbers or process ids, can be simplified based on prior knowledge. Given a port number, for example, we may only record one bit indicating that it was a reserved port or a user port.

2.4.1. BSM Filtering

Using BSM records as input vectors to a neural network-based IDS causes several difficulties. Among these are the variable length of each record, the redundant information included in each record, and the representation of information in each record. For effective training of a neural network, the feature vector used to represent the state space of the host computer system should be as small and informative as possible. In other words, dimensions of the vector that do not contribute unique information with respect to exploit detection will only make IDS training more difficult. Therefore, considerable effort has been spent simplifying the input BSM data as much as possible without leaving out too much significant information. The decision about what to retain and what to leave out is a heuristic decision that will be fine-tuned through trial and error. The original free format of each BSM record was converted to a strict, fixed-size format for input to a neural network. The section below describes each BSM token that was included in the feature vectors of Sandia National Laboratories/New Mexico (Sandia) and the University of Missouri/Rolla (UMR), and the translation of the token data.

2.4.1.1 UMR Feature Vector

Dr. Donald Wunsch of UMR, a collaborator in this work, applied his expertise in adaptive critic designs to the problem of intrusion detection. UMR performed BSM filtering using the following rules, and achieved data compression and a fixed-length feature vector:

- Each BSM record begins with the header token. From this token, the 16-bit event type value that defines the system call described by the record is retained. This value is fed as 16 binary inputs, where –1 corresponds to 0, and +1 to 1.

- The subject token contains the identification information. The user identification number (uid) is translated to a binary input as follows: +1 if the user is root, –1 if the user is a system account (e.g., the uid, equals –2) or if the uid is unavailable, and 0 in all other cases. A total of 3 binary inputs are used to feed the audit, effective, and real user identification numbers.

- The machine Internet Protocol (IP) address is translated to one binary input: 0 if it is the local machine, +1 if it is the remote machine address that was used to start the session, or –1 if it is some other remote machine. If the value is more than 65534, the 32-bit port address is entered using 16 binary inputs all set to –1.

- Since the process token has the same format as the subject token, the same rules are used. An additional binary input is used to indicate if the process token is not present, in which case the input is set to –1 and all other inputs for this token are set to 0. This technique is used for all tokens that can be absent from the current record.

- The return token contains an 8-bit error code and 32-bit return value. Eight binary inputs are used to enter the error code, but the return value is entered through one input as follows: –1 if it is a negative number, 0 if it is zero, and +1 if it is positive (i.e., only the sign of return value is entered).

- The arg token contains a 32-bit argument value for the system call, an 8-bit argument number, and a descriptive text string. The value and number are represented as 32 + 8 binary inputs. To enter the text information, the 29 different strings appearing in the data were enumerated using 5 binary inputs. To enter an unknown string, all 5 inputs are set to –1. An additional input is used to indicate the absence of the arg token from the current record. If there are several arg tokens in the record, the whole record is repeated with new values of the arg inputs.

- A similar approach is used to enter the exec_args token. The total count of arguments is entered through one input scaled to [0,1]. If there is no exec_args token in the record, this input is set to zero. Eight binary inputs are used to enter the ordinal number of the current argument. The length of the text argument is scaled to [0,1] and fed in via one input. Instead of entering the text string, we enter its spectrum, 256 values corresponding to the frequency of each character in the string. The frequency of a character is computed as the number of appearances of this character in the string divided by the length of the string.

- The path token is fed in using the same spectral decomposition. It yields another 256 inputs for the spectrum, one input for the length, plus another input indicating the presence of this token in the record.

- Two fields are taken from the attribute token: the mode, entered through 32 binary inputs; and the user identification number, which is coded the same way as for the subject token. An additional input indicates the presence of the token.

- To input the text token, 4 binary inputs are used to code one of nine enumerated strings. These inputs are set to –1 for an unknown string, and to 0 if the text token is not present.

- The IP address and port number used in in_addr, iport, and socket_inet tokens are coded as for the subject token. Together with the presence indicators, it yields 2 inputs for the in_addr token, 17 inputs for the iport token, and 18 inputs for the socket_inet token.

Thus, the total number of inputs is 711.

2.4.1.2 Sandia Feature Vector

Experiments conducted at Sandia using neural networks utilized a feature vector using a few different filtering rules than the UMR feature vector (see **Section 2.4.1.1**), as indicated below.

Sandia feature vector rules differing from the UMR feature vector:

- The header token contains the time and date of the event recorded by the BSM record in an 8-byte field. Three of the eight bytes are included in the feature vector. These bytes represent a time span of about 5 minutes.
- The text strings present in the arg, exec_arg, path, and text tokens are represented using a simple hash algorithm with a 1-byte output. For the arg and text token, a single byte is used to represent the descriptive text string. For the exec_args token, which can include many arguments in a single token, the first argument, last argument, and those arguments between the first and last are each represented with a separate byte from the hash algorithm. The path string in the path token is separated into a path minus the filename and filename itself. Each of these strings is hashed to produce two separate bytes.
- The session id is included in the feature vector as a 16-bit unsigned integer.
- Numeric fields in various tokens are presented to the neural network as a single input rather than as multiple binary inputs.

The Sandia feature vector representing 35 separate dimensions or inputs is described below as a C structure.

```
struct feature_vector {
   u_short event_id;                /* header Token */
   u_char event_id_mod;
   u_char timeMSB;
   u_char timeMid;
   u_char timeLSB;
   u_char arg_num;                  /* arg Token */
   u_long arg_val;
   u_char arg_hash;
   u_long file_mode;                /* attr Token */
   u_char attr_uid;
   u_char attr_gid;
   u_char arg_count;                /* exec_args Token */
   u_char arg_first;
   u_char arg_mid;
   u_char arg_last;
   u_char ip_addr;                  /* in_addr Token */
   u_char in_exist;
   u_short port;                    /* iport Token */
   u_char port_exist;
   u_char path_len;                 /* path Token */
   u_char path_hash;
   u_char file_hash;
   u_char proc_uid;                 /* process Token */
   u_char proc_gid;
   u_char proc_ruid;
   u_char proc_exist;
   u_char ret_error;                /* return Token */
   u_char ret_val;
   u_char sock_type;                /* socket Token */
   u_short sock_port;
   u_char subj_uid;                 /* subject Token */
   u_char subj_gid;
   u_char subj_ruid;
   u_short sess_id;
   u_char text_hash;                /* text Token */
};
```

2.4.2 Principal Components Analysis

Principal components analysis (PCA) is a statistical preprocessing method frequently used for neural networks. The PCA algorithm finds an optimal linear transformation of data into a lower-dimensional space that preserves most of the information present in its original form. When the input data contain a great deal of redundancy or a great deal of linear correlation among the coordinates, this enables us to train the network on smaller input vectors. It also means that the network does not need to learn about such redundancy or correlation. Using data from all the files listed in **Table 2**, PCA transformed our 35-dimensional feature vector to 17 orthogonal, uncorrelated dimensions while retaining 99 percent of the variance of the original data. (When running a trained network on new data, we use the same transformation matrix rather than running PCA again. The transformation matrix is supposed to capture statistical properties of the input, so it should be computed using as large a set of data as possible; this need not be the same data used for training.)

3. Intrusion Detection Approaches

Current intrusion detection schemes tend to fall into two different categories: signature-based detection and anomaly detection. Signature-based ID is sometimes called *misuse detection* in the literature. It refers to approaches that match computer activity to stored signatures of known exploits or attacks. Anomaly detection is self-defining in that these systems are designed to detect anything that deviates from normal activity. Somewhere in between these two approaches is a technique called *generalized signature-based detection*. This method assumes the existence of identifiable classes of exploits. These classes may have representative known signatures that can be collected and perturbed to define the boundary of each class.

Figure 1 depicts the three intrusion detection approaches along a sliding scale of information that must be learned (the red or darkened areas).[4] In other words, signature-based detection requires learning the least amount of area from the feature space, whereas anomaly detection

[4] The areas referred to in the text as red will appear dark gray in grayscale representations of the figure, and the areas referred to as blue will appear light gray.

must learn a significant portion of the feature space that corresponds to normal activity, and generalized signature-based detection must learn regions of feature space that correspond to exploit classes. The differences of each of the detection approaches are illustrated in **Figures 1** and **2**.

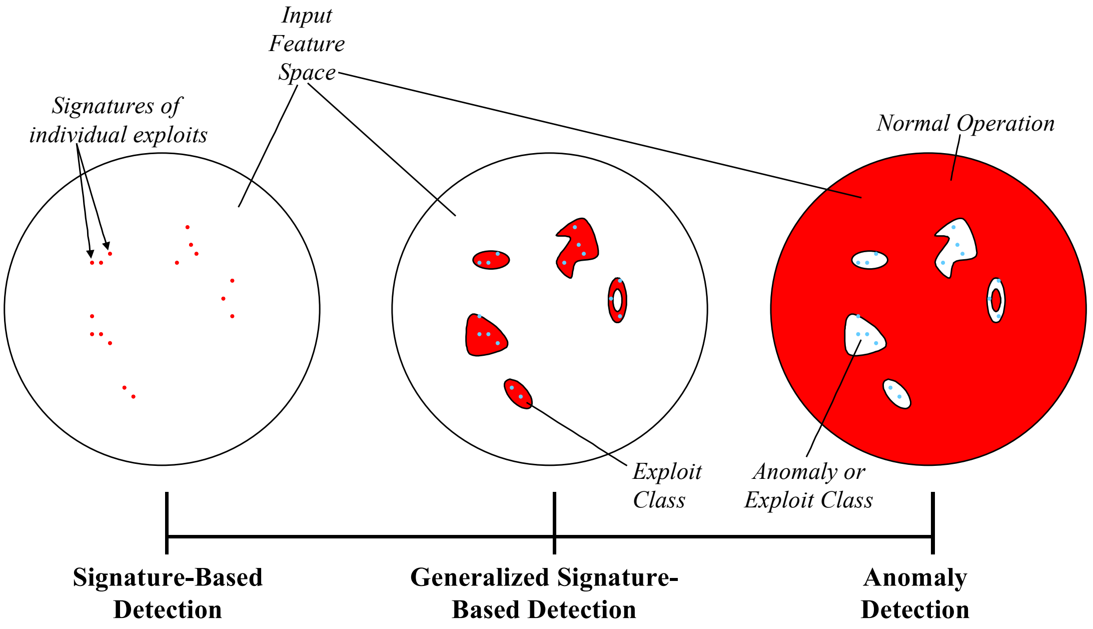

Input Feature Space
Signatures of individual exploits
Normal Operation
Exploit Class
Anomaly or Exploit Class
Signature-Based Detection
Generalized Signature-Based Detection
Anomaly Detection

Figure 1. Theoretical comparison of intrusion detection approaches. The IDS must learn or be programmed to recognize the areas represented in red.

3.1. Signature-Based Intrusion Detection

The most popular commercial IDSs, such as SNORT, RealSecure™, NFR™, and Cisco Secure™, all perform their function through signature-based analysis. In a signature-based system, some known (and supposedly unchangeable characteristic) of the exploit is compared against the data stream, which is usually a sequence of bytes that is important to the functioning of the exploit. When this sequence is found, an alarm is raised and the programmed response is applied.

If the signature is not chosen properly, the system will miss the exploit if it is changed even slightly from the original. This is what makes signature-based systems unable to adapt to slight variations of an exploit. If the signature is not specific enough, it will trigger on many naturally occurring instances of the signature that aren't really an exploit, thus causing a false positive.

The positive side of signature-based systems is the speed. All they are doing is a compare operation, and that can be optimized to work at near-line speed. As other features such as stream reunification are added, these speeds will adversely affect the effectiveness of the system but can be diminished.

A signature-based detection approach has the advantage of precisely codifying specific exploits that are known to exist. Although a signature-based detection approach does not allow detection of novel exploits even when they are similar to known exploits, it is very effective and may be the most efficient approach for detecting known exploits. For this reason, signature-based detection can achieve low false-positive errors with a high incidence of missed exploits. The left circle in **Figure 1** illustrates the signature-based detection concept. It indicates individual points in feature space that represent exploit signatures. The left circle in **Figure 2** illustrates the problem of missing exploits that are close in proximity to known exploit signatures in feature space.

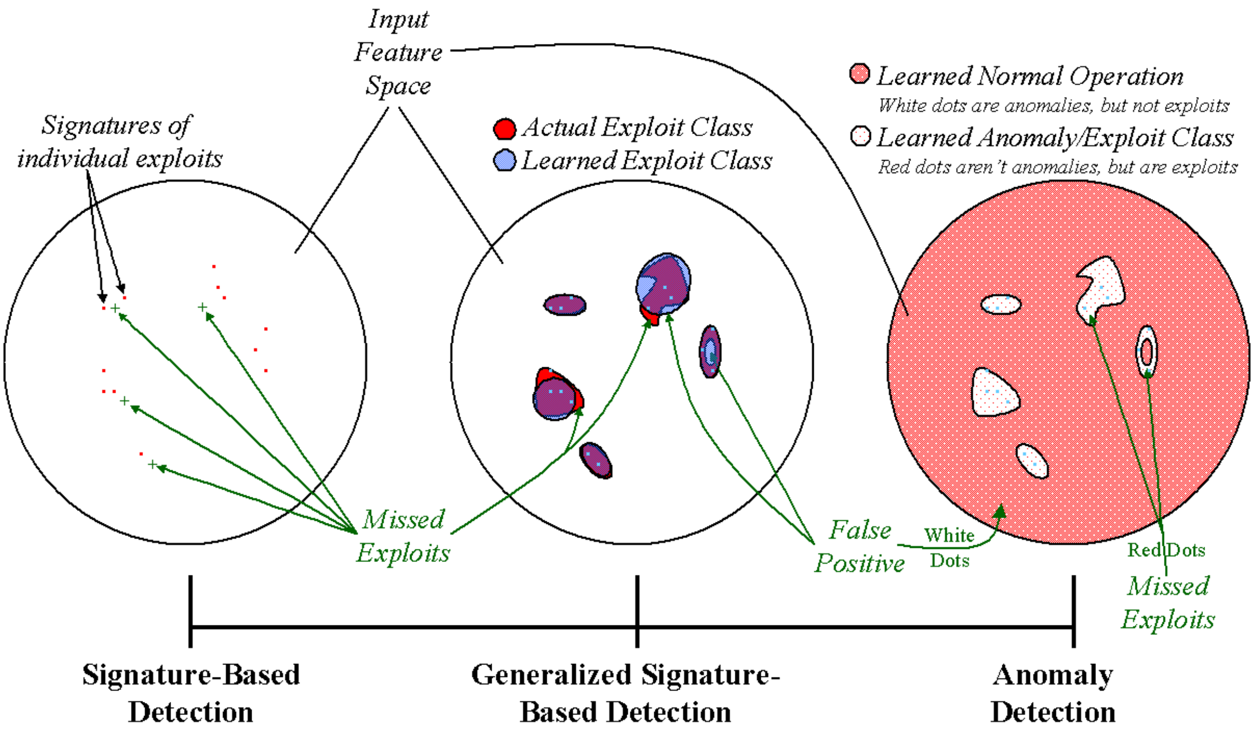


Figure 2. Comparison of different intrusion detection approaches in reality.

3.2. Generalized Signature-Based Intrusion Detection

Generalized signature-based intrusion detection approaches fall between signature-based and anomaly-based detection on the scale of "what an IDS must learn" (see **Figure 1**). The basic idea is to take advantage of the fact that there are well-defined known exploits. With this information, one can theoretically generalize a specific exploit with a representation that will, hopefully, include many exploits that are similar, thereby providing a means to detect new, unseen exploits. A single exploit signature can be generalized by perturbing various dimensions of the exploit's feature vector in sensible ways. Each perturbed exploit will help define a class of similar exploits in feature space. The existence of a class or group of such similar exploits depends on the representation of each. Assuming classes of exploits (i.e., exploits that are similar in some sense) do exist, the job of the generalized signature-based IDS is to determine the borders of these classes or regions in such a way that all exploits can be discriminated from normal data.

One of the problems with this intrusion detection approach is how best to characterize classes of exploits. Do exploits that are similar in end result (e.g., elevation of privilege) or in their technique (e.g., buffer overflow) manifest themselves in close proximity in feature space? Or is each exploit, for all practical purposes, randomly distributed points in feature space? These are difficult questions to answer, but based on the particular feature vector chosen to represent an exploit, one can, at the very least, perturb various elements of the feature vector to acquire additional samples that are close in proximity in feature space. When one defines a region or class of exploits in feature space using this method, the question becomes, "How much of the region represents normal activity?" **Figure 2** illustrates the reality of generalized signature-based detection and the kinds of errors that occur. If exploit classes are defined in a controlled manner with a rich set of examples, generalized signature-based ID is expected to result in fewer missed exploits than basic signature-based IDs, but will potentially result in more false-positive errors.

One reason this approach is attractive is that anomaly detection requires learning an enormous amount of data, and yet much of what is detected may not be an exploit at all. The goal of generalized signature-based detection is to learn to detect exploits rather than anomalies in general.

3.3. Anomaly Detection

Anomaly detection is an approach to detecting intrusions by first learning the characteristics of normal activity. Then, anything that is abnormal is considered an intrusion. The two primary issues in anomaly detection are

1. Learning normal activity
2. Responding to detected anomalies.

The difficulties with learning normal activity on a computer system are the complexities and ever-changing face of computer systems. In addition to frequent hardware and software changes in a computer, users and their habits change over time, as do the kinds of traffic into and out of a computer. Even in the most controlled environments, the space of normal activity is considered to be infinite. This fact is represented in **Figures 1**, **2**, and **3** by the large circle of red that must be learned. The need to indefinitely learn normal activity has been termed *perpetual novelty* [**SF01**]. Nevertheless, techniques for anomaly detection can be designed to be as independent as possible of computer system variability. For example, Ghosh et al. [**GS99**] argue that monitoring process behavior in a computer system can be more easily captured than user behavior, and also is independent of user variability.

Once an anomaly is detected, a decision must be made about the significance and meaning of the anomaly. Since the goal of the system is exploit or intrusion detection, anomaly detection requires additional qualification before raising an alarm. For this reason, anomaly detectors are known to have a high incidence of false positive errors. However, anomaly detectors definitely have the ability to detect new and unseen exploits that would be missed by a signature-based approach.

Figure 1 illustrates the concept of anomaly detection with the right circle, which is filled in red (or dark gray) to indicate the large space of normal activity. Exploits are identified by the white regions, with individual exploits represented as blue (or light gray) dots. The problems with anomaly detection are shown in **Figure 2,** where the red filling is spotted to indicate that normal activity will never fully be learned. Each of these white dots is potentially a falsely identified exploit, or, at least, an anomaly. Missed exploits also are illustrated in **Figure 2**. **Section 4.4** of this report describes an intrusion detection experiment using a statistical anomaly detection scheme.

3.4. Combination of Detection Approaches

As noted in the previous sections on detection approaches, each approach has particular strengths and weaknesses that are summarized in **Table 3**.

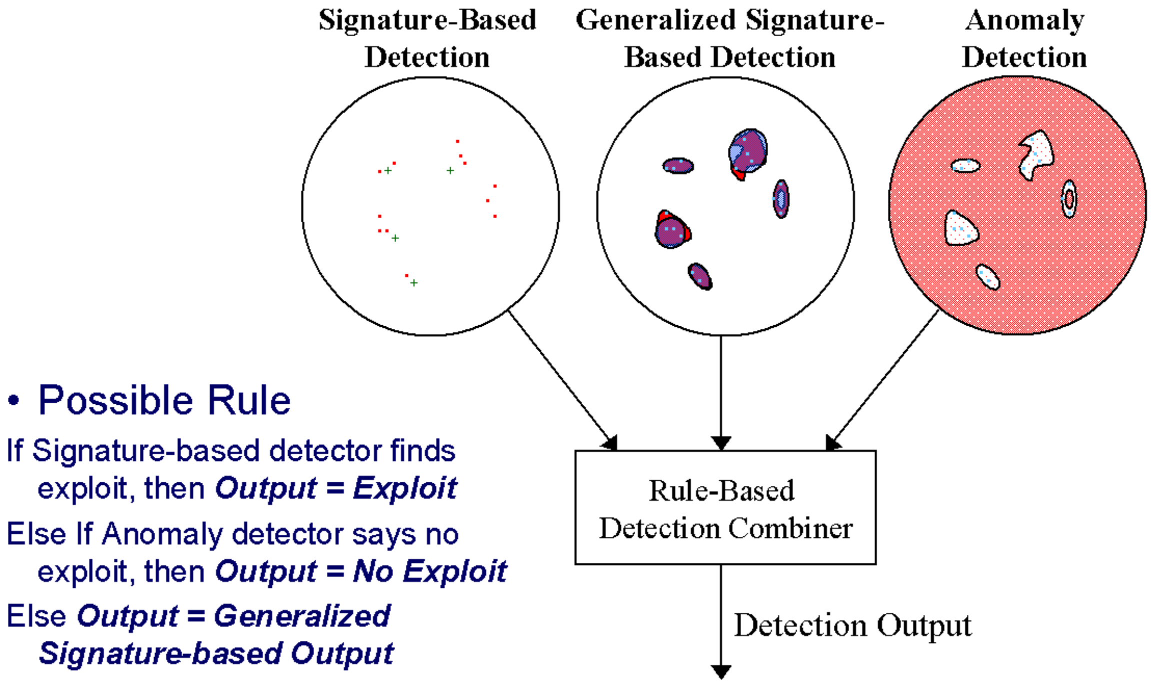
Although not precise and certainly open to debate, **Table 4** provides a judgment as to the expected errors from each of these approaches. This information motivates the consideration of a system that can combine and take advantage of the strengths of each approach. **Figure 3** illustrates the possible architecture and decision rule of such a system.

Table 3. Strengths and weaknesses of intrusion detection approaches.

Detection Approach	Strength	Weakness
Signature-based	Efficient and simple	High maintenance
Generalized signature-based	Learn classes of exploits	Difficult to define classes
Anomaly	Detect novel exploits	Perpetual novelty

Table 4. Expected incidence of false alarms for each intrusion detection approach.

Detection Approach	Incidence of Missed Exploits	False Positive Errors
Signature-based	High	Low
Generalized signature-based	Average	Average
Anomaly	Average	High


Signature-Based Detection
Generalized Signature-Based Detection
Anomaly Detection
Rule-Based Detection Combiner
Detection Output

- Possible Rule

If Signature-based detector finds exploit, then ***Output = Exploit***

Else If Anomaly detector says no exploit, then ***Output = No Exploit***

Else ***Output = Generalized Signature-based Output***

Figure 3. Potential combination of intrusion detection approaches.

4. Intrusion Detection Experiments

Each experiment provides preliminary results of an adaptive intrusion detection technique. By adaptive, we mean that the system can learn and adapt to new situations. In other words, the state of the IDS changes continually in response to changing conditions. To some extent, such change is inherent in the recurrent network architecture described in **Section 2.4**. The network response to the current vector will be different depending upon the recent history of BSM inputs. Furthermore, the capacity for adaptive behavior is inherent in any neural network architecture, since it is possible to continue training (with a small learning rate for stability) while the system is in use.

Ultimately, we want to improve the (cost of detection)/(detection error) ratio. In other words, if the cost of intrusion detection or the detection error is too high, then intrusion detection will be a liability instead of an asset. **Figure 4** provides a pictorial description of this problem. The circle in the upper right of the graph depicts the current state of the art in intrusion detection, while the circle in the lower left represents the intended eventual state of the art.

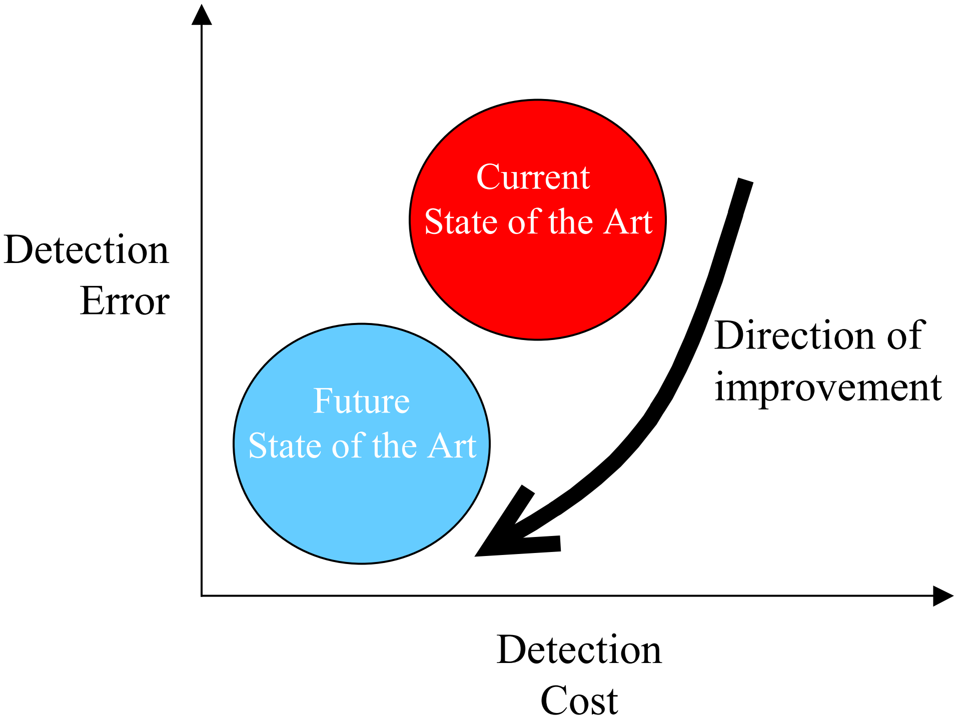


Figure 4. Future direction of intrusion detection systems. The intended trend of future intrusion detection systems to address the two primary intrusion detection issues, cost and errors.

The experiments conducted under this project include research and preliminary results of an IDS controller, two neural network-based IDSs, and an anomaly detection system.

4.1. Intrusion Detection System Control

Adaptive critic designs have had much success in control applications such as aircraft autolanding and chemical process control. For this reason, utilizing an ACD for optimal control of an IDS may provide improved performance. A block diagram for such an IDS controller is shown in **Figure 5**.

The primary difficulty of controlling an IDS is finding controllable parameters. Most IDSs utilize a configuration file that allows the system to be tailored to the customer's computing environment. However, most of the settings, such as a list of authorized users, are not controllable in a quantitative sense. It is fair to ask why the IDS would not already be controlling itself in such a way as to provide effective detection. Here, the object is to utilize a larger or different state space for the controller than for the IDS, thereby allowing adjustment of the IDS to higher or lower sensitivities based on any number of environment variables. The controller's state space might include variables such as time, date, number of users, file system attributes, and other standard ID-oriented information, as well as unusual yet potentially valuable variables such as economic and geopolitical indicators, and computer-security news bulletins. In practice, however, we found that most IDSs are not easily controllable, and constructing the state space for the controller was at least as difficult as constructing the state space to do intrusion detection. For these reasons, IDS control was not deemed to give a high return on investment. Nevertheless, IDS control was considered for neural network-based ID and also for an existing expert system-based IDS called eXpert-BSM™.

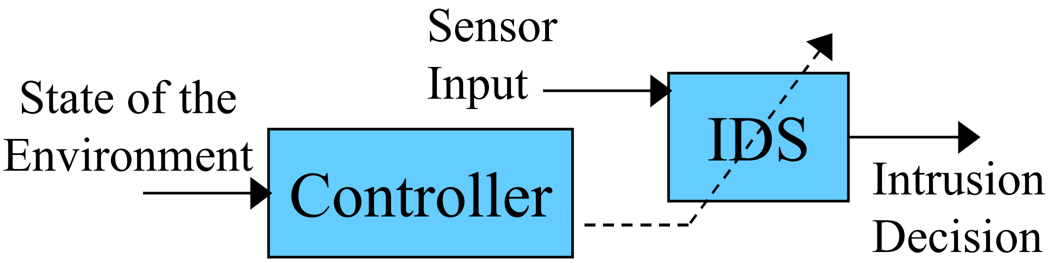


Figure 5. State-based intrusion detection system control.

4.1.1. Neural Network Control

Neural networks have been used for detecting anomalous and misuse intrusions against programs in research that utilized both standard multilayer perceptron (MLP) networks and Elman recurrent networks as experimental IDSs [**GS99, GSS99, GWC98**]. Both of these networks lend themselves to simple control. The parameters and state variables to control in these IDSs include the output threshold values and the leak parameter of the leaky-bucket training enhancement. For example, if the conditions indicating the probability of attack were high, the controller might, based on the state of the environment, lower the output threshold, thereby making the IDS more sensitive to detection of an exploit.

4.1.2. eXpert-BSM Control

Intercepting the BSM audit stream, eXpert-BSM uses an expert system rule base to detect a variety of host-based intrusions. It utilizes a configuration file that allows user control of local environment settings, alert production, an access policy, IP addresses, and even the heuristics used to make the intrusion decision. **Table 5** includes the somewhat self-explanatory numeric configuration parameters deemed controllable. The goal of the controller would be to learn the optimal settings of each parameter for a given state of the environment that achieve the best detection of intrusions with the least number of false alarms.

Table 5. Numeric eXpert-BSM configuration parameters.

Parameter
BSM_MAX_LOGIN_THRESHOLD
BSM_FAILED_LOGIN_WINDOW
BSM_MAX_FTP_BADPASSWORDS
BSM_MAX_NOSPACE_ERRORS
BSM_WRITE_ERR_THRESHOLD_WINDOW
BSM_MAX_CLIENT_PROCS_PER_CYCLE
BSM_EXTERNAL_CONN_THRESHOLD_WINDOW
BSM_MAX_CLIENT_PROCS_PER_CYCLE
BSM_MAX_FAILED_PROCS_THRESHOLD_WINDOW
BSM_MAX_ECHOS_RECEIVED
BSM_ECHO_FLOOD_WINDOW
BSM_NONADMIN_EXPIRE
BSM_FTP_WAREZ_COMPLAINT
BSM_ANON_FILE_EXPIRE

A brief experiment was conducted on eXpert-BSM to evaluate its controllability. The system was tested on DARPA's BSM intrusion detection evaluation data under different configurations. The configuration included manual changes only to quantitative parameters that were considered potentially controllable by an ACD. The results of the system were indeed different, confirming limited controllability of eXpert-BSM. However, the configuration file is not dynamically adjustable, rendering real-time control impractical. In addition, the development of the controller would be at least as much work as the development of an entire IDS based on adaptive critics. Therefore, this effort was halted. Although an ACD was never developed as a controller for eXpert-BSM, the architecture of such a system is shown in **Figure 6**.

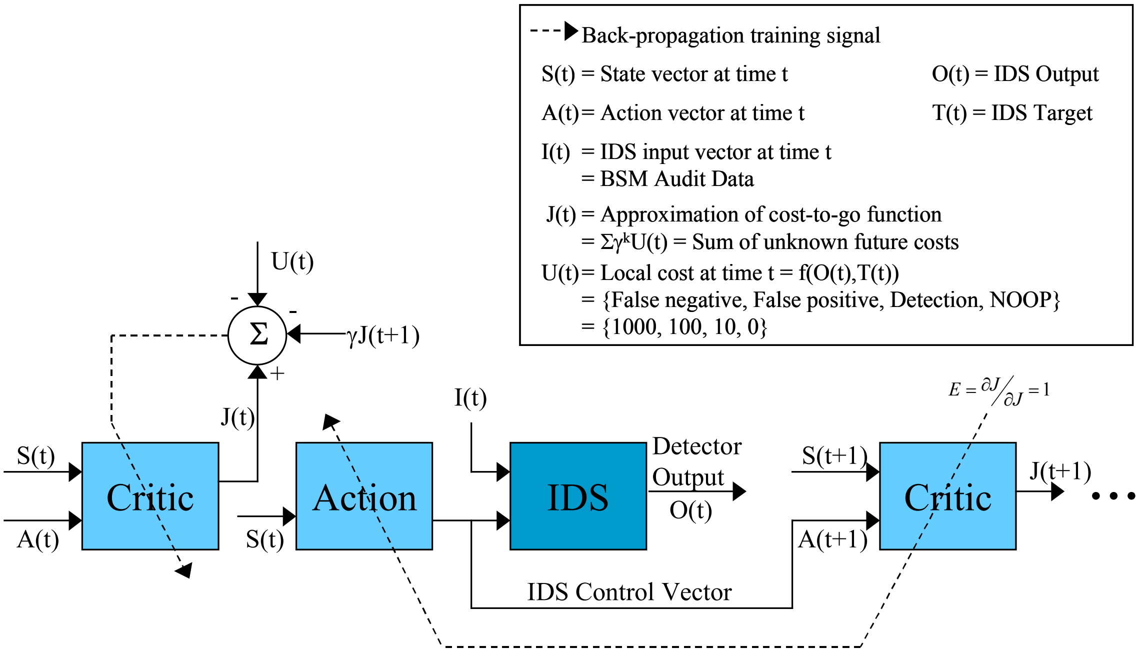

Back-propagation training signal
S(t) = State vector at time t
O(t) = IDS Output
A(t)= Action vector at time t
T(t) = IDS Target
I(t) = IDS input vector at time t
= BSM Audit Data
J(t) = Approximation of cost-to-go function
= Σγ^kU(t) = Sum of unknown future costs
U(t)= Local cost at time t = f(O(t),T(t))
= {False negative, False positive, Detection, NOOP}
= {1000, 100, 10, 0}
U(t)
-
-
Σ
γJ(t+1)
+
J(t)
S(t)
A(t)
Critic
S(t)
Action
I(t)
IDS
Detector
Output
O(t)
IDS Control Vector
E = ∂J/∂J = 1
S(t+1)
A(t+1)
Critic
J(t+1)
• • •

Figure 6. Adaptive critic design for intrusion detection system control.

4.2. Intrusion Detection using an Elman Recurrent Neural Network

The second ID experiment conducted under this project utilized a recurrent neural network known as an Elman network. The goal of this experiment was to establish the efficacy of the Elman network in discriminating between data known to be free of exploits and data known to contain an exploit. The ability of the network to learn a training set indicates, at the very least, that the classes represented in the data (clean and exploit) are separable or distinguishable to a certain extent. The Elman network consists of a multilayer perceptron network with feedback from a hidden-layer output to the hidden-layer input. It is this connection that allows the Elman network to detect time-varying patterns. Two experiments were conducted using an Elman recurrent neural network architecture. The first experiment investigated basic capabilities of this approach and the second experiment pursued a performance-oriented objective.

4.2.1. Experiment 1

When training a neural network, many questions arise. The answers to these questions can impact the performance of the network on the specific task at hand. **Table 6** lists these questions as well as the details of the Elman recurrent neural network Experiment 1.

Table 6. Elman recurrent neural network-related questions, Experiment 1. Important neural network-related questions and the corresponding answers for the Elman recurrent neural network Experiment 1.

Questions	Answers for Elman network Experiment 1
How many hidden layers?	1
How many nodes in each hidden layer?	88
Which training algorithm?	Gradient descent back-propagation with momentum and adaptive learning rate
What activation functions?	Logistic sigmoid
What learning rate?	Adaptive

When one performs supervised training of a neural network, a target or "correct" output value for each input feature vector is used. Clean data were assigned a target value of 0, while exploit data were assigned a value of 1. However, it is unrealistic to expect the neural network to recognize an exploit after just one or a few BSM records. To address this problem, the target network output is ramped up gradually from clean (0) to exploit (1) at the beginning of an exploit.

Figure 7 shows the results of an Experiment 1 network on a fragment of clean data followed by a fragment of exploit data (repeated once), in this case the Rdist exploit (see **Table 2** in **Section 2.3**). Network responses are plotted as red (or dark gray) dots, and target outputs are plotted as a blue (or gray) dashed line. The shape of the target ramp is obvious on the left side of the exploit data sequence. This result clearly shows that there is reason to believe that a clean fragment of data can be distinguished from an exploit fragment. At the very least, this is a validation of a credible feature vector.

Inspection of **Figure 7** indicates that some form of postprocessing might be necessary to arrive at a final decision about whether an intrusion exists in a fragment of data or not. The obvious problem is placing too much value on stray or outlying samples in a clean fragment that might be considered an exploit. **Figure 8** shows an Experiment 1 network output after postprocessing using the following steps:

1. Median filter the output to suppress outliers. Any number of signal processing filters could be utilized here to serve the intended purpose.

2. Threshold the median filtered data, resulting in a binary output.

The results shown in **Figure 8** were produced using a 5-sample median filter and a threshold of 0.5. However, a more complicated threshold scheme could be employed to minimize false positives and missed exploits. The first three exploit and clean fragments were part of the training data, while the final clean and exploit fragments were completely new to the network.

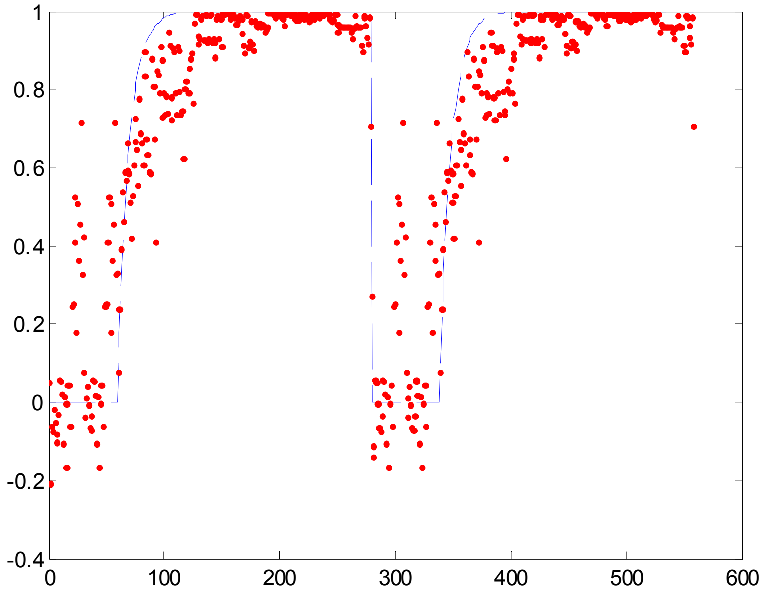


Figure 7. Elman network Experiment 1 after 10,000 training epochs.

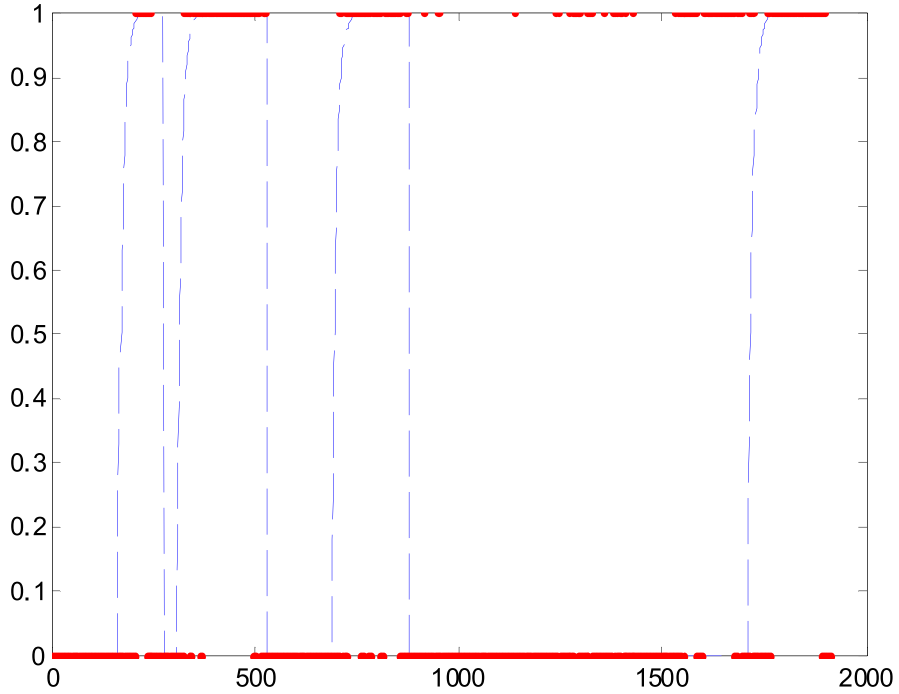


Figure 8. Elman network Experiment 1 after 50,000 training epochs. The data consists of both training data and unseen test data (the final clean and exploit segments.

4.2.2 Experiment 2

While Experiment was performed early on in the research project, Experiment 2 involving an Elman recurrent neural network took advantage of all the experience gained in the project. In addition to the specific tuning of the neural network as described in **Table 7**, principal components analysis (see **Section 2.4.2**) was employed as well.

Table 7. Elman recurrent neural network-related questions, Experiment 2. Important neural network-related questions and the corresponding answers for the Elman recurrent neural network Experiment 2.

Questions	Answers for Elman network Experiment 2
How many hidden layers?	2
How many nodes in each hidden layer?	10 in both hidden layers
Which training algorithm?	Resilient back-propagation
What activation functions?	Tangent sigmoid
What learning rate?	Adaptive

Results of this Elman Experiment 2 are shown in **Figures 9–11**, where the target outputs are shown in a solid line. The ability of the network to discriminate between clean data and data from several different exploits is clear. However, while figures 9 and 10 show results using well controlled, very clean data, Figure 11 uses "messier" data that involved intentionally introducing similar activities as the exploits into the clean data.

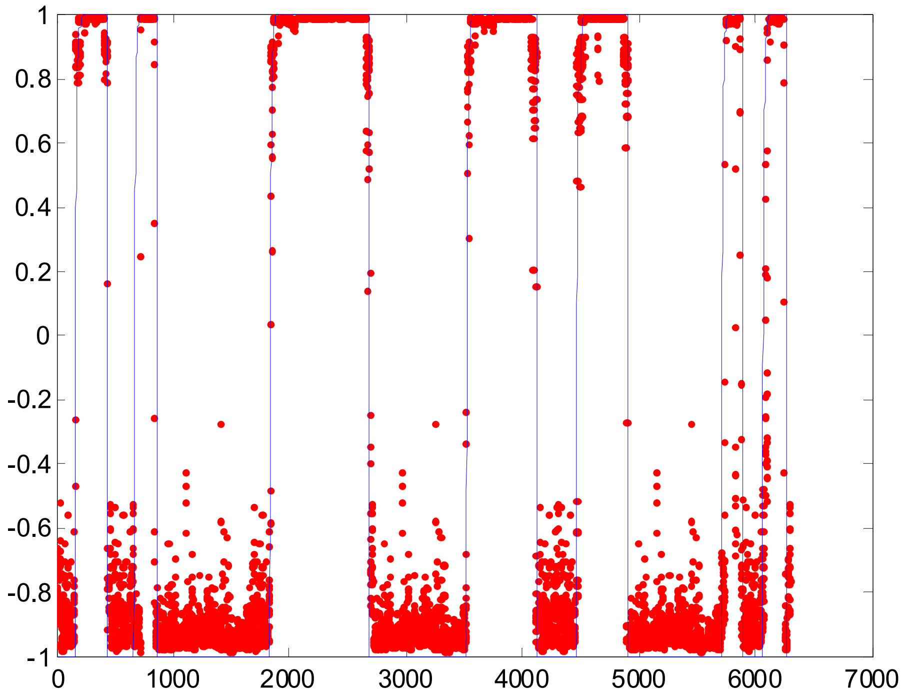


Figure 9. Elman Experiment 2 network training results on multiple exploits. The training data consists of the following exploits in order: Fdformat-T4-Fd-Eject-Ufsrestore-Sdtcm-Rdist.

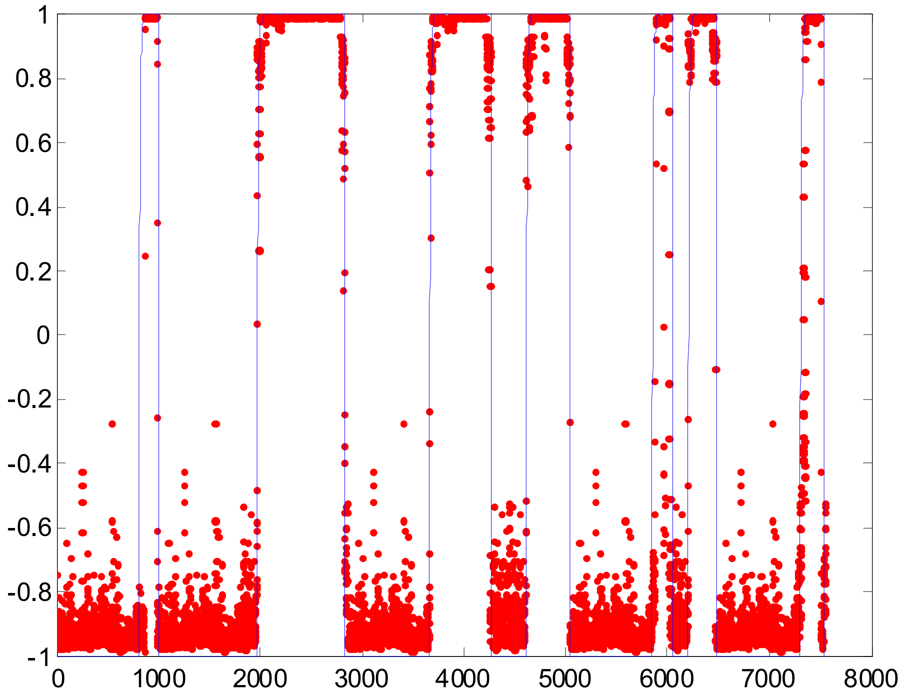


Figure 10. Elman Experiment 2 network test results on multiple exploits. The test data consists of the following exploits in order: T4-Fd-Eject-Ufsrestore-Sdtcm-Fdformat-Rdist.

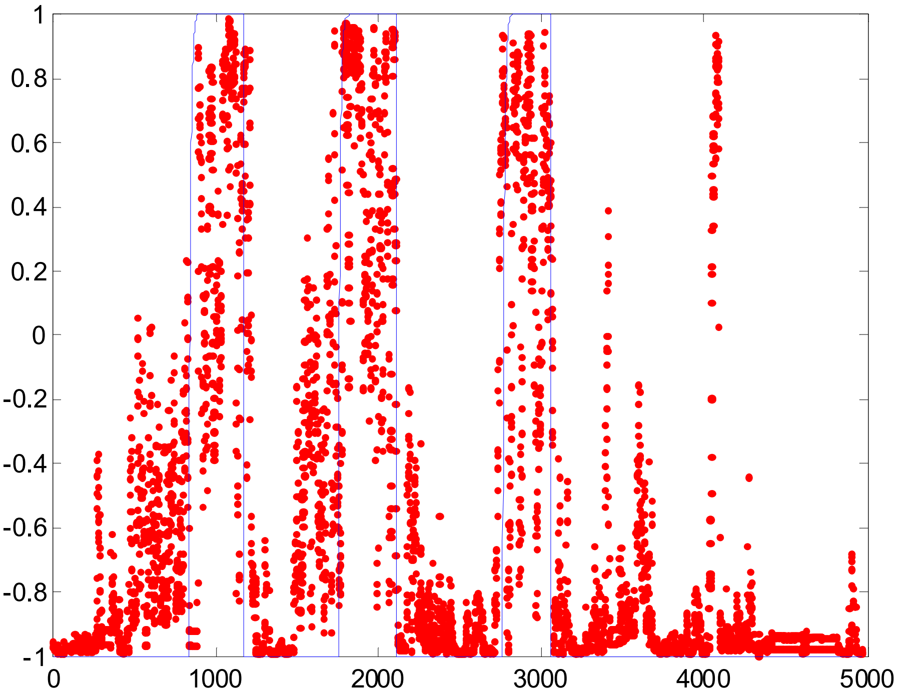


Figure 11. Elman Experiment 2 network training results on Variant data set.

4.3. Intrusion Detection Using Adaptive Critic Designs

Generally, two approaches to the intrusion detection problem can be taken. The first is aimed at learning a legitimate user's behavior, and the second is aimed at detecting anomalies when an intruder poses as a legitimate user. The approach presented in this section is a combination of the two, where the IDS learns both legitimate behavior and anomalous exploits, and is able to distinguish between them.

Thus, we have the following problem: to monitor the input stream and signal when a particular event (intrusion) occurs. We have a set of examples: stream fragments with intrusions, and stream fragments with no intrusions. We also know the following:

- A lot of irrelevant data (data that came from other processes running on the system but not involved in the intrusion) are contained in the fragments
- Absolute identification information may vary and does not signal an intrusion (i.e., an intrusion can be carried out using any or several different user IDs and machine addresses)
- Exploits leading to an intrusion do not necessarily follow the same exact sequence each time.

Therefore, from the examples, our intrusion detection system should find how intrusions differ from normal operations. To achieve this, we employ the reinforcement learning approach. Unlike the supervised learning approach, in which the system is judged at every step, the reinforcement learning approach allows a more natural way of training based on the outcome.

Using reinforcement learning at the end of a training fragment, the system gets a reward if the system has correctly indicated the intrusion at any time during the fragment. The system is penalized at the end of the fragment if the system has missed the intrusion or given a false alarm. As a result, the system has to find out what it did right or wrong, thereby solving a credit assignment problem. In a return, such a delayed credit gives the system the freedom of trial and error exploration. Given a new, unknown fragment, the system makes its best guess based on experience. At the end of the fragment, the system learns from its errors.

Since the system is trained only at the end of a training fragment, it is appropriate to implement lifelong training in which the system keeps learning while it is in field use. This approach allows the system to follow the changing environment and to catch up as new unknown attacks appear.

The important characteristic of an intrusion detection system is the false alarm rate. A false alarm occurs if the system indicates an intrusion when none exists in the data. If a system "cries wolf" too often, most likely that system will be ignored. To minimize this possibility, we introduced a set of immediate penalties for false alarms and missed intrusions, and required the system to minimize a discounted sum of penalties, thus casting our problem as a dynamic programming problem.

4.3.1. Methodology

We use the heuristic dynamic programming (HDP) to approach the problem [**PW97**]. HDP is the most straightforward member of the ACD family. Consider the agent-environment model. The agent senses state information from the environment, responds with an action, and is rewarded or penalized. In our case, the agent is the IDS and the environment is the computer system together with a human system administrator. The environment provides the state information through BSM data. The IDS indicates either an intrusion or no intrusion. When the system administrator investigates the alleged intrusion or discovers a missed one, the administrator provides the IDS with a penalty or reward, summarized in **Table 8**.

Table 8. The reinforcement-learning approach for penalizing or rewarding the IDS.

Intrusion	Alarm	Penalty/Reward*
Yes	Yes	–1
Yes	No	1
No	Yes	1
No	No	0

* A negative number indicates a reward.

The adaptive critic design we use (see **Figure 12**) is comprised of two neural networks: *action* and *critic*. The action network senses the input state information and responds with an action. The critic network gets the state information, analyzes the action taken, and estimates the cumulative cost. The incurred immediate cost is used to correct this estimate during training of the critic. Then, this estimate is minimized via training of the action network.

Consider $U(t)$, the immediate cost incurred at moment t. Then, the estimated cumulative cost $J^*(t)$ is expressed as follows:

$$J^*(t) = \sum_k \gamma^k U(t+k), \qquad \textbf{(Equation 1)}$$

where γ is a discount factor ($0<\gamma<1$). Let $R(t)$ represent the state information at moment t, $A(t)=A(R(t))$ represent the response of the action network, and $J(t)=J(R(t),\mathrm{A}(t))$ represent the output of the critic. The critic is trained, minimizing the following error functional

$$\begin{gathered}\sum_t E^2(t), \\ E(t) = \gamma J(t+1) + U(t) - J(t)\end{gathered} \qquad \textbf{(Equation 2)}$$

i.e., $\gamma J(t+1) + U(t)$ is the target value for $J(t)$. The update rule for the critic weights W_C is given by

$$\Delta W_C = \eta_C (\gamma J(t+1) + U(t) - J(t)) \frac{\partial J(t)}{\partial W_C}, \qquad \textbf{(Equation 3)}$$

where η_C is a small positive learning rate.

Since we seek a decision policy (realized by the action network) that minimizes the cumulative cost, we minimize $J(t)$ using gradients computed by the critic and action networks

$$\Delta W_A = -\eta_A \frac{\partial J(t)}{\partial A(t)} \frac{\partial A(t)}{\partial W_A}, \qquad \textbf{(Equation 4)}$$

where ΔW_A is the weights update for the action network, and η_A is a small positive learning rate.

To address the temporal nature of the state information given by BSM data, we added a recurrent identification network to our design. The purpose of this network is to extract the state information $R(t)$ from the input BSM data $X(t)$ and maintain that information in the recurrent layer. This network is trained together with the critic

$$\Delta W_I = \eta_I (\gamma J(t+1) + U(t) - J(t)) \frac{\partial J(t)}{\partial R(t)} \frac{\partial R(t)}{\partial W_I}, \qquad \textbf{(Equation 5)}$$

where $\partial R(t)/\partial W_I$ is the gradient computed by the identification network with respect to its weights W_I. Derivatives with respect to the recurrent layer weights are calculated to depth 0. In other words, we don't go recursively back in time through delay elements computing these derivatives (i.e., the delay elements of the recurrent layer are considered as additional inputs).

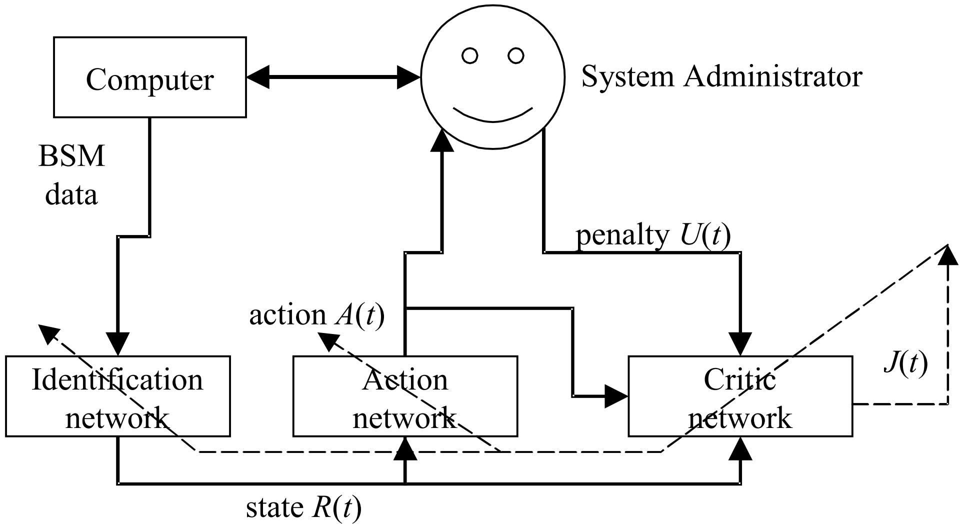


Figure 12. Adaptive critic design for BSM-based intrusion detection.

4.3.2. Initial Results and Discussion

To train the ACD, we chose eight sessions with four different exploits (code-named *format*, *ffb*, *ftp-write*, *eject*, with two different sessions for each type of exploit), and eight clean sessions (i.e., sessions without exploits). The design was further tested using an additional four sessions with exploits and four clean sessions. The design is modeled using the MATLAB™ Neural Network toolbox. The identification network has 711 inputs, 50 neurons in the hidden recurrent layer, and 50 output neurons. The action network has 50 inputs, two hidden layers with 30 and 10 neurons respectively, and one output signaling an intrusion. The critic has 51 inputs and one output, and the same configuration of hidden layers as the action network. The transfer function used in the hidden layers is the hyperbolic tangent function, and the transfer function in the output layers is linear.

The discount factor γ was chosen to be 0.9 and the learning rates were all equal to 0.01. During training, one of 16 training sessions was chosen randomly and fed through the design. This action was repeated 30 times and then the design was tested in eight test sessions.

This experiment was conducted several times. In most cases, the critic degenerated such that it gave a constant output regardless of the input. As a result, the gradients $\partial J/\partial R$, $\partial J/\partial A$ were zero and the action and identification networks did not learn. A few times, when the critic was able to provide nonzero gradients, there was not sufficient time for the action network to achieve reasonable behavior. In some cases, the action network did not report any intrusions; in most cases, it always reported an intrusion.

These results leave plenty of room for future research. An optimal penalty system, values for learning rates, and the discount factor are yet to be determined. The weak performance of HDP critics is well known; therefore, we are considering future application of the more powerful dual heuristic programming (DHP) design.

4.3.3. Later Developments

To enhance the training procedure and take advantage of a variety of training algorithms available in the Neural Network toolbox, we modified the initial ACD. The identification network was incorporated into the action and critic networks as an additional recurrent hidden layer. A model network, which calculates the penalty, was also introduced. To facilitate HDP training in batch mode, a layer was added after the critic network (see **Figure 13**) to calculate the error according to **Equation 2**. Both the action and critic networks now have at least two hidden layers. The first hidden layer gets its input from the input layer and has a recurrent connection to itself. The second hidden layer has input connections from both the input layer and the first recurrent hidden layer. Additional hidden layers can be added if necessary. The model network calculates the penalty value according to **Table 8**. It has two inputs: the output of the action network ("alarm"), and another input that indicates an intrusion during the training. This network is hard-coded and is not changed during training.

The ACD is implemented as one big network object. The original MATLAB Neural Network toolbox was modified to allow "forward look-ups." According to **Equation 2**, a value from the next time step is needed to calculate the error in the current time step. This is possible in batch-training mode when the error is calculated and the network is updated only after the whole training set is presented.

The training of the design is a two-stage process. In the first stage, the critic network is trained to estimate the cost-to-go function *J*. The output of the last layer is used as the training signal while the parameters of the action network are "frozen." In the second stage, the trained critic network is used to train the action network, with the goal of minimizing the estimate of *J*. The weights of the critic network are held frozen and –1 is back-propagated through the critic network to get the training signal for the action network. The training procedure is repeated until the weights of the action network stop changing.

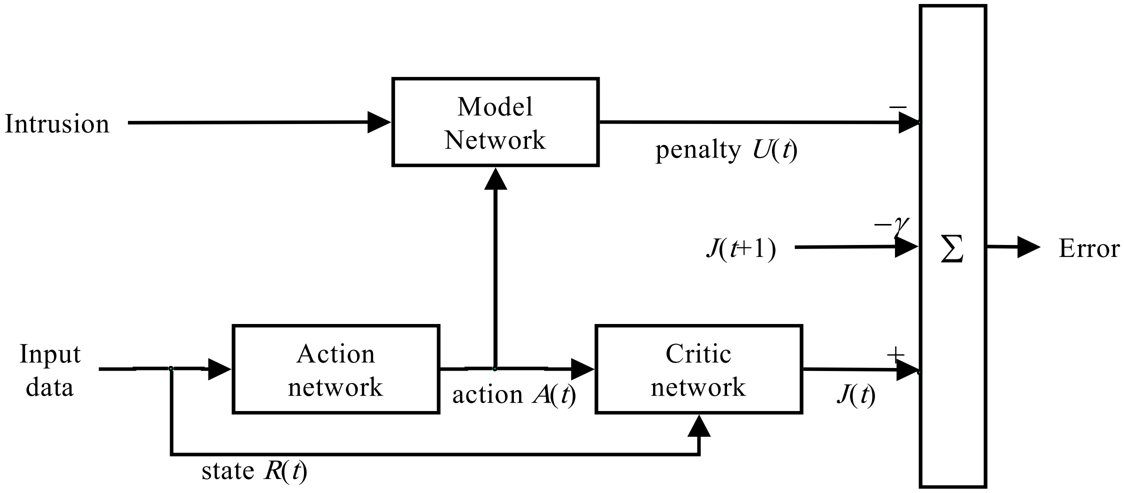


Figure 13. Improved adaptive critic design for HDP training.

4.3.4. Final Results

Figures 14–17 show the ability of an ACD-based IDS to learn a training set. One epoch of training data consisted of the following exploit and clean data sets (see **Table 2** for a description of the data sets):

Clean–Fd–Clean–Clean2–Rdist–Clean2–Clean–Sdtcm

The action network of the ACD-based IDS had three hidden layers, each with 10 nodes, and the critic network had three hidden layers, each with 5 nodes. Each training iteration consisted of 5 epochs of critic training and 5 epochs of action network training. The resilient back-propagation algorithm was used to update weights. Inputs were preprocessed with principal components analysis, which reduced the input dimension from 35 to 17.

Network responses are plotted as dots, and target outputs are plotted as a solid line. Note that the ACD's response to intrusions is easily distinguished from its response to normal behavior, in spite of the fact that the outputs are not always close to target outputs. We also noted a shift in the effective output range as training continued; this appeared to be an artifact of the ACD training iterations.

The plots show the generalization performance of the network after 10 and then after 20 iterations. The clean data is a mix of some clean training data with "clean3" sessions that the network had not previously seen. There are two exploits, neither of which was in the training set. The longer one is the "sunkill" DOS attack (with data from the *attacker's* machine), the other shorter one at the end of the test run is the "fdformat" buffer overflow attack. As with the training data, the neural net's response to intrusions is distinguishable from its response to normal behavior.

Figures 18 and 19 show results from the same ACD using slightly different numbers of training epochs on different training data. Each iteration consisted of 25 epochs of critic training and 50 epochs of action-network training. In this case, one epoch of training data consisted of the Fdformat, T4, Fd, Eject, Ufsrestore, Sdtcm, and Rdist exploits, with each exploit separated by a combination of the Clean, Clean2, and Clean3 data sets. The test data is simply a replica of the training except for the Fdformat exploit moved in time. The ACD is clearly able to discriminate between all exploits and the clean data. However, **Figure 20** shows the difficulty the ACD had in learning the Variant data set where exploit-similar activity is introduced into the normal data. The ACD output (shown as red or dark gray dots) does not closely follow the target (shown as a solid gray line) and is particularly in error in the normal fragments of the data.

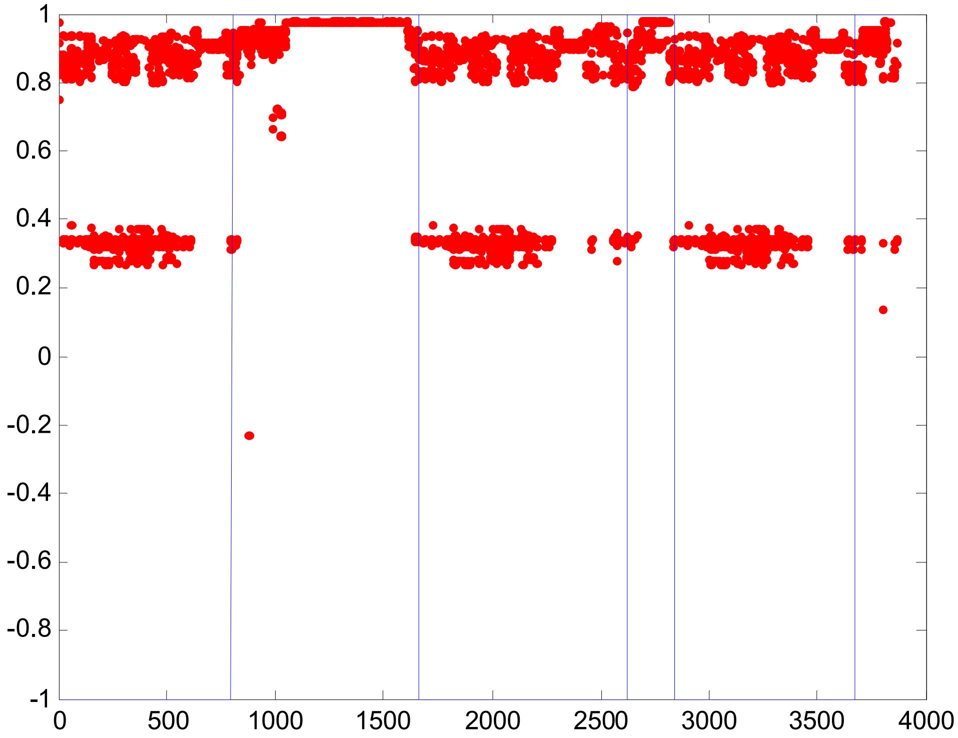


Figure 14. ACD training results after 10 iterations. The training data consists of the following exploits in order: Fd-Rdist-Sdtcm.

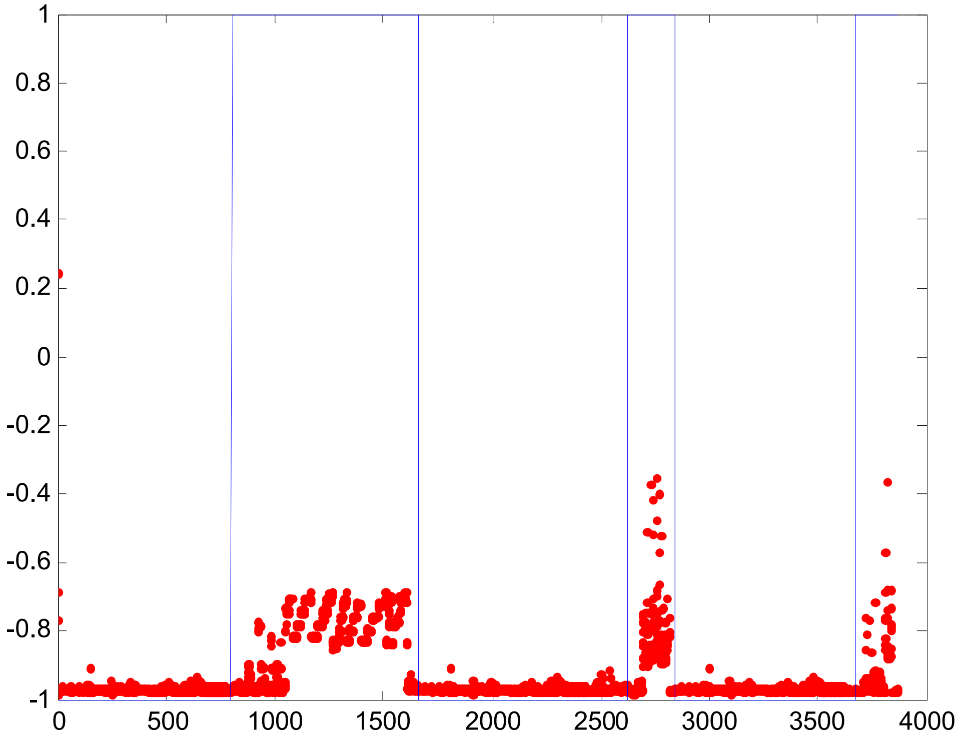


Figure 15. ACD training results after 20 iterations. The training data consists of the following exploits in order: Fd-Rdist-Sdtcm.

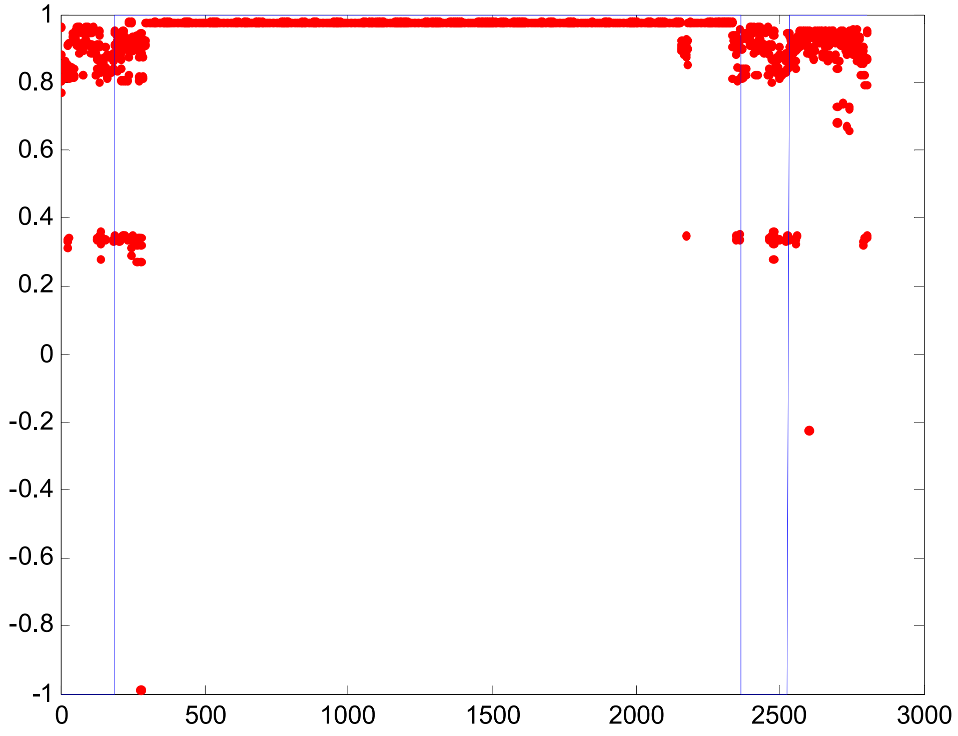


Figure 16. ACD test results after 10 training iterations. The test data consists of the following exploits in order: Sunkill-Fdformat.

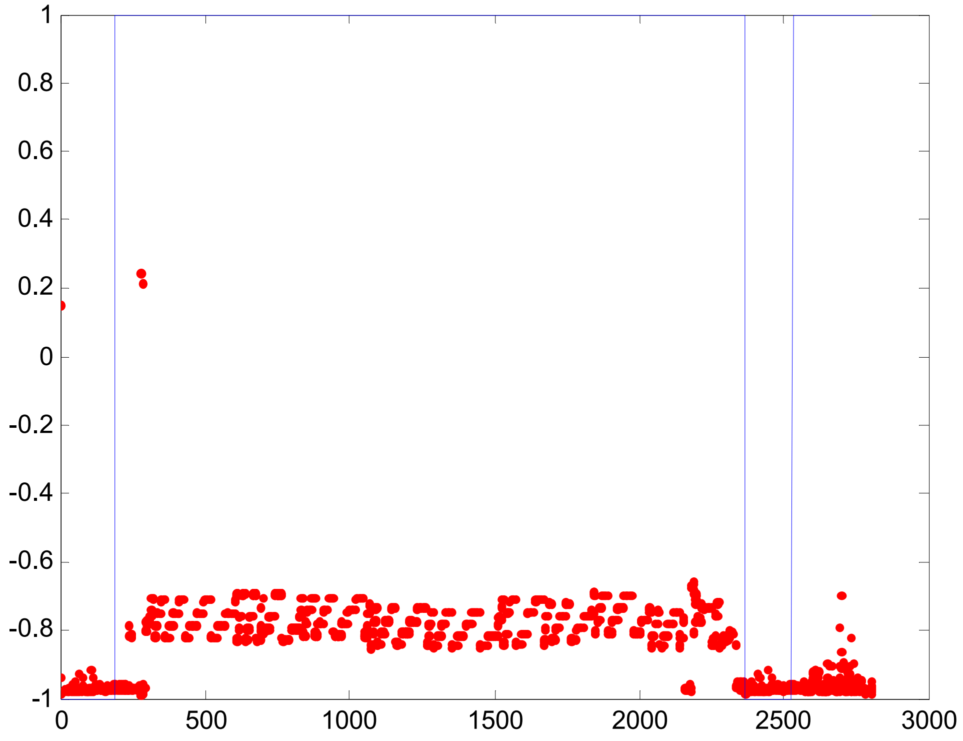


Figure 17. ACD test results after 20 training iterations. The test data consists of the following exploits in order: Sunkill-Fdformat.

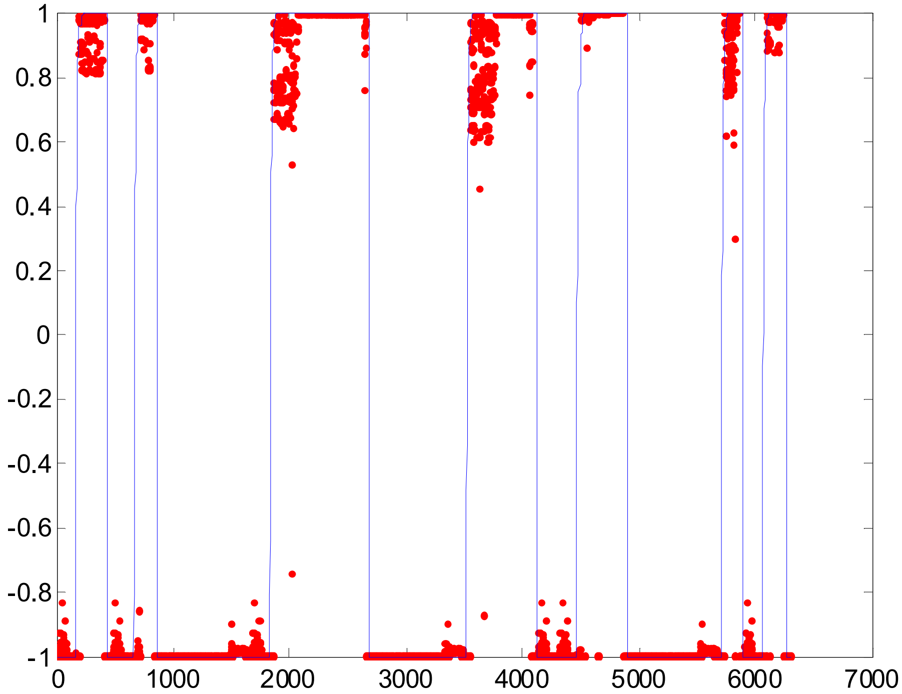


Figure 18. **ACD training results on multiple exploits.** The training data consists of the following exploits in order: Fdformat-T4-Fd-Eject-Ufsrestore-Sdtcm-Rdist.

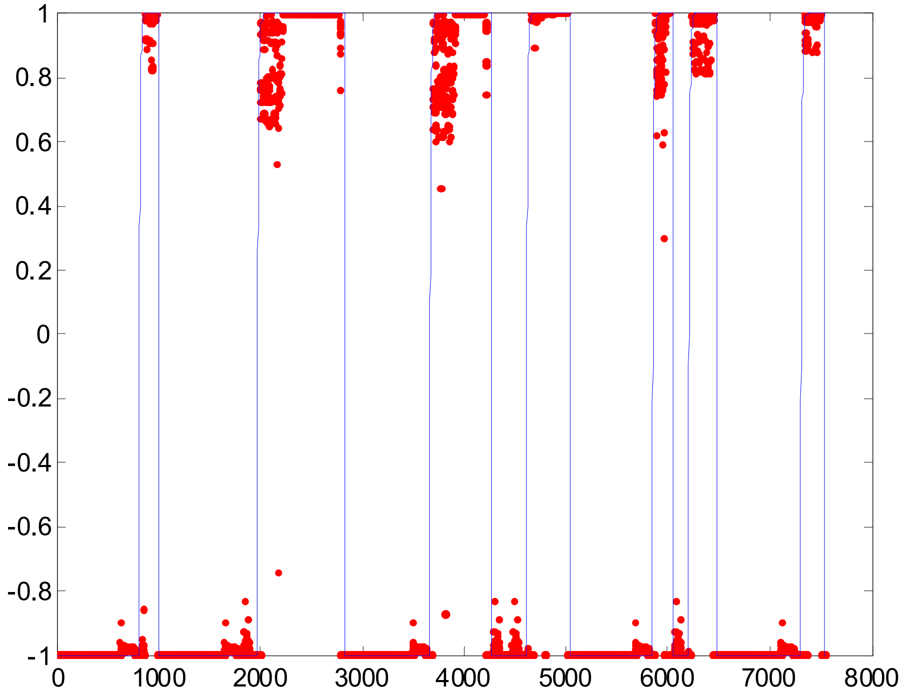


Figure 19. **ACD test results on multiple exploits.** The test data consists of the following exploits in order: T4-Fd-Eject-Ufsrestore-Sdtcm-Fdformat-Rdist.

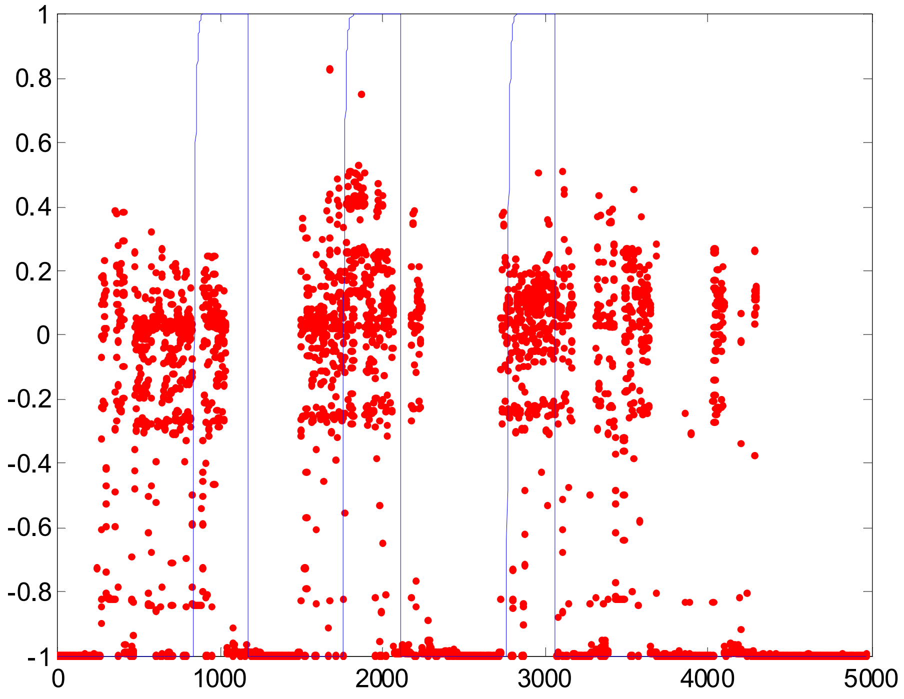


Figure 20. ACD training results on Variant data set.

4.4. Anomaly Detection of Intrusions

This section documents the work associated with the use of BSM audit event types as a basis for network anomaly detection. This work follows from the approach described by Ye et al. [**YELC01**]. The data used to investigate this approach are from MIT's DARPA 1999 Intrusion Detection Evaluation [**HLFZTB01**].

Development and evaluation of any anomaly detection approach requires both training data (assumed to consist exclusively of intrusion-free network activity) and test data (normal network activities interleaved with a number of intrusive sessions). For this investigation, DARPA data from Friday of Week#1 comprise the training data set and DARPA data from Monday of Week#4 comprise the test data set.

The training (test) data set consists of 141 (199) sessions, each of which is comprised of a sequence of system level calls (events) that are recorded by the auditing system. For both the training and test data sets, the number of system level calls per session is typically in the range from one to several thousand. The 44-byte feature vector that was generated provides a 2-byte representation of the event type. The high-order byte of this pair is nearly always set to zero. Thus, the low-order byte gives a good (nearly unique) representation of the event type and is used in the analysis that follows. For example, a relatively common event type, represented by the mnemonic AUE:IOCTL, is associated with event ID number 158. **Figure 21** displays the logarithm of the fraction of events of each type within the training set. Note that there are only 55 event types represented within the training set. **Figure 22** indicates how the fraction of events within each event type varies across sessions. For example, the fraction of events associated with event ID number 133 within a session varied from 0.00038 to 0.075.

The premise underlying the work by Ye et al. [**YELC01**] is that the local occurrence frequencies of the various event types may provide an indication that a specific session is anomalous and therefore potentially intrusive. For example, a large number of rare events occurring in a relatively short sequence of events would by definition be anomalous. Specifically, Ye et al. create a multivariate observation vector for the t^{th} event in the event stream: $\boldsymbol{O}_t = (O_{1,t}, O_{2,t} \ldots, O_{284,t})$ for $t > 0$, where $O_{i,t} = \lambda \times \eta_i + (1-\lambda) \times O_{i,t-1}$ and $O_{i,0} = 0$, and where

η_i is an indicator function that takes the value 1 if event type i is present in the current observation, and 0 if event type i is not present. Ye et al. recommend that the smoothing parameter, λ, be set to 0.3. So, each event type indexed by i has its own exponentially weighted moving average (EWMA) process denoted by $O_{i,t}$. Evidently, as described by Ye et al., the event stream is ordered by time and includes events in concurrent time-interleaved sessions.

The high-dimensional observation vector O_t is mapped to a one-dimensional space by computing the chi-squared distance from the current observation to the mean of the observations in the training data set. That is, $\chi_t^2 = \sum_{i=1}^{k} \frac{(O_{i,t} - \overline{O}_i)^2}{\overline{O}_i}$, where $\overline{O}_i$ is the proportion of all events within the training set that are of the i^{th} type. An alert is generated when χ_t^2 exceeds a prescribed threshold. A problem with this metric is the singularity when $\overline{O}_i = 0$. There are numerous ways to resolve this problem. For example, one could simply eliminate (from the summation) those event types that do not appear in the training set. However, this would eliminate the sensitivity of this metric (when applied prospectively to future data) to those rare event types that are not represented in the training set. Another approach would be to add a small amount to $\overline{O}_i$ (say, $\Delta = 10^{-6}$), such that $\chi_t^2 = \sum_{i=1}^{k} \frac{(O_{i,t} - (\overline{O}_i + \Delta))^2}{(\overline{O}_i + \Delta)}$.

A more fundamental problem with this approach is that the event stream is not localized to a session. The effective signal from activities associated with an intrusive session could be dampened by events from concurrent nonintrusive sessions. A more sensitive variation of this approach would be to establish separate EWMA processes and chi-squared metrics for each session.

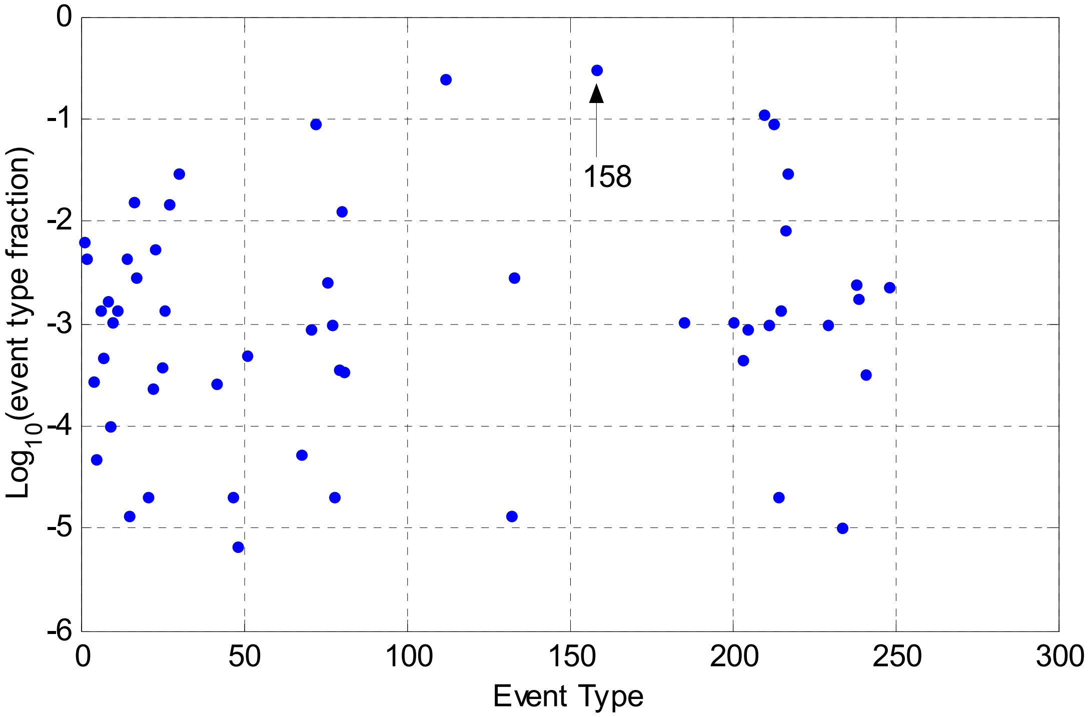


Figure 21. Fraction of events by type in training set.

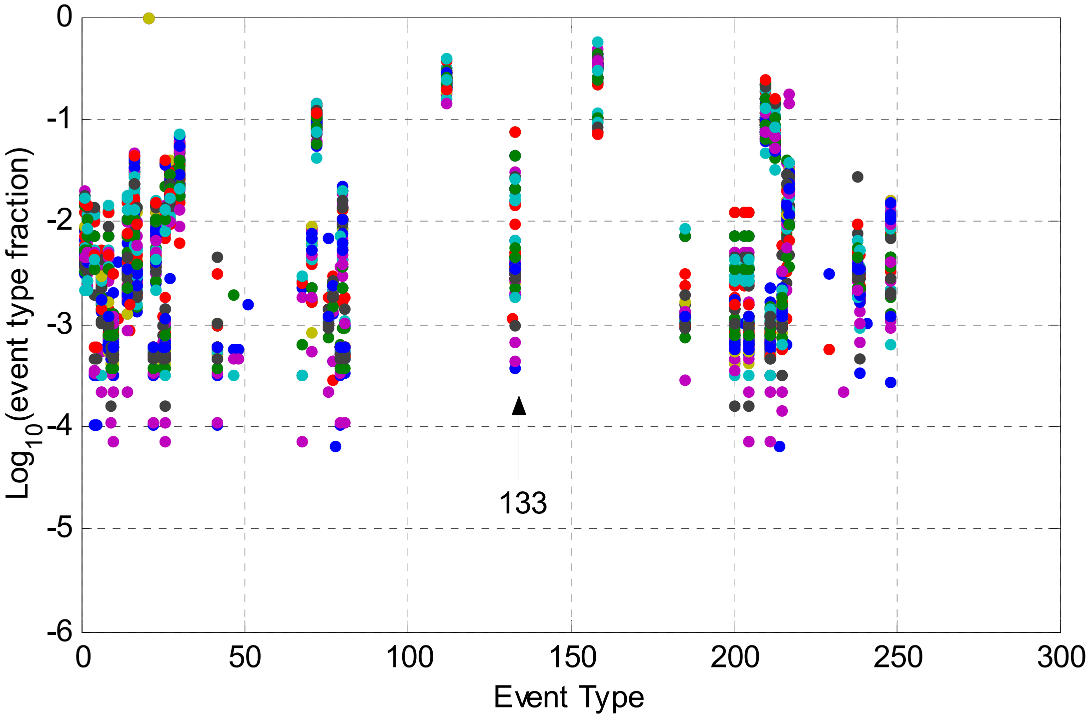


Figure 22. Distribution of fraction of events by type per session in training set.

4.4.1. Proposed Anomaly Detection Method

The method proposed here develops a single EWMA process for each session. Let E_t be the event type of the t^{th} event within a session. Let $X_t = -\log_{10}\left(\overline{O}_{E_t} + 10^{-6}\right)$, where $\overline{O}_{E_t}$ is the proportion of all events within the training set that are of the E_t^{th} type. The EWMA process is defined as $Y_t = \lambda \cdot X_t + (1-\lambda) \cdot Y_{t-1}$, where $Y_0 = 0$ and $0 < \lambda < 1$. For purposes of illustration, we will set $\lambda = .3$. Smaller values for λ will increase the memory of the system while larger values for λ will decrease the memory of the EWMA process. An alert is generated (signifying an abnormal session) when Y_t exceeds some threshold that is developed based on the training set.

The DARPA data is used to illustrate the method. **Figure 23** displays the maximum value of the EWMA process obtained during each session within the training set. **Figure 24** displays a histogram of the distribution of these values. Assuming the training set sessions are representative of future nonintrusive network activity, the distribution of values in **Figure 24** can be used to set a threshold for generating an alert. Here, depending on limitations of false alarms, one might want to set a threshold at somewhere in the neighborhood of 3.5 to 4 (for purposes of analyzing the test set, a threshold was set at 4). **Figure 25** displays the maximum values of the EWMA for each session within the test set. Given a threshold at 4, we find two sessions that generate an alert (sessions 13 and 187). **Figures 26** and **27** display the EWMA processes for each of these sessions. In the case of Session 13, there are a number of rare events that take place close in time in the vicinity of events 1000–1020. These relatively rare events have event identification numbers 48 (AUE_RMDIR), 4 (AUE_CREAT), and 202 (AUE_UTIME). In the case of Session 187, there are a number of consecutive instances of the rare event ID 15 (AUE_KILL) in the vicinity of events 1090–1120.

Examination of known attacks reported by DARPA [**HLFZTB01**, p. 54, tables 6 and 7] confirms that Session 13 is a detection of Attack ID no. 41.084031. It is also interesting to consider sessions 104–108, which are known to be part of a multisession attack sequence. Each of these sessions consists of a single event with event ID 21. In this case, the full 2-byte representation of the event IDs is 6165 (AUE_ftpd:ftp access:lo). Event ID 21 occurred six times in 303,289 total events within the training set, giving a probability of occurrence of $\overline{O}_{E_t} = 1.98\text{x}10^{-5}$. Each of these six events within the training set corresponded to a full 2-byte representation of the event ID of 6165. Thus, the single-byte specification was sufficient in this case. The value for the EWMA in these cases is $-.3 \cdot \log_{10}\left(\overline{O}_{E_t} + 10^{-6}\right) = -1.405$.

As is clear in **Figure 24**, based on its relatively low value, this value for the EWMA is not indicative of unusual activity. Assuming these sessions involve the same user, a modification of the proposed strategy is suggested, i.e., develop the EWMA over successive sessions involving the same user. In this case, such a strategy applied to sessions 104–108 would result in a value for the EWMA of –3.90 at the fifth session. Such a value would be considered at least marginally significant.

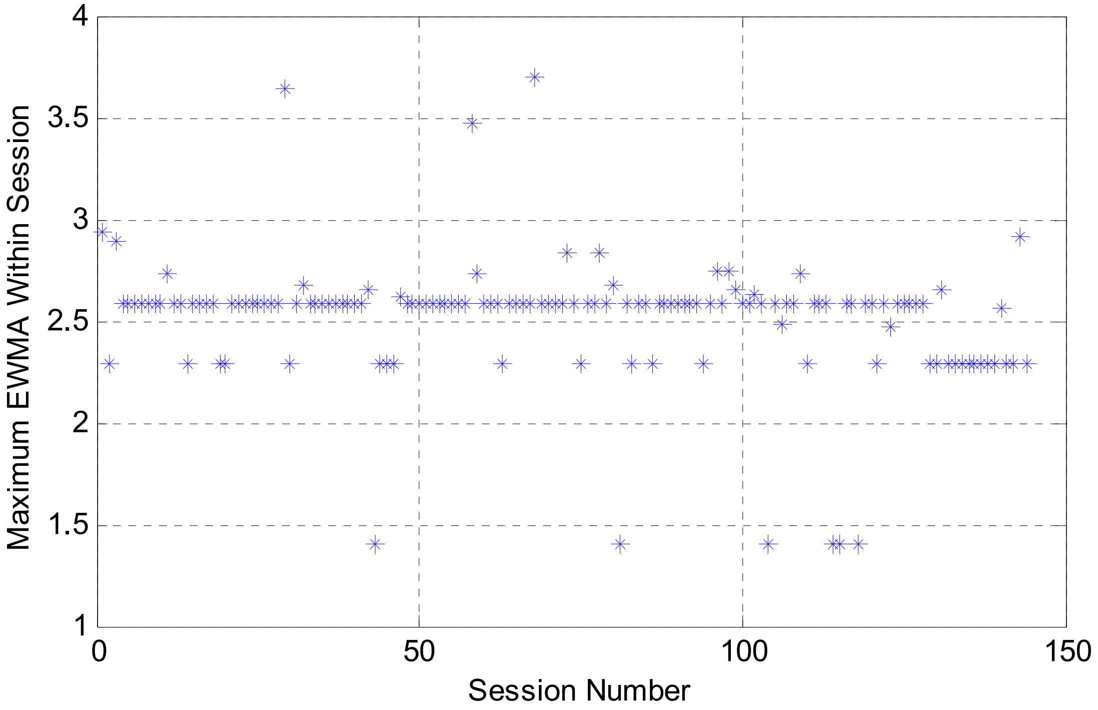


Figure 23. Maximum value of EWMA within each session of training set.

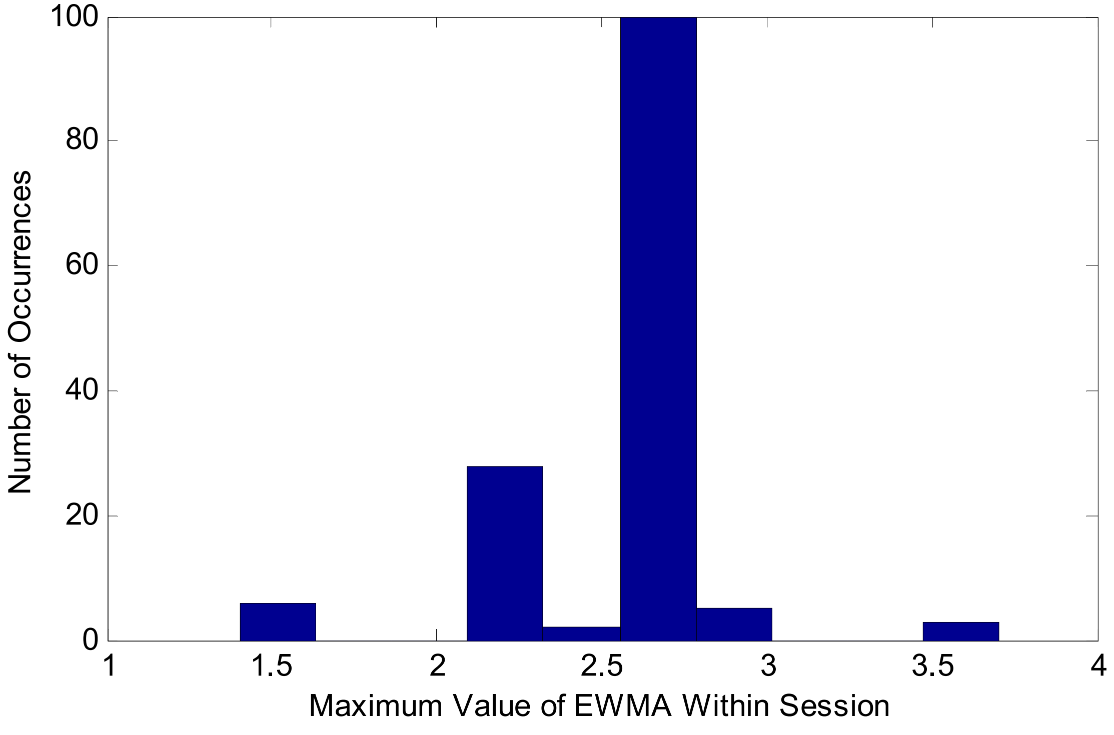


Figure 24. Distribution of maximum value of EWMA within each session of training set.

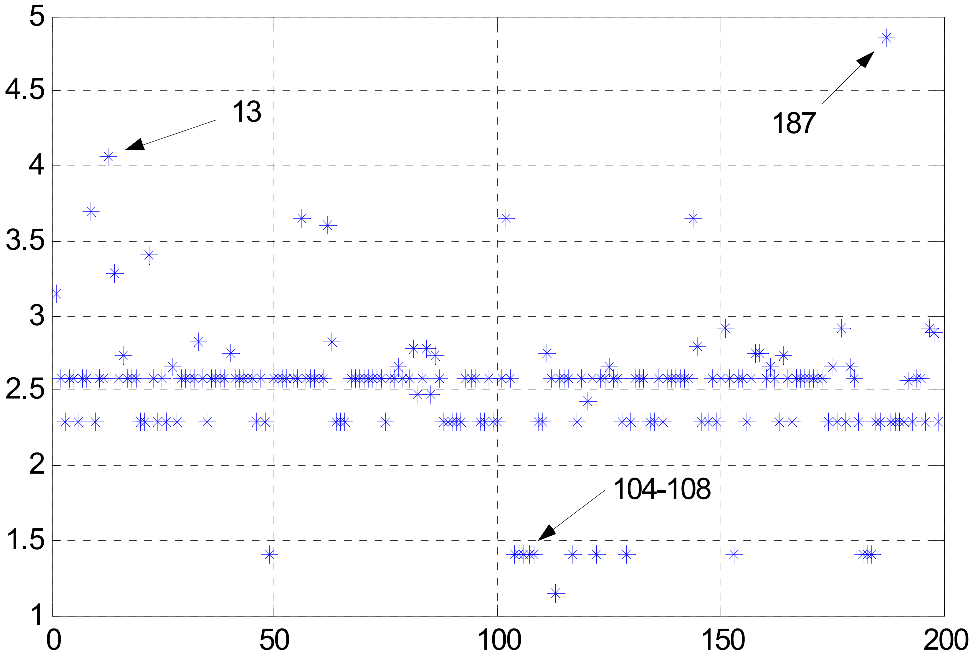


Figure 25. Maximum value of EWMA within each session of test set.

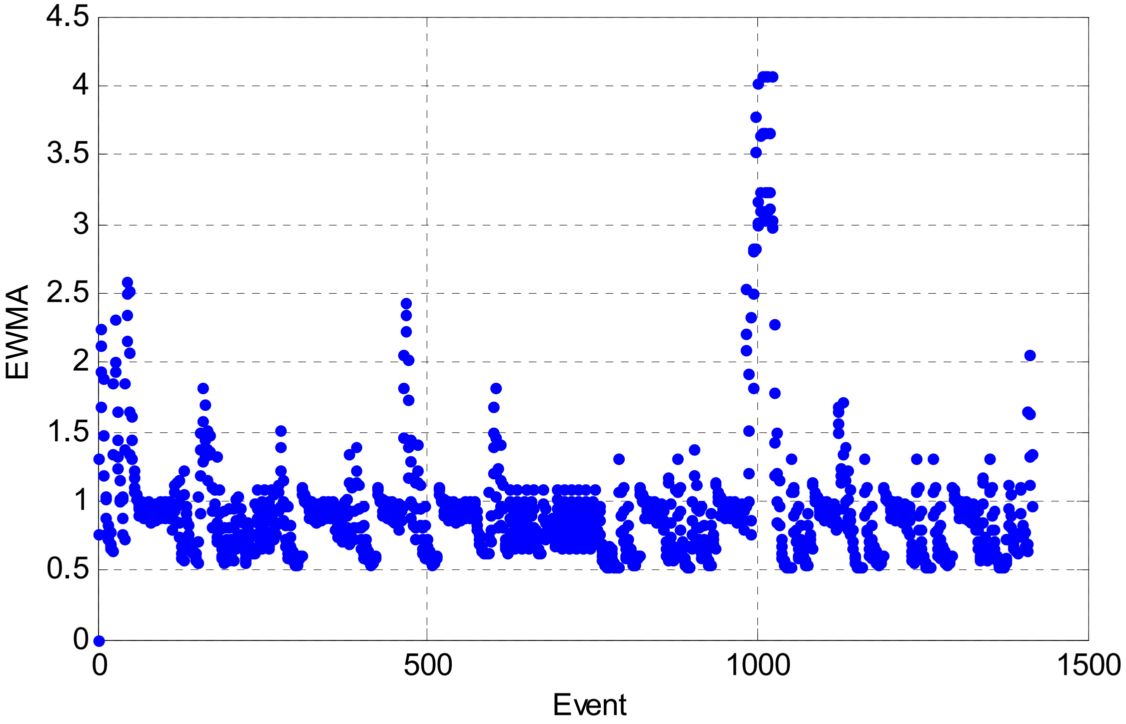


Figure 26. EWMA process for Session 13.

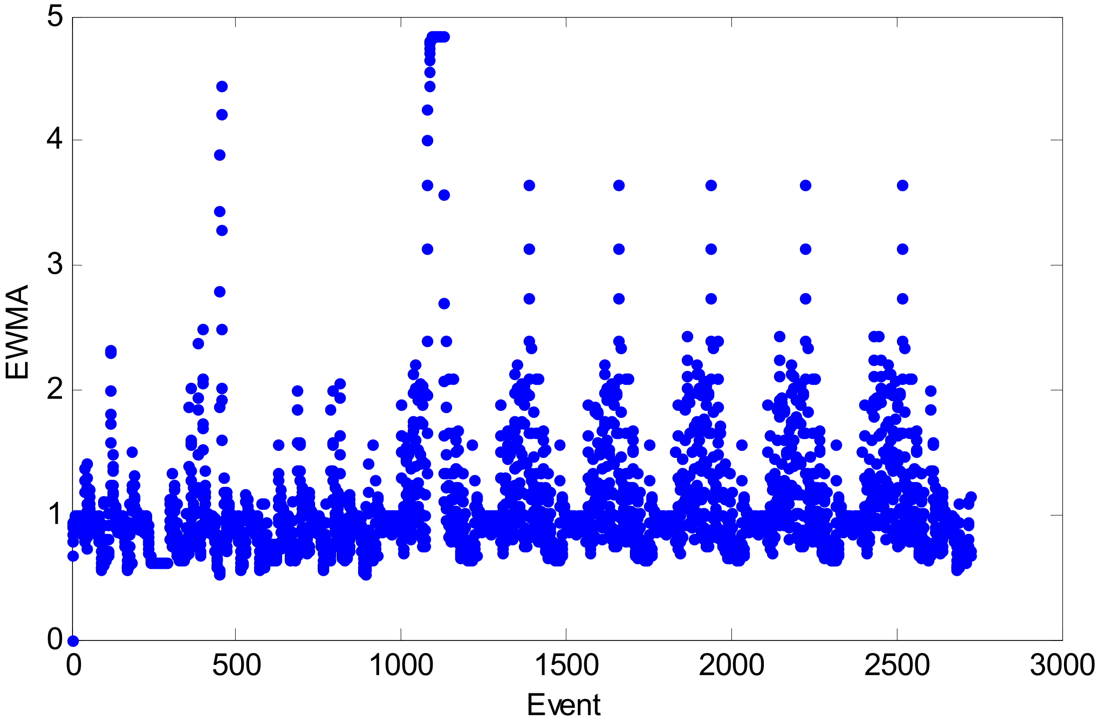


Figure 27. EWMA process for Session 187.

4.4.2. Modifications to Proposed Anomaly Detection Method

Admittedly, the proposed detection method could be developed further. An alert is generated only when there is a relatively high concentration of rare events in a local window such that the EWMA exceeds a certain threshold. The window size (and hence sensitivity to the current event) could be adjusted by varying λ. What is an appropriate value for λ in this context remains to be seen. One also could develop more objective rules for constructing an appropriate threshold based on target false alarm rates. One might also consider other rules for generating an alert, e.g., if the EWMA remains above a certain threshold (different than the threshold above) for a certain number of events.

In a broader sense, one could use X_t in conjunction with some type of control-charting technique other than the EWMA. Examples of other techniques include the cumulative sum and moving average techniques, as reported by Wetherill and Brown [**WB91**]. It is not clear what advantages, if any, each technique might offer over others in this context.

The proposed method is sensitive to a high local density of rare events. The sequence of probabilities of individual events is used to generate an alert. It may be, however, that the probability of a sequence of events is more informative. For example, Event ID V closely followed by Event ID W may be a strong indicator of intrusive activity. However, the implementation of a strategy that provides an alert based on the probability of a sequence of events may be problematic due to difficulties associated with enumerating possible sequences of events.

4.4.3. Conclusions Regarding Anomaly Detection Strategies

The method described here outlines an empirical approach for anomaly detection that does not depend on knowledge of subject matter. It is clear that the method would be vulnerable to a coordinated attack over multiple sessions. Even in its domain of applicability, the efficacy of the approach is unknown. Clearly, as is generally the case for anomaly detection methods, the efficacy of the approach will improve as the training set is enlarged to span a wider variety of normal nonintrusive activities.

5. Conclusions

Intrusion detection is a need and a problem that is not going to go away soon. The challenge of detecting novel exploits and attacks deserves special attention. The preliminary results presented in this report confirm the ability of neural networks and statistical anomaly detectors to learn the differences between clean and exploit data.

To our knowledge, ACDs have not been used for intrusion detection until now. Preliminary results suggest that this approach to ID deserves more attention. Although our research barely scraped the surface of the potential of adaptive critic designs, it yielded promising initial results.

The foremost lesson learned from this research is that much work will be required in order to achieve effective detection of novel exploits. The second most important lesson learned from this research is that IDS control is probably not truly useful. Although the concept of IDS control is promising, practically speaking, a better course of action appears to be to incorporate the most effective mechanisms into the IDS.

The third and final lesson learned from this research is that with current ID sensors, considerable effort must be devoted to data preprocessing before presentation to a neural network. Even more importantly, the availability of new sensors designed from the ground up for automated ID would dramatically improve the possibility of detecting intrusions.

In addition to ID sensor research and development, other avenues for further research activities have been identified. They include

- **Performing extensive signature-generalization data collection**

  The creation of rich data sets that represent classes of exploits will allow generalized signature-based ID to discriminate between different kinds of exploits, including failed exploit attempts. This idea should be expanded to other ID sensors besides BSM.

- **Realizing the full potential of ACDs for intrusion detection**

  Even though the Elman recurrent neural network produced comparable if not superior results to the ACD-based IDS, it is expected that the situation will reverse as the Elman neural networks are tested on more challenging problems. The following aspects of ACDs for intrusion detection warrants further research.

- **Taking full advantage of reinforcement learning**

  The use of reinforcement learning techniques in ACDs offers great promise in the ID environment. The ability to learn about the existence of an exploit even when its precise location is unknown is significant.

- **Developing cost-oriented decision-making**

  When put into practice, many costs are associated with ID (see **Section 1.1**). An IDS that can make decisions based on a cost policy established for a particular site would offer a great benefit. ACDs have the potential to do just that.

- **Continuing anomaly detection research**

  The statistical anomaly detection work presented in **Section 4.4** successfully detected exploits in DARPA’s ID evaluation data, yet many potential improvements are available. Anomaly detectors will continue to be valuable in detecting novel exploits.

6. References

[B00] R.G. Bace, 2000, *Intrusion Detection*, p. 3. Macmillan Technical Publishing, Indianapolis, IN.

[C00] J. Cannady, 2000, “Applying CMAC-based online learning to intrusion detection.” *International Joint Conference on Neural Networks, 2000 (IJCNN 2000)*, S.I. Amari, C.L. Giles, M. Gori, V. Piuri, ed., vol. 5, pp. 405–410.

[C98] J. Cannady, 1998, “Artificial neural networks for misuse detection.” *Proceedings of the 1998 National Information Systems Security Conference (NISSC ’98)*, pp. 443–456, Arlington, VA.

[E98] D. Endler, 1998, “Intrusion detection. Applying machine learning to Solaris audit data.” *Proceedings of 14th Annual Computer Security Applications Conference*, pp. 268-279.

[GS99] A.K. Ghosh and A. Schwartzbard, 1999, “A study in using neural networks for anomaly and misuse detection.” *Proceedings of the 8th USENIX Security Symposium***,** August 1999.

[GSS99] A.K. Ghosh, A. Schwartzbard, and M. Schatz, 1999, “Learning program behavior profiles for intrusion detection.” *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pp. 51–62, Santa Clara, CA, April 1999.

[GWC98] A.K. Ghosh, J. Wanken, and F. Charron, 1998, “Detecting anomalous and unknown intrusions against programs.’ *Proceedings of the 1998 Annual Computer Security Applications Conference* (ACSAC ’98), December 1998.

[HCDB99] H. Hinton, C. Cowan, L. Delcambre, S. Bowers, 1999, “SAM: Security Adaptation Manager.” *Proceedings of the 15th Annual Computer Security Applications Conference*, pp. 361–370, Phoenix, AZ.

[HFS98] S.A. Hofmeyr, S. Forrest, and A. Somayaji, 1998, “Intrusion detection using sequences of system calls.” *Journal of Computer Security*, vol. 6, pp. 151–180.

[HHS00] A.J. Hoglund, K. Hatonen, A.S. Sorvari, 2000, “A computer host-based user anomaly detection system using the self-organizing map.” *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000*, S.I. Amari, C.L. Giles, M. Gori, V. Piuri, ed., vol. 5, pp. 411–416.

[HLFZTB01] J.W. Haines, R.P. Lippmann, D.J. Fried, M.A. Zissman, E. Tran, and S.B. Boswell, 2001, *1999 Intrusion Detection Evaluation: Design and Procedures*. ESC-TR-99-061, Technical Report 1062, Lincoln Laboratory. Massachusetts Institute of Technology, Cambridge, MA.

[K99] K. Kendall, 1999, *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems.* Massachusetts Institute of Technology, Cambridge, MA (master’s thesis).

[L00]	R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman, 2000, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation." *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX)*, vol. 2, Hilton Head, SC, January 2000. IEEE Computer Society Press, Los Alamitos, CA.
[NSD99]	N. Nuansri, S. Singh, and T. Dillon, 1999, "A process state-transition analysis and its application to intrusion detection." *Proceedings of the 15th Annual Computer Security Applications Conference*, pp. 378-387, Phoenix, AZ.
[PW97]	D. Prokhorov and D. Wunsch, 1997, "Adaptive Critic Designs." *IEEE Transactions on Neural Networks* 8, pp. 997–1007.
[RLM98]	J. Ryan, M.J. Lin, R. Miikkulainen, 1998, "Intrusion Detection with Neural Networks." *Advances in Neural Information Processing Systems 10*, M.I. Jordan, M.J. Kearns, S.A. Solla, ed., pp. 943–949. MIT Press, Cambridge, MA.
[S00]	Sun Microsystems, 2000, *SunSHIELD Basic Security Module Guide*. Sun Microsystems, Inc., Palo Alto, CA.
[SF01]	S. Forrest, 2001, personal. communication. University of New Mexico, Albuquerque, NM.
[SFLPC00]	S. Stolfo, W. Fan, W Lee, A. Prodromidis, and P. Chan, 2000, "Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project." *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX)*, vol. 2, Hilton Head, SC, January 2000. IEEE Computer Society Press, Los Alamitos, CA.
[VEK00]	G. Vigna, S. Eckmann, and R. Kemmerer, 2000, "The STAT Tool Suite." *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX)*, vol. 2, Hilton Head, SC, January 2000. IEEE Computer Society Press, Los Alamitos, CA.
[WB91]	G.B. Wetherill and D.W. Brown, 1991, *Statistical Process Control Theory and Practice*. Chapman and Hall, London, England.
[YELC01]	N. Ye, S.M. Emran, X. Li, and Q. Chen**,** 2001, "Statistical Process Control for Computer Intrusion Detection." *Proceedings of the 2001 DARPA Information Survivability Conference and Exposition (DISCEX)*, vol. 1, Anaheim, CA, June 2001. IEEE Computer Society Press, Los Alamitos, CA.

7. Appendix A—Details of the Solaris BSM

A common source of data for IDSs is an audit mechanism for a host computer. Sun Microsystems provides an auditing facility called Basic Security Module (BSM) with its Solaris operating system. BSM keeps a log of ongoing system activities in *audit trail files*. The audit files contain *audit records*, which contain information about one *audit event* (a system call). Possible events are defined in /etc/security/audit_event. The structure of an audit record is described by Sun Microsystems [**S00**]. An audit record usually comprises several *audit tokens*.

- Each record starts with a *header token* containing information about the type of event, the time when the event occurred, and the length of the record in bytes. Usually, a record also contains a subject token and a return token.
- The *subject token* stores identification data: effective and real user IDs, effective and real group IDs, process IDs, session IDs, machine and port IDs, and audit user IDs. The same session ID is assigned to a session at login and to all of its offsprings, allowing the opportunity to divide the audit data into sessions. Similarly, the audit user ID does not change, and keeps the user ID of the original user who has logged in.
- The *return token* stores the return value for the system call described by the record, and the error number, if any.
- Other widely used tokens are *arg tokens*, *exec_args tokens*, *path tokens*, *attribute tokens*, and *process tokens*. A *text token* stores a text string passed in the system call. When dealing with remote machines (e.g., during a telnet session) *in_addr*, *iport*, and *socket_inet* tokens are used.
- The *arg token* contains an argument of the system call. It is a 32-bit value, and an optional text string of variable length. There are as many argument tokens in a record as are needed by the system call.
- The *exec_args token* keeps text strings passed as arguments to an exec system call. Each argument is represented as a variable-length string.
- The *path token* stores a variable-length text string with a path to a file system object. If there are several paths used in the system call, the record contains several path tokens.
- If a path token contains a valid file system path, the path token is followed by the *attribute token*. The attribute token contains file system attributes of the object referred to by the path in the path token.
- The *process token* is similar to the subject token and is used when the system call is dealing with a process (e.g., kill).
- *In_addr* and *iport* tokens store numbers that represent an IP address and a port used in the call. The *socket_inet token* describes a socket connection to a local port and keeps an address of the port and an IP address of remote machine.

7.1. BSM Statistics

A brief statistical analysis performed on three BSM audit files is presented in this appendix. The following files were analyzed:

- **1998 DARPA**—The BSM audit file from Tuesday, Week 1 of the 1998 DARPA ID evaluation test data **[L00]**.

- **2000 DARPA**—The BSM audit file 20000307142426.20000207173942.mill.bsm from the 2000 DARPA ID evaluation data.

- **rdist**—custom BSM data acquired on a Sun workstation running Solaris 2.5 using a remote file distribution utility exploit.

Table 9 contains the labels and hexadecimal values of all of the tokens present in the analyzed audit files.

Table 9. Labels and hexadecimal identifiers for BSM tokens.

Token Identifier	**Token Label**
11 hex	Audit File
13 hex	Trailer
14 hex	Header
23 hex	Path
24 hex	Subject
26 hex	Process
27 hex	Return
28 hex	Text
2a hex	Internet Address
2c hex	TCP/UDP Port Address
2d hex	System Call Arguments
2e hex	Socket
31 hex	Attribute
3c hex	Exec System Call Arguments
3e hex	Attribute 32
72 hex	Return 64

Table 10 contains the token frequencies in each of the analyzed files. Note that the number of audit records in the file can be measured with the number of header tokens (14 hex) plus the number of audit file tokens (11 hex).

Table 10. Token frequencies in the analyzed audit files.

Token	DARPA 1998	DARPA 2000	rdist
Identifier	Token Count	Token Count	Token Count
11 hex	2	2	2
13 hex	2114291	104907	
14 hex	2114291	104907	418
23 hex	1666366	57355	372
24 hex	2114291	104907	418
26 hex	229739	711	1
27 hex	2114291	104791	418
28 hex	579	30	
2a hex	307		
2c hex	307		
2d hex	1669290	183663	759
2e hex	540	68723	
31 hex	1599489		368
3c hex	15797	551	
3e hex		54285	
72 hex		116	

Tables 11, **12**, and **13** list the frequency of token patterns contained in all the records for each of the corresponding audit files. For example, 16,648 records in the 1998 DARPA data [**L00**] were represented by the series or pattern of tokens 23 3E 2D 2D 24 27 13. The variability in the content and length of audit records within a file is obvious from these tables.

Table 11. Frequency of token patterns in the 1998 DARPA audit file.*

Count	Pattern
16648:	23 3E 2D 2D 24 27 13
5776:	23 3E 2D 2D 2D 24 27 13
4279:	2D 2D 24 27 13
4342	23 24 27 13
17920:	23 3E 24 27 13
4456:	2D 24 27 13
550:	23 3E 3C 24 27 13
11519:	2D 23 3E 24 27 13
2137:	24 27 13
43:	2D 2D 2D 2D 2D 2D 24 27 13
711:	2D 26 24 27 13
97:	23 3E 24 72 13
38:	2D 2D 2D 2D 2D 24 27 13
31:	2D 2D 2D 24 27 13
59:	2D 23 3E 2D 24 27 13
15:	2D 3E 24 72 13
479:	2D 3E 24 27 13
20:	24 28 27 13
237:	23 3E 2D 24 27 13
1:	2D 2D 2D 2D 2D 2D 2D 24 27 13
102:	2D 2D 2D 2D 24 27 13
29:	2D 2E 2E 24 27 13
169:	2D 3E 2D 2D 2D 24 27 13
96:	2E 2D 2D 2D 2D 24 27 13
49:	2E 2D 2D 2E 2D 24 27 13
37:	2D 2D 23 3E 24 27 13
575:	2D 3E 2D 2D 24 27 13
46:	23 23 3E 24 27 13
22:	2D 2D 2D 2D 2E 24 27 13
107:	2D 2D 3E 24 27 13
17:	23 3E 23 24 27 13
4:	2D 23 3E 23 3E 24 72 13
65:	2D 2E 24 27 13
6:	2E 2E 24 27 13
2	24 28 28 27 13
1:	2E 2D 2D 24 27 13
1:	2E 2E 2D 24 27 13
34175:	2E 2D 2D 2D 2E 24 27 13
8:	23 3E 23 3E 24 27 13
2:	2D 23 23 3E 24 27 13
3:	24 23 27 13
5:	2D 23 24 27 13
6:	28 23 3E 24 27 13
17:	2D 2D 2E 24 27 13
1:	23 3E 3C 2D 2D 24 27 13
2:	2E 24 27 13
1:	2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 24 27 13

*** L00**

Table 12. Frequency of token patterns in the 2000 DARPA audit file.

Count	Pattern
30320:	24 27 13
79436:	23 31 2D 2D 2D 24 27 13
72121:	2D 24 27 13
513591:	2D 23 31 24 27 13
796996:	23 31 24 27 13
15797:	23 31 3C 24 27 13
65916:	23 24 27 13
112487:	2D 2D 24 27 13
175964:	23 31 2D 2D 24 27 13
8996:	23 31 2D 24 27 13
307:	24 28 2A 2C 27 13
591:	2D 2D 23 31 24 27 13
720:	2D 2D 2D 2D 2D 2D 24 27 13
272:	24 28 27 13
1505:	2D 2D 2D 24 27 13
1155:	2D 2D 2D 2D 24 27 13
6079:	2D 23 31 2D 24 27 13
519:	23 31 23 24 27 13
540:	2D 2D 2E 24 27 13
581:	23 31 23 31 24 27 13
206:	2D 2D 2D 2D 2D 24 27 13
60:	2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 24 27 13
349:	23 23 31 24 27 13
37:	23 23 24 27 13
4:	2D 23 24 27 13
1:	23 31 2D 23 23 31 23 23 31 24 27 13
1:	23 31 23 23 31 23 23 31 23 23 23 23 23 31 23 23 23 23 23 31 23 23 23 24 27 13
229739:	2D 26 24 27 13

Table 13. Frequency of token patterns in the rdist audit file.

Count	Pattern
10	2D 23 31 24 27
15	23 31 24 27
343	23 31 2D 2D 24 27
1	2D 26 24 27
30	2D 24 27
4	24 27
5	2D 2D 2D 2D 24 27
6	2D 2D 24 27
3	23 24 27

DISTRIBUTION:

5 Dr. Donald Wunsch
University of Missouri-Rolla
Department of Electrical & Computer Engineering
131 Emerson Electric Co. Hall
131 Miner Circle
Rolla, MO 65409-0040

1	MS0428	D. D. Carlson, 12300
1	MS0785	D. P. Duggan, 6516
5	MS0785	T. J. Draelos, 6514
1	MS0785	M. J. Collins, 6514
1	MS0784	R. C. Parks, 6512
1	MS0785	M. E. Senglaub, 6516
1	MS0785	D. L. Harris, 6516
1	MS0784	R. E. Trellue, 6501
1	MS0451	S. G. Varnado, 6200
1	MS0829	J. M. Sjulin, 12323
1	MS0829	E. V. Thomas, 12323
1	MS0806	T. D. Tarman, 9336
1	MS0806	L. G. Pierson, 9336
1	MS0806	E. L. Witzke, 9336
1	MS0813	R. A. Suppona, 9327
1	MS0785	R. L. Hutchinson, 6516
1	MS0455	L. R. Phillips, 6517
1	MS0455	S. Y. Goldsmith, 6517
1	MS0455	R. S. Tamashiro, 6517
1	MS0784	M. J. Skroch, 6512
1	MS0801	M. R. Sjulin, 9330
1	MS 0188	LDRD Program Office, 1030 (attn: Donna Chavez)
2	MS 0899	Technical Library, 9616
1	MS 0612	Review & Approval Desk, 9612 for DOE/OSTI
1	MS 9018	Central Technical Files, 8945-1