THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

ANOMALY-BASED SECURITY PROTOCOL ATTACK DETECTION

By

TYSEN GLEN LECKIE

A thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Fall Semester, 2002

The members of the Committee approve the thesis of Tysen Glen Leckie defended on November 14, 2002.

_____
Alec Yasinsac
Professor Directing Thesis

_____
Lois Wright Hawkes
Committee Member

_____
Daniel Schwartz
Committee Member

I dedicate this work to my loving parents, Marlene and Glen. Without their support and driving force, this work might not have been completed.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Anomaly-based intrusion detection systems have traditionally been plagued with a high false alarm rate coupled with a high overhead in the determination of true intrusions. The viability of such systems is directly correlated to the creation of the normal profiles that are then used to compare all other observations from.

This paper presents a novel method to create a profile that accurately reflects user behavior by storing user log activity in a static metadata representation. A dynamically generated profile is computed from the stored log activity.

Standard deviation is used to measure the uniform characteristics of session activity over multiple time granularities. Chi-square is used to measure the normality of data in relation to historical records. This environment is focused on detecting intrusions upon security protocols without the added overhead of analyzing the encrypted payload.

INTRODUCTION

The first step in defending against an information based attack is determining that the attack has either already occurred or is currently in progress. Intrusion Detection Systems (IDS) are used in the detection of unauthorized or malicious behavior of computer activity or network traffic, preferably while such activity is ongoing. A central facet of intrusion detection is to preserve the confidentiality, integrity and availability of an organization's technological resources. Confidentiality is assurance that data is shared only among authorized entities. Integrity is assurance that the information is authentic and complete. Availability is assurance that systems responsible for storing, sending and processing data are accessible when needed. Compromising one or more of these attributes, renders an organization open to attack and the potential failure of critical systems.

Organizations increasingly exploit information technology to support core competencies focused in national security, military, utilities, transportation, financials and other vital areas. In October 1997, the Presidential Commission on Critical Infrastructure suggested increasing spending to a $1 billion level over the next seven years on various important information security topics. The focus included the areas of vulnerability assessment, intrusion detection and information assurance technologies [20].

The computer security problem is real. Ten major governmental agencies had been compromised in previous years, with a success rate of 64% of the attacks reported. The success rate has been doubling every year. Based on previous research, the General Accounting Office (GAO) estimates that only 1% to 4% of these attacks are detected and approximately 1% are reported [4].

In this paper we present an approach that employs an anomaly (behavior-based) method called the Behavioral Secure Enclave Attack Detection System (BSEADS). We examine the problem of anomaly detection as one of learning to describe behaviors of security protocol activity in terms of temporal aspects of user related session data. The purpose of this research is to detect attacks on security protocols by employing a behavior-based method of detection based on a chi-square test statistic. We further utilize standard deviation to measure the uniform characteristics of session data occurring over multiple time granularities. This gives us a richer and more accurate temporal view of our data sets.

In previous research there has been a noticeable lack of discussion in the area of IDS within encrypted environments. Future networks must be secure. These utilize a variety of mechanisms in conjunction with IDS such as firewalls, access controls, secure shells, auditing and encryption techniques. The use of encryption is especially important in communication over insecure mediums such as the Internet. If the physical line is not secure then the transmitted data itself must be. An issue of importance arises in the deployment of IDS in such environments where the analysis of the encrypted payload is a slow or

impossible task based on key space, key availability, processing power, amount of data and system priorities.

Our approach is novel because it is not focused upon the encrypted payload of the security protocol sessions. Instead, we develop a method that uses metadata to describe session activity without the resource-consuming task of analyzing the encrypted payload, a potentially impossible undertaking. Metadata is essentially an underlying definition of data about data. We use it to aid in the identification, description and location of data. This Knowledge Engineering approach extracts session-specific metadata that uniquely characterizes security protocol activity in such a way as to facilitate attack detection. This unique approach is vital in any secure environment where encryption algorithms are used to encode payload, effectively negating unproblematic deciphering.

The system we describe is based on a simulated networked environment within an encompassing architecture called the Secure Enclave Attack Detection System (SEADS) created in the Security and Assurance in Information Technology (SAIT) Laboratory at Florida State University (FSU). The mission of SAIT Laboratory is to support the demands of research, education, and outreach in information assurance and security. FSU is a Center of Academic Excellence in Information Security Education designated by the National Security Agency (NSA).

This paper is divided into six parts. First, background information is presented on previous applications with a discussion on methods of anomaly

detection. Next, taxonomy of events is presented. This is followed by a discussion of the time granularities we use to aid in intrusion detection. We then present an overview of the BSEADS environment and go into further detail of its essential components. This is followed by each of the Behavioral Analyzers that represent our essential detection constructs. Finally, we detail our methods of detection and present an experiment to test our concepts.

CHAPTER 1

SEADS CONCEPT

Background

The SEADS concept was initially proposed in order to allow intrusion detection in encrypted traffic [25]. SEADS was subsequently developed within the SAIT environment in order to replicate security protocol activity. By introducing attack behavior within the system, different intrusion detection techniques could be tested and benchmarked in varied configurations.

We have two flavors employed within the SEADS environment, KSEADS and BSEADS. The Knowledge Secure Enclave Attack Detection System (KSEADS) is a knowledge-based (misuse) method used in the detection of security protocol attacks [6][17][18][25]. Security protocol traffic is compared to stored signatures of known attacks resulting in a decision on whether the comparison results in a positive match for malicious behavior.

IDS are generally categorized in two main areas: misuse detection and anomaly detection. Misuse methods aim to detect well-known attacks by having *a priori* knowledge of those attacks. New attacks must be learned by the system before they can be detected. Anomaly techniques assume that all intrusions display anomalous characteristics, therefore rendering themselves detectable. The assumption is that systems are insecure by definition and that malicious

activities can be detected by analyzing system behavior. Given a “normal activity profile” intrusions produce behavior that varies from an established historical profile by statistically relevant amounts [9].

Misuse based methods generally have a high degree of accuracy. This is the result of known attacks on the system being detected successfully, since their attack patterns are known beforehand. The resulting benefit is a low incidence of false positives. A problem arises in the limited degree of completeness. The intrusions that are detected have a high probability of being actual intrusions but new or previously unknown attacks remain undetected by the IDS. This translates into a high false negative rate.

The predicament resides with the frequent updating of the known attack database. New attack signatures must be added before those attacks can be detected. This involves a high degree of overhead and does not guarantee that new attacks will be detected in a timely manner. The continuous research to analyze new attacks and find their signatures may ultimately fail because a slight change in the attack sequence from polymorphic attacks may result in a bypass of detection. Some traditional misuse based methods include a rule-base expert system [13], a state transition analysis tool for Unix [8] and a state transition network-based application [22].

BSEADS is a behavior-based method under the SEADS architecture that is used in the detection of security protocol attacks. The joint KSEADS and BSEADS initiatives within the SEADS umbrella will give us the benefit of both misuse and anomaly focused initiatives, resulting in an enhancement of our

intrusion detection capabilities. This paper details BSEADS and how it can be used to detect previously unknown attacks within an encrypted environment.

Behavior-Based

This approach uses models of normal data. Attacks are detected by observing deviations in the normal profile caused by observed data. It can detect all attacks without relying on *a priori* knowledge of those attacks. This is because all intrusions should display abnormal characteristics that render them detectable. Abnormal behavior does not necessarily indicate an intrusion, but a deviation from the historical profile. Once the historical long-term profile is built, comparing a subject's short-term observed behavior to this profile facilitates detection of intrusions.

The focus on anomaly detection has been increasing ever sense the seminal first comprehensive model of intrusion detection systems was proposed by Denning [3]. In this frequently cited document, she discusses "anomaly records" which are one of the six basic components of a generic intrusion detection system. Many types of approaches have been attempted in the area of anomaly detection. Neural networks were used to detect abnormalities in the behavior of software programs [5], data mining of TCPdump audit data was implemented [2] and also statistical profiling methods that search for anomalies and maintain a database of profiles [1], [10], [16], [19].

Anomaly detection systems generally have a high degree of completeness. This results in the benefits of a low false negative rate. This is the consequence of detecting known and unknown attacks on the system, both of

which deviate from the model of normal behavior in a noticeable and detectable way. The high degree of completeness is offset by a low degree of accuracy. Because this comparison is not performed by direct matching (signature) but relies upon more arbitrary statistical methods, there is a level of error associated with correctly labeling an attack thereby increasing the false positive rate. This error is the main limiting factor of anomaly applications.

The success of a positive detection relies on the model of normal behavior coupled with the thresholds and parameters set to signify an intrusion. Issues that arise are observed activity that is an intrusion but displays no abnormal behavior and a historical profile that is not entirely normal.

The viability of such systems is directly correlated to the creation of the normal profile database. Attacks may not be detected if they fit the established profile of the user. In this case we use a novel approach that dynamically creates statistics based on a User Activity Log. This should increase our detection abilities of security protocol attacks because we directly model normal behavior from the past protocol session metadata of each individual user. Another drawback is that a malicious user might train the anomaly detection algorithm to learn his or her behavior as normal over time. To counter this scenario we analyze multiple and larger time granularities for intrusive behavior.

Anomaly methods play an essential role in any IDS. A system based solely on the misuse model is not a viable solution alone because of damage that can occur from the time a new attack is detected to the time the attack sequence has been recorded and logged. Also, to have predefined instances of hostile

activity is a somewhat impossible task. An issue of coverage arises [11]. The space of possible malicious behaviors is potentially infinite. Therefore, it is difficult or impossible to have complete coverage of this entire space.

Security Protocols

A security protocol is a communications protocol that encrypts and decrypts a message for the purposes of online transmission. Examples of popular security protocols are SSL, SSH, HTTPS, S/MIME and IPSec. These are used for purposes of authentication and to provide a secure communications link using a method of encryption on the contents of the message. This data encryption is especially important considering that all communication is performed over an insecure medium. Therefore, the packet itself should be encrypted if the physical link is not. The resulting benefit is a private, secure and reliable method of data exchange.

Many protocols exist, and not all of them are free of errors. Flaws in protocol design or implementation sometimes are not discovered until years after the protocols are initially adopted. Security protocols are prone to attacks. The encrypted payload can be compromised if the key used is known or discovered by a malicious intruder. Replay techniques, masquerading, parallel session, oracle, man-in-the-middle and guessing attacks are a few of the ways that security protocols can be defeated [7], [15], [21], [23], [24].

Taxonomy of Events

The definition, creation and subsequent observation of contrasting behaviors are pivotal to anomaly systems. Intrusions that display normal

behavior are undetectable. For example, an IDS is programmed with the metric of “LOGIN NUMBER”. It contains the normal count of this metric per user per time of day. If a certain individual normally logs in five times during the hours of noon to one, and the IDS records on a particular day this amount equals ten times, then it will raise an alarm.

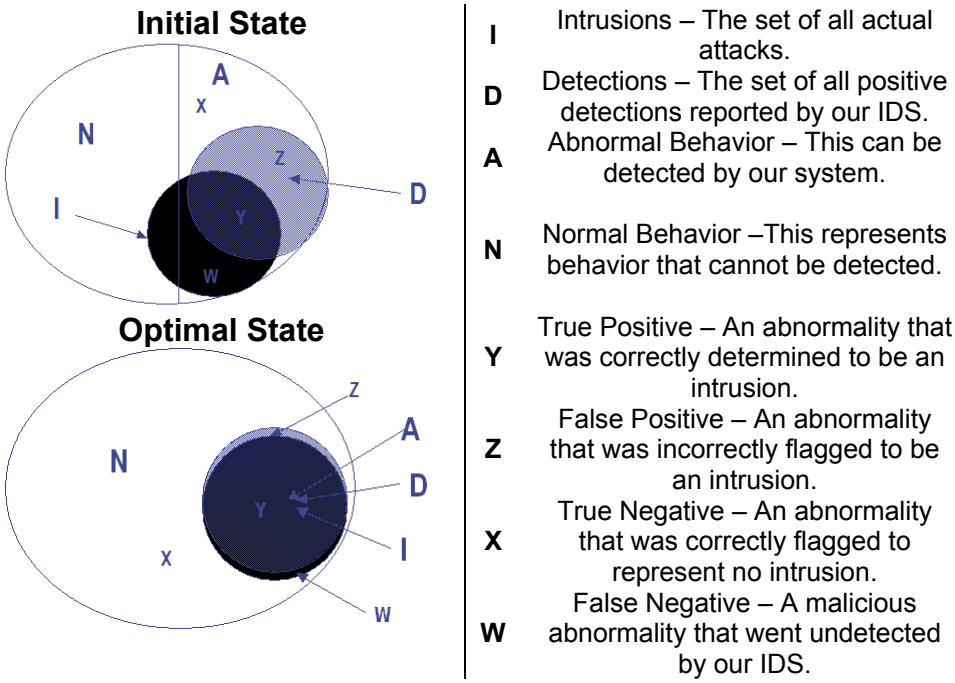

Figure 1

This depends on the threshold or standard deviation from the mean the system uses to signal such an intrusion. However, if the same individual logs in five times during the same time period, but this time maliciously corrupts critical system files then the IDS would not be able to detect this attack. This is because it is not an abnormal deviation from the user’s normal behavior for the time frame of noon to one for that individual metric. Different metrics would be needed, rather than “LOGIN NUMBER” to detect this intrusion.

The Venn diagrams in Figure 1 are a representation of all behavior. Anomaly systems only detect (D) behavior that displays abnormal (A) characteristics. Normal behavior (N) is undetected by our system even if this behavior is an actual intrusion (I).

A non-optimized intrusion detection system that has inadequate metrics, detection thresholds, or an irrelevant representation of normal behavior is characterized by the initial state in Figure 1. This entails an unacceptably high degree of false positives (Z) and false negatives (W) in conjunction with an intolerably low rate of true positives (Y) and true negatives (X) the consequence of which is intrusions that are allowed to execute undetected. False positives are not as bad as false negatives. A false positive is basically an administrative overhead. However, a false negative is no alert at all. Thus, limiting false negatives is a high priority. If there are many false positives or false negatives then one needs to reconsider how anomalous behavior is determined.

When the system corrects the problems listed in the initial state it reaches an optimal state as depicted in Figure 1. This is distinguished by a noticeable improvement in all the rates of detection. In the optimal state, most of the actual intrusions display abnormal characteristics. A significant percentage of abnormal behavior is correctly labeled an intrusion and a noteworthy number of intrusions are detected.

Time Granularities

In this section we describe how time granularities and temporal sequence learning assist us in refining our definition of behavior over multiple aspects of

daily activity. Time is an essential descriptive identifier of any event. As a result of the dynamic nature of the networked environment, activity is time varying. Usage will differ with hour of day, day of week, season, holiday, non-work days, etc. For an IDS to truly be useful it must consider the time varying characteristics of the environment and adapt to changes as users utilize a myriad network of changing components.

Recording when events occur and noting their relative duration can assist intrusion detection by refining to a more accurate and granular aspect the definition of behavior. User activity is different per time period. The total activities of an intruder last from a few seconds to multiple hours. Patterns may only appear in analysis gathered from a number of different metrics, possibly spread over multiple regions of time. Our objective is to understand these nuances and to more richly describe the notion of behavior in order to span multiple time granularities. As a consequence of this, we are able to detect attacks that display abnormal behavior in an individual time category, multiple time categories, or an entire day's worth of activity.

The time characteristics that we are concerned with are the ones that might be encountered during an average workday. We want to be able to describe behavior that accurately portrays the time that session activity happens by mapping it to pre-determined categories of the day. It is often difficult to classify a single user session as normal or abnormal because of the erratic behavior of most people. By grouping session activity within time categories we are able to more richly and accurately portray their behavior not for just individual

sessions, but for entire groupings of sessions. Because attack behavior takes a nonfinite time to develop it is essential that we analyze groupings of sessions for malicious behavior. In a networked environment, metrics are sometimes interdependent. Only a few metric values contribute significantly in changing the overall statistics. Abnormal activities do not just affect the magnitude of the associated metrics but also their relationship with other metrics. It is difficult to detect abnormal behavioral patterns by analyzing individual metrics only. Therefore, multivariate methods are a superior approach.

Time Categories

Category	Coverage of Day
Early Morning	0 – 6 hours
Morning	6 – 11 hours
Afternoon	11 – 16 hours
Evening	16 – 19 hours
Night	19 – 24 hours
Entire Day	0 – 24 hours

Figure 2

Figure 2 contains the categories of time that partition our system. Each category relates to a predefined portion of the day represented by the associated time coverage for that category. By analyzing these groupings of session activity at the conclusion of their time periods, we are able to detect a change in behavioral state after a manageable and controllable duration. This aids in the detection of intrusions by refining our search space, which is imperative because the early detection of problems is important in order to minimize damage.

The categories are defined by dividing a 24-hour day into the notion of early morning, morning, afternoon, evening, night and entire day. By developing

an expanding degree of granularity, we are able to further refine the specificity of these segments to detect variations of workload occurring in different periods.

We define a generation schedule that essentially is a predetermined time during any given day that the Behavioral Intrusion Detection Engine (B-IDE) performs a check on system behavior. After each of the listed time categories in Figure 2 expire, the system performs a behavioral analysis to determine the presence of intrusive activity for that recently completed time category. Once the night category has been exceeded the system performs an overall entire day check. By checking the system during multiple time periods, we are more likely able to catch behavior that shows recent signs of deviations in either one or many time categories. By checking the behavior at the conclusion of the day, thereby analyzing the total accrued session activity, we notice any flux in behavior that took a protracted time to develop.

CHAPTER 2

BSEADS SYSTEM COMPOSITION

Architecture

The system is an extension of the SEADS environment. This is displayed in Figure 3. SEADS is composed of a group of principals (computer processes) operating within a secure environment. Security protocol sessions are transmitted between communicating principals. These principals send session specific data to a Monitor via a secure link. The Monitor is responsible for gathering all session data. This repository is what is used as the basis for the training of BSEADS.

SEADS Diagram

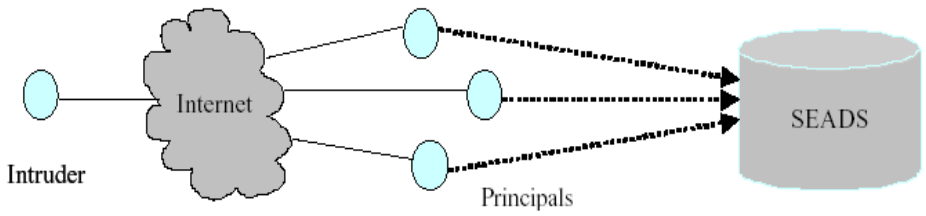

Figure 3

BSEADS is essentially an extension of the SEADS environment. The User Activity Logs are trained using attack-free data from a Monitor Activity File that is based off the representation and structure of the Monitor repository within the SEADS environment. Attack behavior is not inserted into the session stream

during this period. The User Activity Logs for each user are populated with the training data, the majority of which occurs during the normal working hours of 8:00 a.m. to 5:00 p.m. The output of this process is a profile that we call normal, which depicts behavior during periods when there are no intrusions. This serves as a set to which observed data will later be compared during the detection phase, which follows the training phase.

BSEADS Diagram

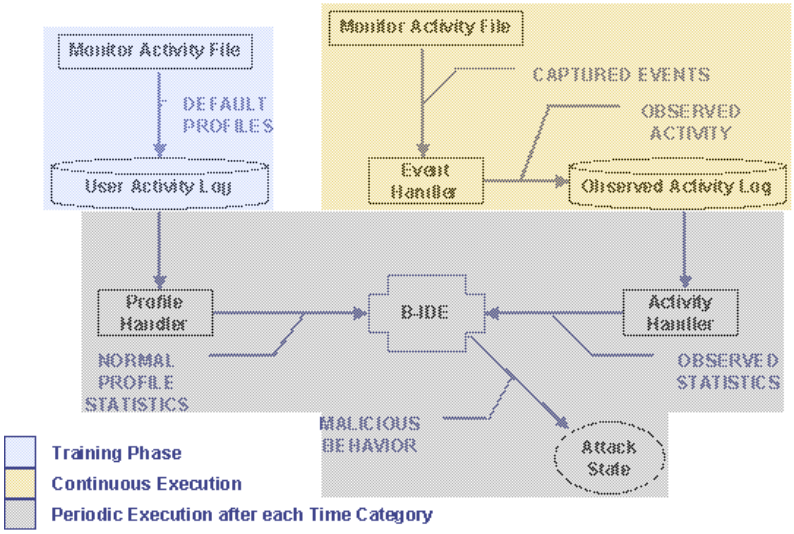

Figure 4

An Event Handler (EH) is used to dynamically capture ongoing protocol session activity from the Monitor. This is a continuous process. It parses the activity to determine which principals are involved in the communication and inserts session specific metadata within the corresponding Observed Activity Log for each principal. The diagram of Figure 4 represents a top-level view of the main objects, relationships and interdependencies between components.

The Activity Handler (AH) maintains each principal's Observed Activity Log and executes method calls upon the log, as requested by the B-IDE. The return structure of these method calls is each user's Observed Statistics.

A Profile Handler (PH) maintains each principal's User Activity Log. This is the permanent historical repository that was populated during the initial training phase. The PH has the responsibility of executing method calls upon the log, as requested by the B-IDE. The return structure of these method calls is each user's Normal Profile Statistics.

The responsibility of the B-IDE is to retrieve Observed Statistics and Normal Profile Statistics from the AH and PH respectively, for each Behavioral Analyzer (BA). This occurs periodically at the conclusion of each time category (early morning, morning, afternoon, evening, night, entire day).

The B-IDE measures and checks each user's BA at the finish of every time category using both standard deviations to determine any non-uniform characteristics and chi-square to analyze any abnormal behavior. If the result of an analysis signifies an intrusion then the system transitions into an Attack State.

CHAPTER 3

USER ACTIVITY

Definition

User activity is represented as a particular user's security protocol sessions. It takes on two main forms: observed activity and normal activity. Observed activity is activity that has not yet passed the inspection of the B-IDE. It is the activity that is used as the comparison set to normal activity. Normal activity is the historical non-attack representation of past behavior that was initially trained into the system.

Within the user log files, the temporal metadata information stored in the fields include start time and end time, which are expressed using the format of year, month, day, hour, minute and second.

Our behavioral profile is initially trained with a representation of normal behavior. After this initial training any subsequent updates are performed on a periodic basis. This is a benefit, because it prevents users from slowly broadening their profile by phasing in maliciously abnormal activities that would then be considered normal once included in the user's profile description.

Normal User Profile

Future events can be predicted by realizing past behavior. A normal profile is a representation of historical behavior. It contains no attack characteristics. If it did contain intrusions then any future occurrences of those same attack types might go unnoticed. This is an unacceptable factor.

Traditionally, in anomaly detection, profiles were constructed using different characteristics, such as consumed resources, command count, command sequences, typing rate, etc. The main focus of our approach is to apply methods to the extensively gathered metadata in order to compute statistics that accurately capture the behavior of normal activities in such a way as to facilitate attack detection. This approach significantly reduces the need to manually analyze and encode behavior patterns and is also effective because it is computed using a large amount of session metadata, thereby normalizing the results.

Normal user profiles generally use two main methods to train models of behavior: “synthetic” and “real” [13]. Synthetic techniques generate behavior by executing the application in as many normal modes as possible. This means that security protocol sessions are executed in varied normal configurations and settings. It is within this synthetic environment that the definition of normal is developed. Observing the security protocol sessions in a live user environment signifies the real method.

The synthetic method works well when a replication of the result is needed and it incurs lower overhead in manpower and resources. We can also be

confident in the results, because we are certain that no intrusion activity contaminates the profile. A real method contains problems in the collection and evaluation of activity. This occurs because of the uncertainty that anomalistic behavior is not present. It has the benefits of attaining a very diverse and actual representation of a user's security protocol behavior.

The User Activity Log represents a user's entire session activity that was created during the training phase using an approach based on a realistic representation of session activity for each user. This metadata representation is then used as the basis for our Normal Profile Statistics that are calculated from this log. Our definition of a normal profile is essentially a combination of all relevant statistics for the time granularities.

The B-IDE compares the statistical values that are returned with the observed statistics dynamically. Multiple methods of detection are used in the final classification of attack behavior.

User Activity Log

A User Activity Log is a static representation of metadata based on user session activity. It is the main repository of stored historical data created during the training phase. During creation we ensure that no attack characteristics are included within these logs. Each user in the system has a corresponding log that contains all activity the user executed during the running of the training phase.

The User Activity Log for a particular user contains all trained activity for that principal. The format of the repository includes an extensive set of fields that uniquely describe each of the sessions. Start time (ST) and end time (ET) are

included to understand when the session occurred and the duration. The protocol used (PR) signifies the protocol corresponding to the session. There are a total of four principals (P1, P2, P3, P4) that can be involved in any one session.

Principal one (P1) is the initiator of the session. Principal two (P2) is the recipient of the initiators first message. Principals three and four (P3, P4) are optional participants of the session that are realized sometime after the first message is exchanged. These are optional participants because some sessions would only require a total of two principals involved. Session completion (C) is included to illustrate whether the session failed or succeeded. The encryption key strength (KS) describes the level of encryption used. The unique session identifier (SID) is a unique number per session stored.

The reason that all of these fields are included is to support the job of each of the Behavioral Analyzers (BA) which are our main detection constructs. Each BA signifies a particular behavioral pattern that we are monitoring within the system. These are the essential constructs that we use in determining whether malicious behavior is present. The BA depends on all of the fields mentioned above.

Time Degradation

When the method calls are executed upon the User Activity Log, a structure of Normal Profile Statistics is retrieved for every BA. These are used in the comparison to Observed Statistics. One of the main issues that the design of our system had to solve is how to correctly represent past behavior. We developed an algorithm that is executed upon past behavior to ensure that older

activity carries less weight than more recent activity. This is an essential factor because a user's more recent past behavior is a better indication or prediction of current behavior than older past behavior.

In a realistic setting the difficulty noticed is that human behaviors are dynamic. This means that the definition of normal behavior is likely to change over time, thereby limiting the lifetime of a static user profile. Thus, we developed a method of degradation that assigned an increasing weighted value as session activity increased in age. We are more concerned with recent activity because it is an enhanced and more current representation of user behavior. The fresher it is, the more weight it should carry in any subsequent calculations.

Time Degradation Calculator

User Activity Log Range	**Degradation Factor**		
1 Week Old	40%	20%	10%
2 Weeks Old	60%	30%	15%
3 Weeks Old	80%	40%	20%
4 Weeks Old	100%	50%	25%
Weekly Standard Deviation	**0 = Low**	**> 10 = Moderate**	**> 20 = High**

Figure 5

We use a Time Degradation Calculator as represented in Figure 5. The execution of this algorithm occurs when the methods are called upon the User Activity Log and is reflected in the returned Normal Profile Statistics. The weekly standard deviation is initially determined based on the four total weekly-accumulated BA value counts. The values that we are analyzing are within each of the normal profile BA. For example, when the Parallel Session Behavioral Analyzer is called using the Normal Profile Statistics, standard deviation is calculated upon the four weeks of prior activity by determining the number of

parallel sessions that are occurring and measuring their level of volatility over this multiple week time span. Depending on this value, a different degradation factor is used. This is a continuous refinement of our definition of normal behavior incorporating a premium on more recent activity. If the standard deviation between the entire four-week periods is over 20, then we can be relatively certain that there is high volatility involved.

Therefore, this activity carries more significance because of its non-uniform characteristics. This is degraded less than activity that carries a moderate or low volatility level. In all cases, as the activity increases with age the degradation factor also increases. Individual activity entries can be manually removed from the input to the calculator in order to avoid skewing final results. Examples would be one-time events such as holidays, vacation time or sick leave.

These are removed because we want an accurate reflection of past behavior that is a close mirror or match to current behavior in the circumstances in which it is present. For example, if in a prior week only three working days contained session activity while the other two days occurred during a holiday, then we would not want that occurrence to negatively skew in comparison to a current week in which all five days have session activity.

We want to be able to manually modify the input to the calculator so that we use the last known good week of a full five days of session activity. This would result in a more accurate comparison to present activity.

Normal Profile Statistics

These are dynamically generated from method calls upon the User Activity Log. The Time Degradation Calculator is used in the result in order to increase the weight of more current data while decreasing the weight of older data. Our algorithms sift through the data and retrieve metric information in order to assemble each Behavioral Analyzer (BA). The result is a structure that satisfies our requirement of a normal activity profile, as the outcome contains no attack data. Therefore, this helps define our concept of normal activity for comparison purposes.

The count of each BA's metric value is returned in the result data set. The Normal Profile Statistics are used to construct each BA. This structure is returned at the conclusion of every time category in order to display the accumulated statistics for that specific time period.

Observed Profile

The observed profile is a representation within the system of ongoing activity for each user. This data might contain malicious activity; therefore it must be analyzed. It is a temporary data repository that will be used as the comparison set to normal activity. When the Event Handler initially retrieves session activity from the Monitor, it is immediately inserted into the observed profile for each corresponding communicating principal or user. This is accomplished by placing it within the Observed Activity Log for that particular user.

Periodically, at the conclusion of each time category, the B-IDE requests each user's Observed Statistics from the Activity Handler that are created from

method calls upon the Observed Activity Log. These will be used to run a test of uniformity using standard deviation in order to determine any volatile or concentrated session behavior and also as the comparison set to the Normal Profile Statistics in order to detect any abnormal behavior.

Observed Activity Log

A user's Observed Activity Log is a static representation of recent protocol sessions for each principal. It is a listing, much like the format of the User Activity Log, of session activity in a metadata format.

Once activity is initially observed for a user it is placed within their corresponding Observed Activity Log. The log contains all relevant session information that will be used to construct each Behavioral Analyzer. The structure of the log is similar to that of the User Activity Log used to construct the Normal Profile Statistics. The start time (ST), end time (ET), protocol used (PR), principals involved (P1, P2, P3, P4), Boolean session completion result (C), key strength (KS) and unique session identifier (SID) are all represented.

Observed Statistics

We classify Observed Statistics as a dynamically generated execution upon the Observed Activity Log instigated by the Activity Handler using method calls in order to create the Behavioral Analyzers (BA). The resulting statistics represent behavior that is important to each BA used to detect attacks. Each BA metric count is returned within the resulting data structure. The B-IDE constructs the Observed Statistics at the conclusion of each time category.

Each BA metric and corresponding value for the eclipsed time period is returned. This is used to aid in the detection of attacks for each time period. Upon the return of this structure we have an accurate portrayal of the observed activity behavior during the respective slice of time that we are analyzing.

CHAPTER 4

BEHAVIORAL ANALYZERS

Definition

An intrusion is an attack on system security that derives from an intelligent threat. It is a deliberate attempt to evade and preempt detection while violating the security policy of an information system.

The fundamental concept of our system is the monitoring of behavioral patterns to detect the attacks instigated by potential malicious intruders. There are some assumptions that we must define for the role of the intruder. An intruder can alter, record, intercept and communicate any data traveling on the network. He has the basic capacity of encryption and decryption, as do all legitimate users in the system.

Quantifying security risks in the system is a nontrivial task. We have partitioned the monitoring activity into modules called Behavioral Analyzers (BA). The BAs that we are most concerned with are parallel sessions, failed sessions, replay sessions and weak encryption sessions.

These BAs represent many of the forms that characterize attacks on security protocols. By monitoring activity in these areas we increase our chances of detecting an attack by focusing on deviating behavior that constitutes a high degree of confidence in representing an actual intrusion. Because a large

number of attacks result in a behavioral fluctuation in one or more of these BAs, the monitoring of such behavior is prudent. This will result in a more accurate and correct diagnoses of abnormalities by the B-IDE.

An open issue in anomaly detection is the selection of measures to monitor. Exact measures that accurately predict intrusive behavior are not known in any context. It seems that a combination of static and dynamic determination of the set of measures is beneficial.

In order to address this issue we developed the BA to be composed of dynamically generated metrics from a static representation. Metrics are observations of a user's actions that can be grouped to create a profile of user behavior. Each BA uses a unique set of metrics that characterize the purpose and functionality of that particular BA. The metrics used are counters that measure the number of events that have occurred. The BA is invoked at the conclusion of each time period by the B-IDE. Our BAs are created from the Normal Profile Statistics and the Observed Statistics.

For example, in order to detect a parallel session attack, we invoke the parallel session BA from the Normal Profile Statistics. This BA satisfies our requirement for a trained normal data set. We also invoke a parallel session BA from the Observed Statistics for the corresponding time period. This satisfies the test requirement for data that will be used as the comparison set to normal data.

Each of the other BAs are constructed in a similar way. Therefore, it is appropriate to assume that a BA can either be a normal BA or an observed BA. These are then compared together for the normality check.

B-IDE Diagram

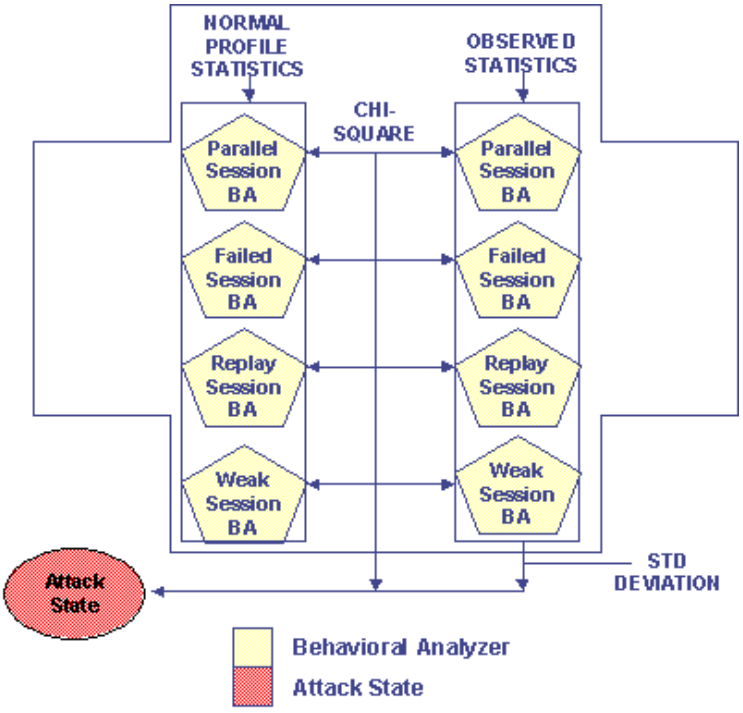

Figure 6

New sets of BAs are created within the B-IDE as displayed in Figure 6 at the conclusion of each time period. BAs are created from the Normal Profile Statistics and another set is created from the Observed Statistics. These two sets are then compared using chi-square in order to determine any abnormal deviations. Also, the BA created from the Observed Statistics is analyzed to determine any non-uniform characteristics using standard deviation. If the result of any of these checks display abnormal behavior then the system transitions into an Attack State.

Parallel Session

Whenever two or more protocols are executed concurrently, a parallel session results. This in itself is not indicative of an attack as parallel sessions can occur normally. An attack can happen though that takes advantage of parallel

sessions. This occurs when messages from one session are used to form messages in another. The purpose of a parallel session attack is that an attacker is able to glean information in one session that is useful in an attack in another session.

The protocol created by Woo and Lam [24] contains a well-known flaw that allows a malicious user to perform a parallel session attack. This flaw allows the intruder (Mallory) to convince a legitimate user (Alice) to accept the same key that was used in the initial session to be used during the parallel session.

One possibility of the attack is that Alice believes that she is in a secure session with another legitimate user (Bob), but Bob is not involved and is not aware that there is an ongoing session. Mallory could have used Alice as an oracle in order to masquerade as Bob. For example, Alice and Bob are involved in a protocol session. Mallory is involved as a man-in-the-middle and intercepts all transmissions to receiver Bob. Mallory than initiates a parallel session to receiver Alice, masquerading as Bob.

A masquerade is when one principal illegitimately poses as another in order to gain access to it or to gain greater privileges than they are authorized for. Masquerading is an active form of wiretapping in which the intruder intercepts and selectively modifies data in order to masquerade as one or more of the entities involved in a communication. The attacker sniffs network packets, potentially modifies them and inserts them back into the network stream. An oracle occurs when the attacker convinces other entities (oracles) to perform an action on the attackers behalf.

The Parallel Session Behavioral Analyzer returns the number of parallel sessions that are occurring. The smallest granularity that is represented is the per hour accumulation.

Failed Session

A failed session is the result of a failed initiation of a session or a failure while the session is ongoing. Intruders do not usually succeed during their first try at breaking into a system or maliciously interfering in a protocol session. Sometimes multiple session failures result as a byproduct of an intruder's blatant intervention.

A brute force attack is represented as a cryptanalysis technique, or similar kind of attack involving an exhaustive process that tries all possibilities, one-by-one. It is a trial and error technique used by intruders to decode encrypted data such as passwords or Data Encryption Standard (DES) keys, through comprehensive effort (brute force) rather than employing intellectual strategies. Brute force is considered to be an infallible, although time-consuming, approach. The Secure Shell (SSH) protocol version 1 contains a flaw that allows a brute force attack to succeed [7].

One type of brute force attack is called the dictionary attack. An intruder that implements a brute-force technique of successively trying all the words in some large, exhaustive list characterizes this form. An example of this would be a dictionary of known words used in order to discover a password. Another scenario involves an attack on encryption by encrypting some known plaintext

phrase with all possible keys so that the key for any given encrypted message containing that phrase may be obtained by lookup of the value.

Just as a thief might crack a safe by trying all possible combinations, a brute force attempt proceeds through all possible combinations of legal values sequentially. The reality is that success on the first try is an unlikely scenario; therefore we employ a Failed Session Behavioral Analyzer to monitor this form of activity. This analyzer returns the number of failed sessions that are occurring. The smallest granularity that is represented is the per hour accumulation.

Weak Session

Encryption standards should be in place to use the highest form of encryption supported between two communicating principals. Sessions that are encoded using a weak form of encryption are prone to many forms of attacks. An increase in the success of a brute force attack occurs because of a limited key space. This aids an intruder by limiting the amount of keys needed to try.

Masquerade attacks benefit from weak authentication, because it is much easier for an intruder to gain access to system resources. Once the attacker gains access to critical data they may be able to modify and delete software or make changes to network components and configurations.

Another type of attack that is characterized by a weak form of encryption is called a cipher suite rollback attack. This occurs when an active attacker edits the cleartext list of cipher suite preferences (encryption strengths) in the initial handshake messages.

The purpose is to maliciously force two parties to use a weaker form of encryption. This weaker form of encryption is lower than what Alice and Bob are capable of using in their communications. The attacker could force a domestic user to use export-weakened encryption. A variation of the cipher suite rollback attack is performed by exploiting a flaw in the SSL 2.0 protocol [23].

The Weak Session Behavioral Analyzer returns the number of weak sessions that are occurring. The smallest granularity that is represented is the per hour accumulation of sessions.

Replay Session

A replay occurs when recorded sessions are replayed on the network at a later time. In a replay attack, a hacker uses a protocol analyzer to monitor and copy packets as they are transmitted via the network. Once the hacker has captured the required amount of packets, he then can filter them and extract the packets that contain things of need. Examples of this are digital signatures or various authentication codes. After the necessary packets have been extracted, they can be put back on the network (replayed), thereby giving the intruder the desired access.

For example, suppose an intruder collects session traffic that satisfies his needs. After an arbitrary amount of time, he replays the previously collected traffic into the message stream. This kind of attack could be used to replace signals that contain session data with signals showing ambient readings. It could also be used to do the exact opposite, create sessions when the sensors are receiving no such signals.

Variations of the Needham-Schroeder, Neuman-Stubblebine and Yahalom protocols are all susceptible to replay attacks [21]. A way to prevent replay attacks is to time-stamp each session or use some other form of unique identifier in order to notice if that session occurs at a later time, thereby violating the ordering of events.

The Replay Session Behavioral Analyzer returns the number of replay sessions that are occurring. The smallest granularity that is represented is the per hour accumulation.

CHAPTER 5

METHODS OF DETECTION

Definition

This section details the methods of detection employed within BSEADS. We are interested in analyzing two main components of our Behavioral Analyzers: measure of spread and measure of normalcy. Measure of spread is focused on observed activity and its uniform or non-uniform spread characteristics. This is based on our Observed Statistics. We use standard deviation for our uniformity test based on each of the Behavioral Analyzers.

Our measure of normalcy centers on comparing our Observed Statistics to our historical Normal Profile Statistics. The statistical method we employ is the chi-square test statistic.

Measure of Spread

Our definition of spread is in the form of session activity spread throughout an entire 24-hour period. Is this activity predominantly located or concentrated at certain times? If so, then it might not pass our test for spread uniformity. A major advance of our method over other IDS is that it considers the time locality and frequency of placement in the determination of attacks. By analyzing the

Observed Statistics we are able to detect volatile spread characteristics without having to rely on a normal comparison set.

Because attack behavior usually happens at certain times, and in high concentrations, our measure of spread will detect abnormally high occurrences of any of our Behavioral Analyzers that is not evenly distributed over individual time categories or throughout the entire day.

Lee and Stolfo developed algorithms for detecting abnormalities in system audit calls within categories of time but did not measure the spread characteristics [12]. We believe our method is a superior approach.

Standard Deviation

The standard deviation is a common measure of spread. It measures the uniform characteristics of a given data set. It is derived from the variance. The variance is the arithmetic mean of all the squared distances between the data values and their arithmetic mean. The standard deviation is defined as the square root of the variance.

This computation indicates how well the calculated measure of location describes the data. The result shows whether the data is accumulated close to the average or spread out over the whole scale.

Because the standard deviation weights the distances proportional to their absolute value, it places greater emphasis on larger rather than smaller distances. This could be the determining factor in capturing volatile session activity. The outliers that the standard deviation is sensitive to could be an

indication of attack behavior. A model of the calculation of variance and standard deviation is presented in Figure 7.

Variance and Standard Deviation

$$\text{Variance} = \frac{1}{N}\sum_{i=1}^{n}\left(X_i - \overline{X}_{ari}\right)^2$$

$$\text{Standard Deviation} = \sqrt{\text{Variance}}$$

Figure 7

We perform our standard deviation test on each of the Behavioral Analyzers based on the Observed Statistics. If the result of this calculation is larger than 1, we signal this as a non-uniform spread. A value of more than 1 standard deviation is a warning sign of a non-uniform distribution of session activity.

We determined the limit of 1 by measuring normal user protocol activity during each of our categories of time. Rarely do sessions deviate in standard deviation by more than 1. When we introduced attack behavior within the sessions, we noticed an increase in the concentrations of attack behavior. This increased standard deviation instigated by attack behavior displayed a result of greater than 1; therefore we came to our conclusion of limit.

The dispersion of values around the mean should indicate limited volatility. The larger the dispersion is, the higher the standard deviation and the higher the volatility. The smaller the dispersion, the lower the standard deviation and the lower the volatility. User behavior is sometimes erratic especially in a networked environment. But when an intruder is manipulating sessions or instigating

sessions of his or her own, the result is a higher concentration of activity. This is the determining factor of attack behavior.

Measure of Normalcy

The definition of normal within BSEADS is comprised of the Normal Profile Statistics that are dynamically generated from the trained historical User Activity Logs.

The observed activity that is represented by the dynamically generated Observed Statistics from the Observed Activity Log is compared to the Normal Profile Statistics. This comparison is used to help us classify observed behavior as either malicious or non-malicious. We use chi-square to satisfy our test of normal behavior. We measure the session normalcy after each time category has been eclipsed.

Chi-Square

Chi-Square is a non-parametric test of statistical significance. It allows us to attain a degree of confidence in accepting or rejecting a hypothesis. Our hypothesis is a mathematical measurement of whether or not the Behavioral Analyzers composed of the Observed Statistics and Normal Profile Statistics are dissimilar enough to warrant an attack classification. Figure 8 is a model of the chi-square test for normality performed within BSEADS. Figure 9 is a list of the symbols and their meanings.

Chi-square does not require the sample data to be more or less normally distributed (as parametric tests such as t-tests do). It relies on the assumption

that the variable is normally distributed in the population from which the sample is drawn.

Chi-Square Multivariate Model

$$X^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i}$$

Figure 8

Our chi-square method is similar to the one employed in [26]. It is essentially a distance measure of observed metric values from expected metric values. It distinguishes a correlation among two or more metrics used in attack detection and contains the benefit of a low computational cost. This is essential, because of the frequency that this test will be performed during program execution. We use numerous metric values in the actual chi-square formula. These metrics are accumulations of BA count values within multiple granularities of time.

Symbol Chart

X_i	Observed value of the i^{th} metric
E_i	Expected value of the i^{th} metric
n	Number of metrics
X^2	Chi-Square test statistic. X^2 is small if an observation is close to the expectation

Figure 9

We measure the size of the difference between the pair of observed and expected frequencies. Squaring the difference ensures a positive number, so that we end up with an absolute value of differences. Dividing the squared difference by the expected frequency essentially removes the expected frequency from the

equation. Therefore, the remaining measures of observed and expected difference are comparable across all cells.

Once we have computed the chi-square test statistic we must have a baseline or calculation to use as our comparison value. This is obtained by implementing an upper control limit. This calculation is essentially the threshold that the chi-square result is compared to. As noticed, the upper control limit is entirely computed from the historical normal profile.

Upper Control Limit

$$\text{Upper Control Limit} = \overline{X} + 2S$$

Figure 10

As shown in Figure 10, the Upper Control Limit is determined to be the Normal Profile Statistics sample mean plus 2x the Normal Profile Statistics standard deviation. The reason for this is because it should be the quantitative comparison measurement based solely on the Normal Profile Statistics, whereas the chi-square value takes into effect the observed behavior. The chi-square test for normalcy is computed for each Behavioral Analyzer at the conclusion of every time category. If the computed chi-square is larger than our upper control limit, we signal the presence of abnormal activity.

CHAPTER 6

EXPERIMENT

Overview

In this section we illustrate our theory and experiment with building our methods of approach in order to test its efficacy in detecting intrusions. The objective of our study is to test whether the distribution of protocol session activity follows the trained normal profile and whether the session spread is represented uniformly.

We employ the use of two Monitor Activity Files within this experiment. The first Monitor Activity File, to be used during the initial training phase, was constructed of attack-free protocol session activity. This represented a simulation of the Monitor within the SEADS environment. Most of the data inserted fell within the working hours of 8:00 a.m. to 5:00 p.m.

The major objective of the experiment was to validate the assumptions stated in previous sections; that we can create a normal and observed user representation and that we can dynamically generate the statistical structures required from the static storage. Finally, we are able to detect variations in session spread and also abnormal deviations at the conclusion of each time category with the last category analyzed being the entire day.

The first activity of BSEADS was to construct the User Activity Log by

dynamically training it with metadata from the Monitor Activity File. This signified the training phase. Four weeks worth of data was accumulated.

Our simulation was performed using the Windows 2000 platform. The application was coded in standard C++ code using the Standard Template Library (STL) and utilizing the Visual C++ development environment.

We began the simulation at hour 0 (12:00 a.m.) by the EH requesting Monitor activity represented within a second Monitor Activity File that contained activity that reflected attack characteristics. These attack characteristics represented each of our Behavioral Analyzers of parallel, failed, weak and replay sessions. This simulated the capturing of protocol sessions within a real-time environment.

As sessions were transmitted to the EH, events were parsed by principal and the metadata was placed within each communicating principal's Observed Activity Log. Once the conclusion of the early morning (EM) time category was realized, at the end of hour 6, the analysis began by checking each of the Behavioral Analyzers. We searched for volatility within the session spread of the EM category based upon the Observed Statistics. The normalcy within the EM was checked using both the Observed and Normal Profile Statistics employing chi-square.

Activity was checked throughout each time category (morning, afternoon, evening, night). Once the night category had been exceeded, we proceeded to check the entire days behavior (24-hours). We began by checking the spread; our first check for uniformity was based on each of the individual time categories.

Once this was determined we checked the uniformity based on each of the individual 24 hours.

Finally, we ran our test for normalcy based on the entire day's activity. We now present examples of each of our Behavioral Analyzers as they would be used in a real detection scenario.

Parallel Session Behavioral Analyzer

Observed Activity

ST	ET	PR	P1	P2	P3	P4	C	KS	SID
1200	1220	SSL	A	M	Null	Null	Y	128	342332
1200	1220	SSL	M	A	Null	Null	Y	128	948032

Figure 11

In Figure 11, Observed Activity is captured from the Monitor Activity File by the Event Handler (EH) representing the total accumulated activity for the afternoon time category. The first row signifies a session between the principals A and B. M acting as a man-in-the-middle, intercepted all messages intended for B. Therefore, A believed that it was involved in a session with B, but this was not the case. This was a parallel session between M and A.

Once captured, the Observed Activity was immediately placed within the Observed Activity Log for each principal by the EH. Once the afternoon category was realized, the B-IDE proceeded to check the entire afternoon accumulated behavior. It began by requesting the Normal Profile Statistics from the Profile Handler (PH) which when retrieved, were modified by the Time Degradation Calculator. This ensured the freshness of activity.

Figure 12 is a representation of the afternoon Parallel Session Behavior Analyzer Normal Profile Statistics retrieved for user M. Once this was

accomplished, the B-IDE requested the Observed Statistics from the Activity Handler (AH). Figure 13 details this. We measured the spread of the Observed Statistics using standard deviation. The result of this calculation was 1.304. This told us that the events did not occur uniformly over this time period.

Normal Profile Statistics

Metric	Value
Number of Parallel Sessions Hour 12	0
Number of Parallel Sessions Hour 13	0
Number of Parallel Sessions Hour 14	2
Number of Parallel Sessions Hour 15	0
Number of Parallel Sessions Hour 16	0
Total	**2**

Figure 12

Observed Statistics

Metric	Value
Number of Parallel Sessions Hour 12	1
Number of Parallel Sessions Hour 13	0
Number of Parallel Sessions Hour 14	3
Number of Parallel Sessions Hour 15	0
Number of Parallel Sessions Hour 16	0
Total	**4**

Figure 13

As mentioned earlier, the larger the dispersion was, the higher the volatility. Our calculated result signified that the observed values were slightly volatile because anything more than one standard deviation was a sign of concern.

The B-IDE next checked for any abnormal characteristics based on the Observed and Normal Profile Statistics. The result of this calculation was 2. We obtained the upper control limit, which was 1.50. This aided in our determination that there was abnormal behavior as a result of our chi-square value being larger than our upper control limit.

We therefore signaled an attack was occurring that showed a non-uniform spread in conjunction with abnormal behavior.

Failed Session Behavioral Analyzer

Observed Activity

ST	ET	PR	P1	P2	P3	P4	C	KS	SID
1832	1832	SSH	M	B	Null	Null	N	64	341342
1833	1833	SSH	M	B	Null	Null	N	64	123214
1834	1834	SSH	M	B	Null	Null	N	64	325421
1835	1835	SSH	M	B	Null	Null	N	64	462243

Figure 14

In Figure 14, Observed Activity is captured from the Monitor Activity File by the EH. This occurred during the evening time category. The rows represented multiple failed sessions between M and B. This was warrant for concern. Once this Observed Activity was captured, representing the entire activity for that time period, it was placed within each communicating parties Observed Activity Log. This was performed by the EH.

As the evening category ended, the B-IDE performed a system check on the behavior. The PH returned the Normal Profile Statistics and the AH returned the Observed Statistics. Once again, the Time Degradation Calculator as mentioned earlier modified the Normal Profile Statistics upon retrieval.

Normal Profile Statistics

Metric	Value
Number of Failed Sessions Hour 17	0
Number of Failed Sessions Hour 18	1
Number of Failed Sessions Hour 19	0
Total	**1**

Figure 15

Figure 15 is a representation of the evening Failed Session Behavior Analyzer Normal Profile Statistics retrieved for user M. Figure 16 shows the

Observed Statistics. We measured the spread of the Observed Statistics using our method of standard deviation resulting in the calculation of 2.31, a sign of alarm. By falling above our upper threshold, it signified that the metric values were not occurring uniformly over this time period. This volatile dispersion warranted an attack classification.

Observed Statistics

Metric	Value
Number of Failed Sessions Hour 17	0
Number of Failed Sessions Hour 18	4
Number of Failed Sessions Hour 19	0
Total	**4**

Figure 16

The B-IDE proceeded to analyze the behavioral characteristics based on the Observed and Normal Profile Statistics. The chi-square result of this calculation was 9. We obtained the upper control limit, which was 1.49. This lets us come to the conclusion that there was a definite observation of abnormal behavior since our chi-square result was significantly larger than our upper control limit. We therefore signaled an attack was occurring that showed a non-uniform spread in conjunction with abnormal behavior.

Weak Session Behavioral Analyzer

In Figure 17, Observed Activity is captured by the EH while in the morning time category. The rows represented sessions between A and B. Intruder M forced A and B to use weak encryption. The Observed Activity was placed within the Observed Activity Log of each principal by the EH. The B-IDE procedure was repeated to check the entire accumulated behavior.

Observed Activity

ST	ET	PR	P1	P2	P3	P4	C	KS	SID
0800	0810	SSL	A	B	M	Null	Y	32	345222
0920	0930	SSL	A	B	M	Null	Y	32	423453
0930	0935	SSL	A	B	M	Null	Y	32	634523
0940	0950	SSL	A	B	M	Null	Y	32	545223

Figure 17

Normal Profile Statistics

Metric	Value
Number of Sessions Using Weak Encryption Hour 7	0
Number of Sessions Using Weak Encryption Hour 8	1
Number of Sessions Using Weak Encryption Hour 9	0
Number of Sessions Using Weak Encryption Hour 10	1
Number of Sessions Using Weak Encryption Hour 11	0
Total	**2**

Figure 18

Observed Statistics

Metric	Value
Number of Sessions Using Weak Encryption Hour 7	0
Number of Sessions Using Weak Encryption Hour 8	1
Number of Sessions Using Weak Encryption Hour 9	3
Number of Sessions Using Weak Encryption Hour 10	0
Number of Sessions Using Weak Encryption Hour 11	0
Total	**4**

Figure 19

During data accumulation, the B-IDE requested the time degradation modified Normal Profile Statistics from the PH. Figure 18 is a representation of the morning Weak Session Behavior Analyzer Normal Profile Statistics for user M. Once this was retrieved, the B-IDE obtained the Observed Statistics from the AH. Figure 19 shows this recovery.

The B-IDE next measured the spread of the Observed Statistics using standard deviation. The result of this calculation was 1.30, a classification of statistics not occurring uniformly over this time period. As mentioned earlier, the larger this dispersion value was, the higher the volatility.

The B-IDE subsequently checked for any abnormal characteristics based on the Observed and Normal Profile Statistics. The result of this calculation was 2. We obtained the upper control limit, which was 1.50. This aided in our determination that there was abnormal behavior because our chi-square result was greater than our upper control limit. We therefore signaled an attack was occurring that showed a non-uniform spread in conjunction with abnormal behavior.

Replay Session Behavioral Analyzer

Observed Activity

ST	ET	PR	P1	P2	P3	P4	C	KS	SID
1220	1230	NS	M	B	Null	Null	Y	128	546456
1230	1240	NS	M	B	Null	Null	Y	128	546456
1240	1250	NS	M	B	Null	Null	Y	128	546456

Figure 20

In Figure 20, Observed Activity is captured by the EH representing the entire accumulated afternoon time category. The rows represented replay sessions between M and B. The Observed Activity was immediately placed within the Observed Activity Log as mentioned before. Once the afternoon category was realized, the B-IDE performed the system check.

The B-IDE requested the time degradation Normal Profile Statistics as displayed in Figure 21. After this was complete, the B-IDE retrieved the corresponding Observed Statistics as displayed in Figure 22.

Next, the measure of spread was performed, resulting in the calculation of 1.34. This showed that the metric values were not occurring uniformly over this

time period. The B-IDE next checked for any abnormal characteristics. The result of this calculation was 0.50.

Normal Profile Statistics

Metric	Value
Number of Replay Sessions Hour 12	0
Number of Replay Sessions Hour 13	0
Number of Replay Sessions Hour 14	2
Number of Replay Sessions Hour 15	0
Number of Replay Sessions Hour 16	0
Total	**2**

Figure 21

Observed Statistics

Metric	Value
Number of Replay Sessions Hour 12	0
Number of Replay Sessions Hour 13	0
Number of Replay Sessions Hour 14	3
Number of Replay Sessions Hour 15	0
Number of Replay Sessions Hour 16	0
Total	**3**

Figure 22

We obtained the upper control limit, which was 1.50. This was different from previous calculations. It seems that this attack behavior was not showing a noticeable degree of abnormality. This is shown by the chi-square result being smaller than our upper control limit. This can and will happen because no anomaly based system will be accurate 100% of the time. In this situation though, we were aided by our test of uniformity because it raised the attack alarm. We therefore signaled an attack was occurring that showed a non-uniform spread.

CONCLUSION

In this paper we presented an anomaly-based system that can be a viable solution for intrusion detection in systems where payloads are encrypted. The system uses a representation of a normal profile to model historical behavior. It utilizes collected observed data for the purpose of measuring the spread during each time category. We are then able to detect any high concentrations of session activity, in any one of our time granularities.

A model of normal behavior is created using an algorithm that places a premium on more current data. By measuring for normalcy using the Normal and Observed Statistics we are able to determine what is normal or abnormal depending on temporal differences for each Behavioral Analyzer. The system calculates and analyzes behavior in real time so that the most accurate and current behavior is represented. This allows us to have a streamlined process of gathering data, creating and then subsequently evaluating models.

Our demonstration of the detection of abnormal session activity for each of our Behavioral Analyzers shows that based on multiple attack detection methods we can have a high degree of certainty in our final conclusions. These combined methods give us a powerful detection tool in the pursuit of capturing malicious behavior within a realistic time frame. The appendix shows the source code of the BSEADS program. It includes the entire header and definition C++ files.

APPENDIX

The below code was made in standard C++. This is the BSEADS.h class header file.

```
// BSEADS.h: Header file for the BSEADS classes
// Classes Contained: BIDE, ACTIVITYHANDLER, PROFILE
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_BSEADS_H__DBCBCF7F_5B6E_4DF0_9892_834E96ADCB63__
INCLUDED_)
#define
AFX_BSEADS_H__DBCBCF7F_5B6E_4DF0_9892_834E96ADCB63__INCLUD
ED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

//Included files
#include <stdio.h>
#include <vector>
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <time.h>
#include <string>
#include <fstream>
#include <sstream>
#include <math.h>
#include <strstream>

//Use C++ standard library
using namespace std;

//ACTIVIVTYHANDLER class. User to handle observed activity.
class ACTIVITYHANDLER
{
public:
```

```
//Constructor
        ACTIVITYHANDLER();

        //Get monitor activity
        void getObservedActivity(char *);

        //Initialize observed activity
        void createObservedActivityLogs();

        //Write to observed activity logs
        void writeObservedActivityLog(string,int);

        //Print Monitor sessions
        void printMonitorVector();

        //Print Observed activity logs
        void printObservedLogs();

        //Clears observed logs
        void clearObservedLogs();

        //Returns monitor sessions
        vector<vector<string> > getMonitorActivity();

        //Contains userA observed activity
        vector<vector<string> > a;
        //Contains userB observed activity
        vector<vector<string> > b;
        //Contains userS observed activity
        vector<vector<string> > s;
        //Contains userM observed activity
        vector<vector<string> > m;

private:

        //Contains the monitor activity
        vector<vector<string> > v;

        char * observedLogA;      //File for user A observed log activity
        char * observedLogB;      //File for user B observed log activity
        char * observedLogS;      //File for user S observed log activity
        char * observedLogM;      //File for user M observed log activity

};
```

```
//PROFILEHANDLER class. Holds normal user profiles
class PROFILEHANDLER
{
public:
	//Constructor
	PROFILEHANDLER();

	//Loads user profile logs
	void loadUserProfile(char *);

	//Prints normal user profile logs
	void printNormalProfileLogs();

	//Contains userA normal profile
	vector<vector<string> > a;
	//Contains userB normal profile
	vector<vector<string> > b;
	//Contains userS normal profile
	vector<vector<string> > s;
	//Contains userM normal profile
	vector<vector<string> > m;

private:

};

//BIDE class - Intrusion Detection Engine Class - Main Controller Class
class BIDE
{
public:
	//Constructor
	BIDE();

	//Constructs instance of ACTIVITYHANDLER Class
	ACTIVITYHANDLER startActivityHandler();

	//Constructs instance of PROFILEHANDLER Class
	PROFILEHANDLER startProfileHandler();

	//Intrusion detection simulation starts here
	void startSimulation(ACTIVITYHANDLER &,PROFILEHANDLER &);

	//Accumulate hourly activity
	void accumulateActivity(int,ACTIVITYHANDLER &);

	//Generate statistics for hourly activity
```

```
        void generateStatistics(string, ACTIVITYHANDLER &,
PROFILEHANDLER &);

        //Counter for normal profile statistics
        void countStatistics(PROFILEHANDLER &, int, int);

        //Counter for observed statistics
        void countStatistics(ACTIVITYHANDLER &, int, int);

        //Print statistics to screen
        void printStatistics(string, int []);

        //Print statistics to screen
        void printStatistics();

        //Used to clear variables
        void clearVariables();

        //Measure standard deviation/volatility/uniformity of session activity
        void measureSpread(string, int [], int);

        void performChiSquare(string, int [], int [],int);

        double chiSqr(int [], int []);

        double chiSqr(int, int);

        double upperLimit(double);

        //Print header of software program
        void printHeader();

        //Check to see if the standard deviation is uniform
        void checkBehavior(double, string);

        void checkBehavior(double, double, string);

        //Transition to attack state if spread is not uniform
        void attackState(string,string);

        //Determine average for standard deviation
        double avg(int [], int);

        //Determine standard deviation
        double std(int [], int, double);
```

```
private:

	//Variables
	int aCount;
	int bCount;
	int sCount;
	int mCount;
	int count;
	string sTime;
	string eTime;
	string replayObservedID[1000];
	string replayProfileID[1000];

	double stdDevLimit;

	//Parallel Session Observed Statistic Container
	int parallelSessionObservedStatistics[24];

	//Failed Session Observed Statistic Container
	int failedSessionObservedStatistics[24];

	//Weak Session Observed Statistic Container
	int weakSessionObservedStatistics[24];

	//Replay Session Observed Statistic Container
	int replaySessionObservedStatistics[24];

	//Normal Profile Statistic Containers
	int parallelSessionProfileStatistics[24];
	int failedSessionProfileStatistics[24];
	int weakSessionProfileStatistics[24];
	int replaySessionProfileStatistics[24];

};



#endif //
!defined(AFX_BSEADS_H__DBCBCF7F_5B6E_4DF0_9892_834E96ADCB63__
INCLUDED_)
```

The below code was made in standard C++. This is the BSEADS.cpp class definition file.

```
// BSEADS.cpp: implementation of the PROFILE class.
//
```

```
//////////////////////////////////////////////////////////////////////

#include "BSEADS.h"

//BIDE CLASS
BIDE::BIDE()
{
	//Initialize variables
	stdDevLimit = 1.0;
	aCount = 0;
	bCount = 0;
	sCount = 0;
	mCount = 0;
	count = 0;
	sTime = "Null";
	eTime = "Null";

	//Initialize observed and profile statistic holders
	for (int k = 0; k < 24; k++)
	{
		parallelSessionObservedStatistics[k] = 0;
		failedSessionObservedStatistics[k] = 0;
		weakSessionObservedStatistics[k] = 0;
		replaySessionObservedStatistics[k] = 0;
		parallelSessionProfileStatistics[k] = 0;
		failedSessionProfileStatistics[k] = 0;
		weakSessionProfileStatistics[k] = 0;
		replaySessionProfileStatistics[k] = 0;
	}

	//Initialized for replay session purposes
	for (k = 0; k < 1000; k++)
	{
		replayObservedID[k] = "Null";
		replayProfileID[k] = "Null";
	}
}

//Start activity handler
ACTIVITYHANDLER BIDE::startActivityHandler()
{
	//Monitor sessions are stored here
	char * activityFile = "monitorActivity.csv";

	//Start Activity handler
	cout << "Activity Handler Starting Up.....";
```

```
	ACTIVITYHANDLER run = ACTIVITYHANDLER();
	cout << "OK" << endl;

	//Get observed monitor activity
	cout << "Activity Handler Interfacing with Monitor.....";
	run.getObservedActivity(activityFile);
	cout << "OK" << endl;

	//Initialize observed activity logs
	cout << "Initializing Observed Activity Logs.....OK" << endl;
	run.createObservedActivityLogs();

	//Return ACTIVITYHANDLER object
	return run;
}

//Start PROFILEHANDLER
PROFILEHANDLER BIDE::startProfileHandler()
{
	//User normal profile activities are stored in these files
	char * profileA = "profileLogA.csv";
	char * profileB = "profileLogB.csv";
	char * profileS = "profileLogS.csv";
	char * profileM = "profileLogM.csv";

	//Create PROFILEHANDLER instance
	cout << "Profile Handler Starting Up.....";
	PROFILEHANDLER run = PROFILEHANDLER();
	cout << "OK" << endl;

	//Load normal user profile logs
	cout << "Loading User A Profile.....";
	run.loadUserProfile(profileA);

	cout << "Loading User B Profile.....";
	run.loadUserProfile(profileB);

	cout << "Loading User S Profile.....";
	run.loadUserProfile(profileS);

	cout << "Loading User M Profile.....";
	run.loadUserProfile(profileM);

	//Return PROFILEHANDLER object
	return run;
}
```

```
//Print entire hourly statistics
void BIDE::printStatistics()
{
        //Print normal profile statistics
        cout << endl << endl;
        cout << "******NORMAL PROFILE STATISTICS******" << endl << endl;
        cout << "          Parallel Session" << endl;
        cout << "          ----------------" << endl;
        printStatistics("Parallel", parallelSessionProfileStatistics);
        cout << "           Failed Session" << endl;
        cout << "           --------------" << endl;
   printStatistics("Failed", failedSessionProfileStatistics);
        cout << "            Weak Session" << endl;
        cout << "            ------------" << endl;
        printStatistics("Weak", weakSessionProfileStatistics);
        cout << "           Replay Session" << endl;
        cout << "           --------------" << endl;
        printStatistics("Replay", replaySessionProfileStatistics);
        cout << endl;

        //Print observed statistics
        cout << "******OBSERVED STATISTICS******" << endl << endl;
        cout << "        Parallel Session" << endl;
        cout << "        ----------------" << endl;
        printStatistics("Parallel", parallelSessionObservedStatistics);
        cout << "         Failed Session" << endl;
        cout << "         --------------" << endl;
   printStatistics("Failed", failedSessionObservedStatistics);
        cout << "          Weak Session" << endl;
        cout << "          ------------" << endl;
        printStatistics("Weak", weakSessionObservedStatistics);
        cout << "         Replay Session" << endl;
        cout << "         --------------" << endl;
        printStatistics("Replay", replaySessionObservedStatistics);
        cout << "*********************************************************" << endl;
        cout << "*******************IDS SIMULATION ENDING*****************" <<
endl;
        cout << "*********************************************************" << endl <<
endl;

}

//Print entire hourly statistics
void BIDE::printStatistics(string type, int array[])
{
```

```
	for (int k = 0, t = 1, total = 0; k < 24; k++,t++)
	{
		cout << "Number of " << type << " Sessions Hour "<< t << " " <<
array[k] << endl;
		total+=array[k];
		if (t == 6)
		{
			cout << "Number of " << type << " Sessions Early Morning
Period " << total << endl << endl;
			total = 0;
		}
		else if(t == 11)
		{
			cout << "Number of " << type << " Sessions Morning Period "
<< total << endl << endl;
			total = 0;
		}
		else if(t == 16)
		{
			cout << "Number of " << type << " Sessions Afternoon
Period " << total << endl << endl;
			total = 0;
		}
		else if(t == 19)
		{
			cout << "Number of " << type << " Sessions Evening Period "
<< total << endl << endl;
			total = 0;
		}
		else if(t == 24)
		{
			cout << "Number of " << type << " Sessions Night Period "
<< total << endl << endl;
			total = 0;
		}
	}
	cout << endl;
}

void BIDE::printHeader()
{
	cout << "*********************************************************" << endl;
	cout << "*Welcome to the Behavior-Based Intrusion Detection System*"
<< endl;
	cout << "*                                                       *" << endl;
	cout << "*Version 1.0                                               *" << endl;
```

```
        cout << "*Created by Tysen Leckie                          *" << endl;
        cout << "*Florida State University                        *" << endl;
        cout << "*Computer Science Department                          *" << endl;
        cout << "**********************************************************" << endl <<
endl;
}

//Start Intrusion Detection Simulation
void BIDE::startSimulation(ACTIVITYHANDLER & observedActivity,
PROFILEHANDLER & userProfiles)
{

        cout << endl << "**********************************************************" <<
endl;
        cout << "********************IDS SIMULATION STARTING***************" <<
endl;
        cout << "**********************************************************" << endl <<
endl;
        //Accumulate hourly activity
        cout << "Accumulating Hour 1 Session Activity.....OK" << endl;
        accumulateActivity(1,observedActivity);
        cout << "Accumulating Hour 2 Session Activity.....OK" << endl;
        accumulateActivity(2,observedActivity);
        cout << "Accumulating Hour 3 Session Activity.....OK" << endl;
        accumulateActivity(3,observedActivity);
        cout << "Accumulating Hour 4 Session Activity.....OK" << endl;
        accumulateActivity(4,observedActivity);
        cout << "Accumulating Hour 5 Session Activity.....OK" << endl;
        accumulateActivity(5,observedActivity);
        cout << "Accumulating Hour 6 Session Activity.....OK" << endl <<
endl;
        accumulateActivity(6,observedActivity);

        //Check Behavior
        cout << "**********************************************************" << endl;
        cout << "Running Behavior Check on Early Morning Activity....." <<
endl;
        generateStatistics("EM",observedActivity, userProfiles);
        measureSpread("EM",parallelSessionObservedStatistics,0);
        performChiSquare("EM",parallelSessionObservedStatistics,parallelSessio
nProfileStatistics,0);
        measureSpread("EM",failedSessionObservedStatistics,1);
        performChiSquare("EM",failedSessionObservedStatistics,failedSessionPr
ofileStatistics,1);
        measureSpread("EM",weakSessionObservedStatistics,2);
```

```
        performChiSquare("EM",weakSessionObservedStatistics,weakSessionPro
fileStatistics,2);
        measureSpread("EM",replaySessionObservedStatistics,3);
        performChiSquare("EM",replaySessionObservedStatistics,replaySessionP
rofileStatistics,3);
        cout << "************************************************************" << endl <<
endl;

        //Accumulate hourly activity
        cout << "Accumulating Hour 7 Session Activity.....OK" << endl;
        accumulateActivity(7,observedActivity);
        cout << "Accumulating Hour 8 Session Activity.....OK" << endl;
        accumulateActivity(8,observedActivity);
        cout << "Accumulating Hour 9 Session Activity.....OK" << endl;
        accumulateActivity(9,observedActivity);
        cout << "Accumulating Hour 10 Session Activity.....OK" << endl;
        accumulateActivity(10,observedActivity);
        cout << "Accumulating Hour 11 Session Activity.....OK" << endl
<< endl;
        accumulateActivity(11,observedActivity);

        //Check Behavior
        cout << "************************************************************" << endl;
        cout << "Running Behavior Check on Morning Activity....." << endl;
        generateStatistics("M",observedActivity, userProfiles);
        measureSpread("M",parallelSessionObservedStatistics,0);
        performChiSquare("M",parallelSessionObservedStatistics,parallelSession
ProfileStatistics,0);
        measureSpread("M",failedSessionObservedStatistics,1);
        performChiSquare("M",failedSessionObservedStatistics,failedSessionProfi
leStatistics,1);
        measureSpread("M",weakSessionObservedStatistics,2);
        performChiSquare("M",weakSessionObservedStatistics,weakSessionProfi
leStatistics,2);
        measureSpread("M",replaySessionObservedStatistics,3);
        performChiSquare("M",replaySessionObservedStatistics,replaySessionPr
ofileStatistics,3);
        cout << "************************************************************" << endl <<
endl;

        //Accumulate hourly activity
        cout << "Accumulating Hour 12 Session Activity.....OK" << endl;
        accumulateActivity(12,observedActivity);
        cout << "Accumulating Hour 13 Session Activity.....OK" << endl;
        accumulateActivity(13,observedActivity);
        cout << "Accumulating Hour 14 Session Activity.....OK" << endl;
```

```
        accumulateActivity(14,observedActivity);
        cout << "Accumulating Hour 15 Session Activity.....OK" << endl;
        accumulateActivity(15,observedActivity);
        cout << "Accumulating Hour 16 Session Activity.....OK" << endl
<< endl;
        accumulateActivity(16,observedActivity);

        //Check Behavior
        cout << "*********************************************************" << endl;
        cout << "Running Behavior Check on Afternoon Activity....." << endl;
        generateStatistics("A",observedActivity, userProfiles);
        measureSpread("A",parallelSessionObservedStatistics,0);
        performChiSquare("A",parallelSessionObservedStatistics,parallelSession
ProfileStatistics,0);
        measureSpread("A",failedSessionObservedStatistics,1);
        performChiSquare("A",failedSessionObservedStatistics,failedSessionProfil
eStatistics,1);
        measureSpread("A",weakSessionObservedStatistics,2);
        performChiSquare("A",weakSessionObservedStatistics,weakSessionProfil
eStatistics,2);
        measureSpread("A",replaySessionObservedStatistics,3);
        performChiSquare("A",replaySessionObservedStatistics,replaySessionPro
fileStatistics,3);
        cout << "*********************************************************" << endl <<
endl;

        //Accumulate hourly activity
        cout << "Accumulating Hour 17 Session Activity.....OK" << endl;
        accumulateActivity(17,observedActivity);
        cout << "Accumulating Hour 18 Session Activity.....OK" << endl;
        accumulateActivity(18,observedActivity);
        cout << "Accumulating Hour 19 Session Activity.....OK" << endl
<< endl;
        accumulateActivity(19,observedActivity);

        //Check Behavior
        cout << "*********************************************************" << endl;
        cout << "Running Behavior Check on Evening Activity....." << endl;
        generateStatistics("E",observedActivity, userProfiles);
        measureSpread("E",parallelSessionObservedStatistics,0);
        performChiSquare("E",parallelSessionObservedStatistics,parallelSession
ProfileStatistics,0);
        measureSpread("E",failedSessionObservedStatistics,1);
        performChiSquare("E",failedSessionObservedStatistics,failedSessionProfil
eStatistics,1);
        measureSpread("E",weakSessionObservedStatistics,2);
```

```
        performChiSquare("E",weakSessionObservedStatistics,weakSessionProfil
eStatistics,2);
        measureSpread("E",replaySessionObservedStatistics,3);
        performChiSquare("E",replaySessionObservedStatistics,replaySessionPro
fileStatistics,3);
        cout << "**********************************************************" << endl <<
endl;

        //Accumulate hourly activity
        cout << "Accumulating Hour 20 Session Activity.....OK" << endl;
        accumulateActivity(20,observedActivity);
        cout << "Accumulating Hour 21 Session Activity.....OK" << endl;
        accumulateActivity(21,observedActivity);
        cout << "Accumulating Hour 22 Session Activity.....OK" << endl;
        accumulateActivity(22,observedActivity);
        cout << "Accumulating Hour 23 Session Activity.....OK" << endl;
        accumulateActivity(23,observedActivity);
        cout << "Accumulating Hour 24 Session Activity.....OK" << endl
<< endl;
        accumulateActivity(24,observedActivity);

        //Check Behavior
        cout << "**********************************************************" << endl;
        cout << "Running Behavior Check on Night Activity....." << endl;
        generateStatistics("N",observedActivity, userProfiles);
        measureSpread("N",parallelSessionObservedStatistics,0);
        performChiSquare("N",parallelSessionObservedStatistics,parallelSession
ProfileStatistics,0);
        measureSpread("N",failedSessionObservedStatistics,1);
        performChiSquare("N",failedSessionObservedStatistics,failedSessionProfi
leStatistics,1);
        measureSpread("N",weakSessionObservedStatistics,2);
        performChiSquare("N",weakSessionObservedStatistics,weakSessionProfil
eStatistics,2);
        measureSpread("N",replaySessionObservedStatistics,3);
        performChiSquare("N",replaySessionObservedStatistics,replaySessionPro
fileStatistics,3);
        cout << "**********************************************************" << endl <<
endl;

        //Check Behavior
        cout << "**********************************************************" << endl;
        cout << "Running Behavior Check on Day Activity Per Hour....." <<
endl;
        generateStatistics("DHOUR",observedActivity, userProfiles);
        measureSpread("DHOUR",parallelSessionObservedStatistics,0);
```

```
        performChiSquare("DHOUR",parallelSessionObservedStatistics,parallelS
essionProfileStatistics,0);
        measureSpread("DHOUR",failedSessionObservedStatistics,1);
        performChiSquare("DHOUR",failedSessionObservedStatistics,failedSessi
onProfileStatistics,1);
        measureSpread("DHOUR",weakSessionObservedStatistics,2);
        performChiSquare("DHOUR",weakSessionObservedStatistics,weakSessio
nProfileStatistics,2);
        measureSpread("DHOUR",replaySessionObservedStatistics,3);
        performChiSquare("DHOUR",replaySessionObservedStatistics,replaySess
ionProfileStatistics,3);
        cout << "************************************************************" << endl <<
endl;

        //Check Behavior
        cout << "************************************************************" << endl;
        cout << "Running Behavior Check on Day Activity Per Time Category.." <<
endl;
        generateStatistics("DTIME",observedActivity, userProfiles);
        measureSpread("DTIME",parallelSessionObservedStatistics,0);
        performChiSquare("DTIME",parallelSessionObservedStatistics,parallelSes
sionProfileStatistics,0);
        measureSpread("DTIME",failedSessionObservedStatistics,1);
        performChiSquare("DTIME",failedSessionObservedStatistics,failedSessio
nProfileStatistics,1);
        measureSpread("DTIME",weakSessionObservedStatistics,2);
        performChiSquare("DTIME",weakSessionObservedStatistics,weakSessio
nProfileStatistics,2);
        measureSpread("DTIME",replaySessionObservedStatistics,3);
        performChiSquare("DTIME",replaySessionObservedStatistics,replaySessi
onProfileStatistics,3);
        cout << "************************************************************" << endl <<
endl;
}

double BIDE::chiSqr(int oTotal, int pTotal)
{

        int subResult;
        double powResult;


        subResult = oTotal - pTotal;
        powResult = pow(subResult,2);

        if (pTotal == 0)
```

```
                return (powResult/1);
        else
                return (powResult/pTotal);
}

double BIDE::chiSqr(int oTemp[], int pTemp[])
{
        int subResult;
        double powResult;
        double divResult;
        double totResult = 0;

        for (int z = 0; z < 5; z++)
        {
                subResult = oTemp[z] - pTemp[z];
                powResult = pow(subResult,2);
                if (pTemp[z] == 0)
                {
                        divResult =  powResult/1;
                }
                else
                {
                        divResult = powResult/pTemp[z];
                }
                totResult += divResult;
        }
        return totResult;
}

double BIDE::upperLimit(double StdDev)
{
        return 2*StdDev;
}

void BIDE::performChiSquare(string type,int observedStatistics[],int
normalStatistics[],int index)
{
        if (type == "EM")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0,0,0,0};
                int size = 6;
                double ChiSquare, uLimit, StdDev;
                for (int z = 0; z < 6; z++)
                {
```

```
                pTemp[z] = normalStatistics[z];
                oTotal += observedStatistics[z];
                pTotal += normalStatistics[z];
        }
        //Parallel Session Chi-Square
        if (index == 0)
        {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Early Morning Parallel Session Chi-Square Value:
"<< ChiSquare << endl;
                cout <<"Early Morning Parallel Session Upper Control Limit:
" << uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"EM");
        }
        //Failed Session Chi-Square
        else if (index == 1)
        {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Early Morning Failed Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Early Morning Failed Session Upper Control Limit: "
<< uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"EM");
        }
        //Weak Session Chi-Square
        else if (index == 2)
        {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Early Morning Weak Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Early Morning Weak Session Upper Control Limit: "
<< uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"EM");
        }
        //Replay Session Chi-Square
        else if (index == 3)
        {
```

```
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Early Morning Replay Session Chi-Square Value:
"<< ChiSquare << endl;
                        cout <<"Early Morning Replay Session Upper Control Limit: "
<< uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"EM");
                }
        }
        else if (type == "M")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0,0,0};
                int size = 5;
                double ChiSquare, uLimit, StdDev;
                int y = 0;
                for (int z = 6; z < 11; z++)
                {
                        pTemp[y] = normalStatistics[z];
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
                        y++;
                }
                //Parallel Session Chi-Square
                if (index == 0)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Morning Parallel Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Morning Parallel Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"M");
                }
                //Failed Session Chi-Square
                else if (index == 1)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
```

```
                        cout <<"Morning Failed Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Morning Failed Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"M");
                }
                //Weak Session Chi-Square
                else if (index == 2)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Morning Weak Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Morning Weak Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"M");
                }
                //Replay Session Chi-Square
                else if (index == 3)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Morning Replay Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Morning Replay Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"M");
                }
        }
        else if (type == "A")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0,0,0};
                int size = 5;
                double ChiSquare, uLimit, StdDev;
                int y = 0;
                for (int z = 11; z < 16; z++)
                {
                        pTemp[y] = normalStatistics[z];
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
```

```
                y++;
            }
            //Parallel Session Chi-Square
            if (index == 0)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Afternoon Parallel Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Afternoon Parallel Session Upper Control Limit: " <<
uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"A");
            }
            //Failed Session Chi-Square
            else if (index == 1)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Afternoon Failed Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Afternoon Failed Session Upper Control Limit: " <<
uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"A");
            }
            //Weak Session Chi-Square
            else if (index == 2)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Afternoon Weak Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Afternoon Weak Session Upper Control Limit: " <<
uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"A");
            }
            //Replay Session Chi-Square
            else if (index == 3)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
```

```
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Afternoon Replay Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Afternoon Replay Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"A");
                }

        }
        else if (type == "E")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0};
                int size = 3;
                double ChiSquare, uLimit, StdDev;
                int y = 0;
                for (int z = 16; z < 19; z++)
                {
                        pTemp[y] = normalStatistics[z];
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
                        y++;
                }
                //Parallel Session Chi-Square
                if (index == 0)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Evening Parallel Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Evening Parallel Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"E");
                }
                //Failed Session Chi-Square
                else if (index == 1)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
```

```
                        cout <<"Evening Failed Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Evening Failed Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"E");
                }
                //Weak Session Chi-Square
                else if (index == 2)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Evening Weak Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Evening Weak Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"E");
                }
                //Replay Session Chi-Square
                else if (index == 3)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Evening Replay Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Evening Replay Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"E");
                }
        }
        else if (type == "N")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0,0,0};
                int size = 5;
                double ChiSquare, uLimit, StdDev;
                int y = 0;
                for (int z = 19; z < 24; z++)
                {
                        pTemp[y] = normalStatistics[z];
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
```

```
                y++;
            }
            //Parallel Session Chi-Square
            if (index == 0)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Night Parallel Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Night Parallel Session Upper Control Limit: " <<
uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"N");
            }
            //Failed Session Chi-Square
            else if (index == 1)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Night Failed Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Night Failed Session Upper Control Limit: " <<
uLimit << endl;
                checkBehavior(ChiSquare,uLimit,"N");
            }
            //Weak Session Chi-Square
            else if (index == 2)
            {
                ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
                uLimit = upperLimit(StdDev);
                cout <<setiosflags(ios::fixed)<<setprecision(2);
                cout <<"Night Weak Session Chi-Square Value: "<<
ChiSquare << endl;
                cout <<"Night Weak Session Upper Control Limit: " << uLimit
<< endl;
                checkBehavior(ChiSquare,uLimit,"N");
            }
            //Replay Session Chi-Square
            else if (index == 3)
            {
            ChiSquare = chiSqr(oTotal, pTotal);
                StdDev = std(pTemp, size, avg(pTemp,size));
```

```
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Night Replay Session Chi-Square Value: "<<
ChiSquare << endl;
                        cout <<"Night Replay Session Upper Control Limit: " <<
uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"N");
                }
        }
        else if (type == "DHOUR")
        {
                int oTotal = 0;
                int pTotal = 0;
                int size = 24;
                double ChiSquare, uLimit, StdDev;
                for (int z = 0; z < 24; z++)
                {
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
                }
                //Parallel Session Chi-Square
                if (index == 0)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(normalStatistics, size,
avg(normalStatistics,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Parallel Session Chi-Square Result Per Hour:
"<< ChiSquare << endl;
                        cout <<"Day Parallel Session Upper Control Limit Per Hour: "
<< uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DHOUR");
                }
                //Failed Session Chi-Square
                else if (index == 1)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(normalStatistics, size,
avg(normalStatistics,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Failed Session Chi-Square Result Per Hour:
"<< ChiSquare << endl;
                        cout <<"Day Failed Session Upper Control Limit Per Hour: "
<< uLimit << endl;
```

```
                        checkBehavior(ChiSquare,uLimit,"DHOUR");
                }
                //Weak Session Chi-Square
                else if (index == 2)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(normalStatistics, size,
avg(normalStatistics,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Weak Session Chi-Square Result Per Hour: "<<
ChiSquare << endl;
                        cout <<"Day Weak Session Upper Control Limit Per Hour: "
<< uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DHOUR");
                }
                //Replay Session Chi-Square
                else if (index == 3)
                {
                        ChiSquare = chiSqr(oTotal, pTotal);
                        StdDev = std(normalStatistics, size,
avg(normalStatistics,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Replay Session Chi-Square Result Per Hour:
"<< ChiSquare << endl;
                        cout <<"Day Replay Session Upper Control Limit Per Hour: "
<< uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DHOUR");
                }
        }
        else if (type == "DTIME")
        {
                int oTotal = 0;
                int pTotal = 0;
                int pTemp[] = {0,0,0,0,0};
                int oTemp[] = {0,0,0,0,0};
                int size = 5;
                double ChiSquare, uLimit, StdDev;
                //Accumulate category totals for early morning period
                for (int z = 0; z < 6; z++)
                {
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
                }
                if (oTotal > 0)
```

```
{
        oTemp[0] = oTotal;
        oTotal = 0;
}
if (pTotal > 0)
{
        pTemp[0] = pTotal;
        pTotal = 0;
}
//Accumulate category totals for morning period
for (z = 6; z < 11; z++)
{
        oTotal += observedStatistics[z];
        pTotal += normalStatistics[z];
}
if (oTotal > 0)
{
        oTemp[1] = oTotal;
        oTotal = 0;
}
if (pTotal > 0)
{
        pTemp[1] = pTotal;
        pTotal = 0;
}
//Accumulate category totals for afternoon period
for (z = 11; z < 16; z++)
{
        oTotal += observedStatistics[z];
        pTotal += normalStatistics[z];
}
if (oTotal > 0)
{
        oTemp[2] = oTotal;
        oTotal = 0;
}
if (pTotal > 0)
{
        pTemp[2] = pTotal;
        pTotal = 0;
}
//Accumulate category totals for evening period
for (z = 16; z < 19; z++)
{
        oTotal += observedStatistics[z];
        pTotal += normalStatistics[z];
```

```
                }
                if (oTotal > 0)
                {
                        oTemp[3] = oTotal;
                        oTotal = 0;
                }
                if (pTotal > 0)
                {
                        pTemp[3] = pTotal;
                        pTotal = 0;
                }
                //Accumulate category totals for night period
                for (z = 19; z < 24; z++)
                {
                        oTotal += observedStatistics[z];
                        pTotal += normalStatistics[z];
                }
                if (oTotal > 0)
                {
                        oTemp[4] = oTotal;
                        oTotal = 0;
                }
                if (pTotal > 0)
                {
                        pTemp[4] = pTotal;
                        pTotal = 0;
                }
                //Parallel Session Chi-Square
                if (index == 0)
                {
                        ChiSquare = chiSqr(oTemp, pTemp);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Parallel Session Chi-Square Result Per Time
Category: "<< ChiSquare << endl;
                        cout <<"Day Parallel Session Upper Control Limit Per Time
Category: " << uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DTIME");
                }
                //Failed Session Chi-Square
                else if (index == 1)
                {
                        ChiSquare = chiSqr(oTemp, pTemp);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
```

```
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Failed Session Chi-Square Result Per Time
Category: "<< ChiSquare << endl;
                        cout <<"Day Failed Session Upper Control Limit Per Time
Category: " << uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DTIME");
                }
                //Weak Session Chi-Square
                else if (index == 2)
                {
                        ChiSquare = chiSqr(oTemp, pTemp);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Weak Session Chi-Square Result Per Time
Category: "<< ChiSquare << endl;
                        cout <<"Day Weak Session Upper Control Limit Per Time
Category: " << uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DTIME");
                }
                //Replay Session Chi-Square
                else if (index == 3)
                {
                        ChiSquare = chiSqr(oTemp, pTemp);
                        StdDev = std(pTemp, size, avg(pTemp,size));
                        uLimit = upperLimit(StdDev);
                        cout <<setiosflags(ios::fixed)<<setprecision(2);
                        cout <<"Day Replay Session Chi-Square Result Per Time
Category: "<< ChiSquare << endl;
                        cout <<"Day Replay Session Upper Control Limit Per Time
Category: " << uLimit << endl;
                        checkBehavior(ChiSquare,uLimit,"DTIME");
                }
        }

}

void BIDE::measureSpread(string type, int array[], int index)
{

        if (type == "EM")
        {
                int temp[] = {0,0,0,0,0,0};
                int size = 6;
                double StdDev;
```

```
                for (int z = 0; z < 6; z++)
                {
                        temp[z] = array[z];
                }
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Early Morning Parallel Session Standard Deviation:
"<< StdDev << endl;
                        checkBehavior(StdDev, "EM");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Early Morning Failed Session Standard Deviation:
"<< StdDev << endl;
                        checkBehavior(StdDev, "EM");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Early Morning Weak Session Standard Deviation:
"<< StdDev << endl;
                        checkBehavior(StdDev, "EM");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Early Morning Replay Session Standard Deviation:
"<< StdDev << endl;
                        checkBehavior(StdDev, "EM");
                }
        }
        else if (type == "M")
        {
                int temp[] = {0,0,0,0,0};
                int size = 5;
                double StdDev;
```

```
                int y = 0;
                for (int z = 6; z < 11; z++)
                {
                        temp[y] = array[z];
                        y++;
                }
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Morning Parallel Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "M");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Morning Failed Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "M");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Morning Weak Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "M");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Morning Replay Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "M");
                }
        }
        else if (type == "A")
        {
                int temp[] = {0,0,0,0,0};
```

```
                int size = 5;
                double StdDev;
                int y = 0;
                for (int z = 11; z < 16; z++)
                {
                        temp[y] = array[z];
                        y++;
                }
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Afternoon Parallel Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "A");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Afternoon Failed Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "A");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Afternoon Weak Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "A");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Afternoon Replay Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "A");
                }

        }
```

```
        else if (type == "E")
        {
                int temp[] = {0,0,0};
                int size = 3;
                double StdDev;
                int y = 0;
                for (int z = 16; z < 19; z++)
                {
                        temp[y] = array[z];
                        y++;
                }
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Evening Parallel Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "E");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Evening Failed Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "E");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Evening Weak Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "E");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Evening Replay Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "E");
```

```
                }
        }
        else if (type == "N")
        {
                int temp[] = {0,0,0,0,0};
                int size = 5;
                double StdDev;
                int y = 0;
                for (int z = 19; z < 24; z++)
                {
                        temp[y] = array[z];
                        y++;
                }
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Night Parallel Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "N");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Night Failed Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "N");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Night Weak Session Standard Deviation: "<< StdDev
<< endl;
                        checkBehavior(StdDev, "N");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(temp, size, avg(temp,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
```

```
                        cout<<"Night Replay Session Standard Deviation: "<<
StdDev << endl;
                        checkBehavior(StdDev, "N");
                }
        }
        else if (type == "DHOUR")
        {
                int size = 24;
                double StdDev;
                //Parallel Session Spread
                if (index == 0)
                {
                        StdDev = std(array, size, avg(array,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Day Parallel Session Standard Deviation Per Hour:
"<< StdDev << endl;
                        checkBehavior(StdDev, "DHOUR");
                }
                //Failed Session Spread
                else if (index == 1)
                {
                        StdDev = std(array, size, avg(array,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Day Failed Session Standard Deviation Per Hour:
"<< StdDev << endl;
                        checkBehavior(StdDev, "DHOUR");
                }
                //Weak Session Spread
                else if (index == 2)
                {
                        StdDev = std(array, size, avg(array,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Day Weak Session Standard Deviation Per Hour:
"<< StdDev << endl;
                        checkBehavior(StdDev, "DHOUR");
                }
                //Replay Session Spread
                else if (index == 3)
                {
                        StdDev = std(array, size, avg(array,size));
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Day Replay Session Standard Deviation Per Hour:
"<< StdDev << endl;
                        checkBehavior(StdDev, "DHOUR");
                }
        }
```

```
else if (type == "DTIME")
{
        int temp[] = {0,0,0,0,0};
        int size = 5;
        double StdDev;
        int total = 0;

        //Accumulate category totals for early morning period
        for (int z = 0; z < 6; z++)
        {
                total += array[z];
        }
        if (total > 0)
        {
                temp[0] = total;
                total = 0;
        }
        //Accumulate category totals for morning period
        for (z = 6; z < 11; z++)
        {
                total += array[z];
        }
        if (total > 0)
        {
                temp[1] = total;
                total = 0;
        }

        //Accumulate category totals for afternoon period
        for (z = 11; z < 16; z++)
        {
                total += array[z];
        }
        if (total > 0)
        {
                temp[2] = total;
                total = 0;
        }

        //Accumulate category totals for evening period
        for (z = 16; z < 19; z++)
        {
                total += array[z];
        }
        if (total > 0)
        {
```

```
                temp[3] = total;
                total = 0;
        }

        //Accumulate category totals for night period
        for (z = 19; z < 24; z++)
        {
                total += array[z];
        }
        if (total > 0)
        {
                temp[4] = total;
                total = 0;
        }

        //Parallel Session Spread
        if (index == 0)
        {
                StdDev = std(temp, size, avg(temp,size));
                cout<<setiosflags(ios::fixed)<<setprecision(2);
                cout<<"Day Parallel Session Standard Deviation Time
Category: "<< StdDev << endl;
                checkBehavior(StdDev, "DTIME");
        }
        //Failed Session Spread
        else if (index == 1)
        {
                StdDev = std(temp, size, avg(temp,size));
                cout<<setiosflags(ios::fixed)<<setprecision(2);
                cout<<"Day Failed Session Standard Deviation Time
Category: "<< StdDev << endl;
                checkBehavior(StdDev, "DTIME");
        }
        //Weak Session Spread
        else if (index == 2)
        {
                StdDev = std(temp, size, avg(temp,size));
                cout<<setiosflags(ios::fixed)<<setprecision(2);
                cout<<"Day Weak Session Standard Deviation Time
Category: "<< StdDev << endl;
                checkBehavior(StdDev, "DTIME");
        }
        //Replay Session Spread
        else if (index == 3)
        {
                StdDev = std(temp, size, avg(temp,size));
```

```
                        cout<<setiosflags(ios::fixed)<<setprecision(2);
                        cout<<"Day Replay Session Standard Deviation Time
Category: "<< StdDev << endl;
                        checkBehavior(StdDev, "DTIME");
                }
        }
}

//Check standard deviation
void BIDE::checkBehavior(double StdDev, string timePeriod)
{
        if (StdDev > stdDevLimit)
                attackState(timePeriod, "UNIFORMLY");
}


//Check chi square and control limit values
void BIDE::checkBehavior(double ChiSqr, double uLimit, string timePeriod)
{
        if (ChiSqr > uLimit)
                attackState(timePeriod, "NORMALY");
}

//Transformation to attack state. Intrusion behavior was noticed
void BIDE::attackState(string timePeriod, string type)
{
        if (timePeriod == "EM")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER THE EARLY MORNING TIME PERIOD" << endl << endl;
        }
        else if (timePeriod == "M")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER THE MORNING TIME PERIOD" << endl << endl;
        }
        else if (timePeriod == "A")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER THE AFTERNOON TIME PERIOD" << endl << endl;
        }
        else if (timePeriod == "E")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER THE EVENING TIME PERIOD" << endl << endl;
        }
```

```
        else if (timePeriod == "N")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER THE NIGHT TIME PERIOD" << endl << endl;
        }
        else if (timePeriod == "DHOUR")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER DAY PERIOD PER HOUR" << endl << endl;
        }
        else if (timePeriod == "DTIME")
        {
                cout << endl << "WARNING: ACTIVITY IS NOT OCCURRING " <<
type << " OVER DAY PERIOD PER TIME CATEGORY" << endl << endl;
        }
}

//Determine average for standard deviation purposes
double BIDE::avg(int ArrInt[], int size)
{
  long sum = 0;
  for(int i = 0; i < size; i++)
  {
                sum = sum + ArrInt[i];
  }

  if(size != 0)
  {
    return (double) sum/size;
  }
  return 0;
}

//Determine standard deviation
double BIDE::std(int ArrInt[], int size, double average)
{
        double variance = 0;

        for(int i = 0; i < size;i++)
        {
                variance = variance + pow((average - ArrInt[i]),2);
        }
        return sqrt(variance / size);
}

//Accumulate hourly activity
```

```
void BIDE::accumulateActivity(int hour, ACTIVITYHANDLER & observedActivity)
{
	//Local variables
	string sTime;
	string eTime;

	//Temporary container for monitor data
	vector<string> tTwoDim(10);
   vector<vector<string> > v(20, tTwoDim);

	v = observedActivity.getMonitorActivity();

	//Contains users
	vector<string> users;
	users.push_back("A");
	users.push_back("B");
	users.push_back("S");
	users.push_back("M");

	//Used because hours are represented as strings
	switch(hour)
	{
		case 1:
			sTime = "0000";
			eTime = "0100";
			break;
		case 2:
			sTime = "0100";
			eTime = "0200";
			break;
		case 3:
			sTime = "0200";
			eTime = "0300";
			break;
		case 4:
			sTime = "0300";
			eTime = "0400";
			break;
		case 5:
			sTime = "0400";
			eTime = "0500";
			break;
		case 6:
			sTime = "0500";
			eTime = "0600";
			break;
```

```
case 7:
        sTime = "0600";
        eTime = "0700";
        break;
case 8:
        sTime = "0700";
        eTime = "0800";
        break;
case 9:
        sTime = "0800";
        eTime = "0900";
        break;
case 10:
        sTime = "0900";
        eTime = "1000";
        break;
case 11:
        sTime = "1000";
        eTime = "1100";
        break;
case 12:
        sTime = "1100";
        eTime = "1200";
        break;
case 13:
        sTime = "1200";
        eTime = "1300";
        break;
case 14:
        sTime = "1300";
        eTime = "1400";
        break;
case 15:
        sTime = "1400";
        eTime = "1500";
        break;
case 16:
        sTime = "1500";
        eTime = "1600";
        break;
case 17:
        sTime = "1600";
        eTime = "1700";
        break;
case 18:
        sTime = "1700";
```

```
                    eTime = "1800";
                    break;
            case 19:
                    sTime = "1800";
                    eTime = "1900";
                    break;
            case 20:
                    sTime = "1900";
                    eTime = "2000";
                    break;
            case 21:
                    sTime = "2000";
                    eTime = "2100";
                    break;
            case 22:
                    sTime = "2100";
                    eTime = "2200";
                    break;
            case 23:
                    sTime = "2200";
                    eTime = "2300";
                    break;
            case 24:
                    sTime = "2300";
                    eTime = "2400";
                    break;
            default:
                    //Value undefined
                    cout << "FAILURE";
                    break;
    }

    //Search Monitor Log
    for(int z = 0; z < v.size(); z++)
    {
      //Add corresponding monitor activity to user's observed activity log
      if((v[z][0] >= sTime) && (v[z][1] <= eTime))
      {
            for(int k = 0; k < users.size(); k++)
            {
                  if((v[z][3] == users[k]) || (v[z][4] == users[k]) || (v[z][5] ==
users[k]) || (v[z][6] == users[k]))
                  {
                        //User A
                        if (k == 0)
                        {
```

```
                                        observedActivity.a[aCount][0] = v[z][0];
                                        observedActivity.a[aCount][1] = v[z][1];
                                        observedActivity.a[aCount][2] = v[z][2];
                                        observedActivity.a[aCount][3] = v[z][3];
                                        observedActivity.a[aCount][4] = v[z][4];
                                        observedActivity.a[aCount][5] = v[z][5];
                                        observedActivity.a[aCount][6] = v[z][6];
                                        observedActivity.a[aCount][7] = v[z][7];
                                        observedActivity.a[aCount][8] = v[z][8];
                                        observedActivity.a[aCount][9] = v[z][9];
                                        //Write observed activity to observed activity
log

observedActivity.writeObservedActivityLog("A",z);
                                        aCount++;
                                }
                                //User B
                                else if (k == 1)
                                {
                                        observedActivity.b[bCount][0] = v[z][0];
                                        observedActivity.b[bCount][1] = v[z][1];
                                        observedActivity.b[bCount][2] = v[z][2];
                                        observedActivity.b[bCount][3] = v[z][3];
                                        observedActivity.b[bCount][4] = v[z][4];
                                        observedActivity.b[bCount][5] = v[z][5];
                                        observedActivity.b[bCount][6] = v[z][6];
                                        observedActivity.b[bCount][7] = v[z][7];
                                        observedActivity.b[bCount][8] = v[z][8];
                                        observedActivity.b[bCount][9] = v[z][9];
                                        //Write observed activity to observed
activity log

        observedActivity.writeObservedActivityLog("B",z);
                                        bCount++;
                                }
                                //User S
                                else if (k == 2)
                                {
                                        observedActivity.s[sCount][0] = v[z][0];
                                        observedActivity.s[sCount][1] = v[z][1];
                                        observedActivity.s[sCount][2] = v[z][2];
                                        observedActivity.s[sCount][3] = v[z][3];
                                        observedActivity.s[sCount][4] = v[z][4];
                                        observedActivity.s[sCount][5] = v[z][5];
                                        observedActivity.s[sCount][6] = v[z][6];
                                        observedActivity.s[sCount][7] = v[z][7];
```

```
                                        observedActivity.s[sCount][8] = v[z][8];
                                        observedActivity.s[sCount][9] = v[z][9];
                                        //Write observed activity to observed
activity log

        observedActivity.writeObservedActivityLog("S",z);
                                        sCount++;
                                }
                                //User M
                                else if (k == 3)
                                {
                                        observedActivity.m[mCount][0] = v[z][0];
                                        observedActivity.m[mCount][1] = v[z][1];
                                        observedActivity.m[mCount][2] = v[z][2];
                                        observedActivity.m[mCount][3] = v[z][3];
                                        observedActivity.m[mCount][4] = v[z][4];
                                        observedActivity.m[mCount][5] = v[z][5];
                                        observedActivity.m[mCount][6] = v[z][6];
                                        observedActivity.m[mCount][7] = v[z][7];
                                        observedActivity.m[mCount][8] = v[z][8];
                                        observedActivity.m[mCount][9] = v[z][9];
                                        //Write observed activity to observed
activity log

        observedActivity.writeObservedActivityLog("M",z);
                                        mCount++;
                                }
                        }
                }

          }
        }

}

//Count profile statistics
void BIDE::countStatistics(PROFILEHANDLER & userProfiles, int z, int hour)
{
        int keyStrength;

        //Determine parallel session activity
        if(sTime == "Null" || eTime == "Null")
        {
                sTime = userProfiles.m[z][0];
           eTime = userProfiles.m[z][1];
        }
```

```
        else
        {
                if((sTime == userProfiles.m[z][0]) && (eTime ==
userProfiles.m[z][1]))
                {
                        if(parallelSessionProfileStatistics[hour] < 1)
                        {
                                parallelSessionProfileStatistics[hour]++;
                                parallelSessionProfileStatistics[hour]++;
                        }
                        else
                        {
                                parallelSessionProfileStatistics[hour]++;
                        }
                }
        }

        //Determine replay session activity
        replayProfileID[z] = userProfiles.m[z][9];

        count = 0;
        string temp[1000];
        int e = 0;
        for (int h = 0; h < 1000; h++)
        {
                if (replayProfileID[h] != "Null")
                {
                        temp[e] = replayProfileID[h];
                        e++;
                }
        }


        string tempString = "";
        for (h = 0; h < e; h++)
        {
                if (h == 0)
                {
                        tempString = temp[h];
                }
                else
                {
                        if (tempString == temp[h])
                        {
                                replaySessionProfileStatistics[hour]++;
                        }
```

```
                else
                        tempString = temp[h];
                }
        }

        //Determine failed session activity
        if(userProfiles.m[z][7] == "N")
           failedSessionProfileStatistics[hour]++;

        //Conversion of string to integer
        istringstream(userProfiles.m[z][8]) >> keyStrength;

        //Determine weak encryption activity
        if(keyStrength <= 64)
           weakSessionProfileStatistics[hour]++;
}

//Count statistical behavior
void BIDE::countStatistics(ACTIVITYHANDLER & observedActivity, int z, int
hour)
{
        int keyStrength;

        //Parallel session count
        if(sTime == "Null" || eTime == "Null")
        {
                sTime = observedActivity.m[z][0];
           eTime = observedActivity.m[z][1];
        }
        else
        {
                if((sTime == observedActivity.m[z][0]) && (eTime ==
observedActivity.m[z][1]))
                {
                        if(parallelSessionObservedStatistics[hour] < 1)
                        {
                                parallelSessionObservedStatistics[hour]++;
                                parallelSessionObservedStatistics[hour]++;
                        }
                        else
                        {
                                parallelSessionObservedStatistics[hour]++;
                        }
                }
        }
```

```
        //Determine replay session activity
        replayObservedID[z] = observedActivity.m[z][9];

        count = 0;
        string temp[1000];
        int e = 0;
        for (int h = 0; h < 1000; h++)
        {
                if (replayObservedID[h] != "Null")
                {
                        temp[e] = replayObservedID[h];
                        e++;
                }
        }


        string tempString = "";
        for (h = 0; h < e; h++)
        {
                if (h == 0)
                {
                        tempString = temp[h];
                }
                else
                {
                        if (tempString == temp[h])
                        {
                                replaySessionObservedStatistics[hour]++;
                        }
                        else
                                tempString = temp[h];
                }
        }

        //Failed session count
        if(observedActivity.m[z][7] == "N")
           failedSessionObservedStatistics[hour]++;

        //Used to convert string to integer
        istringstream(observedActivity.m[z][8]) >> keyStrength;

        //Weak session count
        if(keyStrength <= 64)
           weakSessionObservedStatistics[hour]++;
}
```

```
//Used to clear variables
void BIDE::clearVariables()
{
            sTime = "Null";
                eTime = "Null";
                for (int k = 0; k < 1000; k++)
                {
                        replayObservedID[k] = "Null";
                        replayProfileID[k] = "Null";
                }
}



//Generate statistics
void BIDE::generateStatistics(string type, ACTIVITYHANDLER &
observedActivity, PROFILEHANDLER & userProfiles)
{

        //Early morning period
        if (type == "EM")
        {
                for (int z = 0; z < userProfiles.m.size(); z++)
                {
                        if((userProfiles.m[z][0] >= "0000") && (userProfiles.m[z][1] <=
"0100"))
                        {
                                countStatistics(userProfiles,z,0);
                        }
                        else if((userProfiles.m[z][0] >= "0100") &&
(userProfiles.m[z][1] <= "0200"))
                        {
                                countStatistics(userProfiles,z,1);
                        }
                        else if((userProfiles.m[z][0] >= "0200") &&
(userProfiles.m[z][1] <= "0300"))
                        {
                                countStatistics(userProfiles,z,2);
                        }
                        else if((userProfiles.m[z][0] >= "0300") &&
(userProfiles.m[z][1] <= "0400"))
                        {
                                countStatistics(userProfiles,z,3);
                        }
                        else if((userProfiles.m[z][0] >= "0400") &&
(userProfiles.m[z][1] <= "0500"))
```

```
                        {
                                countStatistics(userProfiles,z,4);
                        }
                        else if((userProfiles.m[z][0] >= "0500") &&
(userProfiles.m[z][1] <= "0600"))
                        {
                                countStatistics(userProfiles,z,5);
                        }
                }
                clearVariables();
                for (z = 0; z < observedActivity.m.size(); z++)
                {
                        if((observedActivity.m[z][0] >= "0000") &&
(observedActivity.m[z][1] <= "0100"))
                        {
                                countStatistics(observedActivity,z,0);
                        }
                        else if((observedActivity.m[z][0] >= "0100") &&
(observedActivity.m[z][1] <= "0200"))
                        {
                                countStatistics(observedActivity,z,1);
                        }
                        else if((observedActivity.m[z][0] >= "0200") &&
(observedActivity.m[z][1] <= "0300"))
                        {
                                countStatistics(observedActivity,z,2);
                        }
                        else if((observedActivity.m[z][0] >= "0300") &&
(observedActivity.m[z][1] <= "0400"))
                        {
                                countStatistics(observedActivity,z,3);
                        }
                        else if((observedActivity.m[z][0] >= "0400") &&
(observedActivity.m[z][1] <= "0500"))
                        {
                                countStatistics(observedActivity,z,4);
                        }
                        else if((observedActivity.m[z][0] >= "0500") &&
(observedActivity.m[z][1] <= "0600"))
                        {
                                countStatistics(observedActivity,z,5);
                        }
                }
                clearVariables();
        }
        //Morning period
```

```
	else if (type == "M")
	{
		for (int z = 0; z < userProfiles.m.size(); z++)
		{
			if((userProfiles.m[z][0] >= "0600") && (userProfiles.m[z][1] <=
"0700"))
			{
				countStatistics(userProfiles,z,6);
			}
			else if((userProfiles.m[z][0] >= "0700") &&
(userProfiles.m[z][1] <= "0800"))
			{
				countStatistics(userProfiles,z,7);
			}
			else if((userProfiles.m[z][0] >= "0800") &&
(userProfiles.m[z][1] <= "0900"))
			{
				countStatistics(userProfiles,z,8);
			}
			else if((userProfiles.m[z][0] >= "0900") &&
(userProfiles.m[z][1] <= "1000"))
			{
				countStatistics(userProfiles,z,9);
			}
			else if((userProfiles.m[z][0] >= "1000") &&
(userProfiles.m[z][1] <= "1100"))
			{
				countStatistics(userProfiles,z,10);
			}
		}
		clearVariables();
		for (z = 0; z < observedActivity.m.size(); z++)
		{
			if((observedActivity.m[z][0] >= "0600") &&
(observedActivity.m[z][1] <= "0700"))
			{
				countStatistics(observedActivity,z,6);
			}
			else if((observedActivity.m[z][0] >= "0700") &&
(observedActivity.m[z][1] <= "0800"))
			{
				countStatistics(observedActivity,z,7);
			}
			else if((observedActivity.m[z][0] >= "0800") &&
(observedActivity.m[z][1] <= "0900"))
			{
```

```
                        countStatistics(observedActivity,z,8);
                    }
                    else if((observedActivity.m[z][0] >= "0900") &&
(observedActivity.m[z][1] <= "1000"))
                    {
                        countStatistics(observedActivity,z,9);
                    }
                    else if((observedActivity.m[z][0] >= "1000") &&
(observedActivity.m[z][1] <= "1100"))
                    {
                        countStatistics(observedActivity,z,10);
                    }
                }
                clearVariables();
            }
            //Afternoon period
            else if (type == "A")
            {
                for (int z = 0; z < userProfiles.m.size(); z++)
                {
                    if((userProfiles.m[z][0] >= "1100") && (userProfiles.m[z][1] <=
"1200"))
                    {
                        countStatistics(userProfiles,z,11);
                    }
                    else if((userProfiles.m[z][0] >= "1200") &&
(userProfiles.m[z][1] <= "1300"))
                    {
                        countStatistics(userProfiles,z,12);
                    }
                    else if((userProfiles.m[z][0] >= "1300") &&
(userProfiles.m[z][1] <= "1400"))
                    {
                        countStatistics(userProfiles,z,13);
                    }
                    else if((userProfiles.m[z][0] >= "1400") &&
(userProfiles.m[z][1] <= "1500"))
                    {
                        countStatistics(userProfiles,z,14);
                    }
                    else if((userProfiles.m[z][0] >= "1500") &&
(userProfiles.m[z][1] <= "1600"))
                    {
                        countStatistics(userProfiles,z,15);
                    }
                }
```

```
                clearVariables();
                for (z = 0; z < observedActivity.m.size(); z++)
                {
                        if((observedActivity.m[z][0] >= "1100") &&
(observedActivity.m[z][1] <= "1200"))
                        {
                                countStatistics(observedActivity,z,11);
                        }
                        else if((observedActivity.m[z][0] >= "1200") &&
(observedActivity.m[z][1] <= "1300"))
                        {
                                countStatistics(observedActivity,z,12);
                        }
                        else if((observedActivity.m[z][0] >= "1300") &&
(observedActivity.m[z][1] <= "1400"))
                        {
                                countStatistics(observedActivity,z,13);
                        }
                        else if((observedActivity.m[z][0] >= "1400") &&
(observedActivity.m[z][1] <= "1500"))
                        {
                                countStatistics(observedActivity,z,14);
                        }
                        else if((observedActivity.m[z][0] >= "1500") &&
(observedActivity.m[z][1] <= "1600"))
                        {
                                countStatistics(observedActivity,z,15);
                        }
                }
                clearVariables();
        }
        //Evening period
        else if (type == "E")
        {
                for (int z = 0; z < userProfiles.m.size(); z++)
                {
                        if((userProfiles.m[z][0] >= "1600") && (userProfiles.m[z][1] <=
"1700"))
                        {
                                countStatistics(userProfiles,z,16);
                        }
                        else if((userProfiles.m[z][0] >= "1700") &&
(userProfiles.m[z][1] <= "1800"))
                        {
                                countStatistics(userProfiles,z,17);
                        }
```

```
                        else if((userProfiles.m[z][0] >= "1800") &&
(userProfiles.m[z][1] <= "1900"))
                        {
                                countStatistics(userProfiles,z,18);
                        }
                }
                clearVariables();
                for (z = 0; z < observedActivity.m.size(); z++)
                {
                        if((observedActivity.m[z][0] >= "1600") &&
(observedActivity.m[z][1] <= "1700"))
                        {
                                countStatistics(observedActivity,z,16);
                        }
                        else if((observedActivity.m[z][0] >= "1700") &&
(observedActivity.m[z][1] <= "1800"))
                        {
                                countStatistics(observedActivity,z,17);
                        }
                        else if((observedActivity.m[z][0] >= "1800") &&
(observedActivity.m[z][1] <= "1900"))
                        {
                                countStatistics(observedActivity,z,18);
                        }
                }
                clearVariables();
        }
        //Night period
        else if (type == "N")
        {
                for (int z = 0; z < userProfiles.m.size(); z++)
                {
                        if((userProfiles.m[z][0] >= "1900") && (userProfiles.m[z][1] <=
"2000"))
                        {
                                countStatistics(userProfiles,z,19);
                        }
                        else if((userProfiles.m[z][0] >= "2000") &&
(userProfiles.m[z][1] <= "2100"))
                        {
                                countStatistics(userProfiles,z,20);
                        }
                        else if((userProfiles.m[z][0] >= "2100") &&
(userProfiles.m[z][1] <= "2200"))
                        {
                                countStatistics(userProfiles,z,21);
```

```
                        }
                        else if((userProfiles.m[z][0] >= "2200") &&
(userProfiles.m[z][1] <= "2300"))
                        {
                                countStatistics(userProfiles,z,22);
                        }
                        else if((userProfiles.m[z][0] >= "2300") &&
(userProfiles.m[z][1] <= "2400"))
                        {
                                countStatistics(userProfiles,z,23);
                        }
                }
                clearVariables();
                for (z = 0; z < observedActivity.m.size(); z++)
                {
                        if((observedActivity.m[z][0] >= "1900") &&
(observedActivity.m[z][1] <= "2000"))
                        {
                                countStatistics(observedActivity,z,19);
                        }
                        else if((observedActivity.m[z][0] >= "2000") &&
(observedActivity.m[z][1] <= "2100"))
                        {
                                countStatistics(observedActivity,z,20);
                        }
                        else if((observedActivity.m[z][0] >= "2100") &&
(observedActivity.m[z][1] <= "2200"))
                        {
                                countStatistics(observedActivity,z,21);
                        }
                        else if((observedActivity.m[z][0] >= "2200") &&
(observedActivity.m[z][1] <= "2300"))
                        {
                                countStatistics(observedActivity,z,22);
                        }
                        else if((observedActivity.m[z][0] >= "2300") &&
(observedActivity.m[z][1] <= "2400"))
                        {
                                countStatistics(observedActivity,z,23);
                        }
                }
                clearVariables();
        }
        else if (type == "DTIME")
        {
                //Activity already counted above
```

```
        }
        else if (type == "DHOUR")
        {
                //Activity already counted above
        }

}

//ACTIVITYHANDLER CLASS
ACTIVITYHANDLER::ACTIVITYHANDLER()
{
        //Observed activity logs
   observedLogA = "observedLogA.csv";
        observedLogB = "observedLogB.csv";
        observedLogS = "observedLogS.csv";
        observedLogM = "observedLogM.csv";

        //Initialize the vector that holds all monitor activity
        vector<string> twoDim(10);
   vector<vector<string> > vec(20, twoDim);
        v = vec;
}

//Initialize observed activity
void ACTIVITYHANDLER::getObservedActivity(char * file)
{
        //Local variables
        int i = 0, j = 0;

        //Each row of data
        string line;

        //Open the infile
        ifstream in(file, ios::in);

        //Make sure that it was opened correctly
        if (!in.is_open())
   {
                cout << "FAILURE" << endl;
                exit(1);
   }

        //Read in each line of the infile and populate the vector
   while(getline(in,line,','))
   {
```

```
            if (i == 8)
            {
                    v[j][i] = line;
                    i++;
                    getline(in,line);
                    v[j][i] = line;
                    i++;
            }
            if (i == 10)
            {
                            i = 0;
                            j++;
                            getline(in,line,',');
                            v[j][i] = line;
                            i++;
            }
            else
            {
                            v[j][i] = line;
                            i++;
            }
    }
        in.close();
}

//Write observed activity to logs
void ACTIVITYHANDLER::writeObservedActivityLog(string user, int count)
{
        char * file;

        //Determine log
        if (user == "A")
                file = observedLogA;
        else if (user == "B")
                file = observedLogB;
        else if (user == "S")
                file = observedLogS;
        else if (user == "M")
                file = observedLogM;

        //Open the outfile
        ofstream out(file, ios::app);

        //Make sure that it was opened correctly
        if (!out.is_open())
    {
```

```
                cout << "FAILURE" << endl;
                exit(1);
    }

        for (int k = 0; k < v[count].size(); k++)
        {
                out << v[count][k];
        if (k != 9)
                        out << ",";
        }
        out << endl;

        out.close();
}

void ACTIVITYHANDLER::createObservedActivityLogs()
{
        //Initialize the vector that holds userA observed activity log
        vector<string> aTwoDim(10);
    vector<vector<string> > avec(20, aTwoDim);
        cout << "Creating User A Observed Activity Log.....";
        a = avec;
        cout << "OK" << endl;

        //Initialize the vector that holds userB observed activity log
        vector<string> bTwoDim(10);
    vector<vector<string> > bvec(20, bTwoDim);
        cout << "Creating User B Observed Activity Log.....";
        b = bvec;
        cout << "OK" << endl;

        //Initialize the vector that holds userS observed activity log
        vector<string> sTwoDim(10);
    vector<vector<string> > svec(20, sTwoDim);
        cout << "Creating User S Observed Activity Log.....";
        s = svec;
        cout << "OK" << endl;

        //Initialize the vector that holds userM observed activity log
        vector<string> mTwoDim(10);
    vector<vector<string> > mvec(20, mTwoDim);
        cout << "Creating User M Observed Activity Log.....";
        m = mvec;
        cout << "OK" << endl;
}
```

```
//Output all monitor activity to the screen
void ACTIVITYHANDLER::printMonitorVector()
{
	for (int z = 0; z < v.size(); z++)
	{
		for (int k = 0; k < v[z].size(); k++)
		{
		  cout << v[z][k] << " ";
		}
		cout << endl;
	}
}

void ACTIVITYHANDLER::clearObservedLogs()
{
	//Clear User Observed Activity Logs
	ofstream clearA("observedLogA.csv", ios::trunc);
	ofstream clearB("observedLogB.csv", ios::trunc);
	ofstream clearS("observedLogS.csv", ios::trunc);
	ofstream clearM("observedLogM.csv", ios::trunc);

	//Close files
	clearA.close();
	clearB.close();
	clearS.close();
	clearM.close();
}

//Output all user observed log activity to the screen
void ACTIVITYHANDLER::printObservedLogs()
{
	for (int z = 0; z < a.size(); z++)
	{
		for (int k = 0; k < a[z].size(); k++)
		{
		  cout << a[z][k] << " ";
		}
		cout << endl;
	}

	for (z = 0; z < b.size(); z++)
	{
		for (int k = 0; k < b[z].size(); k++)
		{
		  cout << b[z][k] << " ";
		}
```

```
                cout << endl;
        }

        for (z = 0; z < s.size(); z++)
        {
                for (int k = 0; k < s[z].size(); k++)
                {
                  cout << s[z][k] << " ";
                }
                cout << endl;
        }

        for (z = 0; z < m.size(); z++)
        {
                for (int k = 0; k < m[z].size(); k++)
                {
                  cout << m[z][k] << " ";
                }
                cout << endl;
        }
}

//Return monitor activity
vector<vector<string> > ACTIVITYHANDLER::getMonitorActivity()
{
        return v;
}

//PROFILEHANDLER CLASS
PROFILEHANDLER::PROFILEHANDLER()
{
        //Initialize the vector that holds userA normal profile data
        vector<string> aTwoDim(10);
   vector<vector<string> > avec(20, aTwoDim);
        a = avec;
        //Initialize the vector that holds userB normal profile data
        vector<string> bTwoDim(10);
   vector<vector<string> > bvec(20, bTwoDim);
        b = bvec;
        //Initialize the vector that holds userS normal profile data
        vector<string> sTwoDim(10);
   vector<vector<string> > svec(20, sTwoDim);
        s = svec;
        //Initialize the vector that holds userM normal profile data
        vector<string> mTwoDim(10);
   vector<vector<string> > mvec(20, mTwoDim);
```

```
        m = mvec;
}

void PROFILEHANDLER::loadUserProfile(char * file)
{
        //Temporary container for normal profile data
        vector<string> tTwoDim(10);
    vector<vector<string> > v(20, tTwoDim);

        //Local variables
        int i = 0, j = 0;

        //Each row of data
        string line;

        //Open the infile
        ifstream in(file, ios::in);

        //Make sure that it was opened correctly
        if (!in.is_open())
    {
                cout << "FAILURE" << endl;
                exit(1);
    }

        //Read in each line of the infile and populate the vector
    while(getline(in,line,','))
    {

          if (i == 8)
          {
                  v[j][i] = line;
                  i++;
                  getline(in,line);
                  v[j][i] = line;
                  i++;
          }
          if (i == 10)
          {
                        i = 0;
                        j++;
                        getline(in,line,',');
                        v[j][i] = line;
                        i++;
          }
          else
```

```
            {
                        v[j][i] = line;
                        i++;
            }
    }

        //Determine user
        if (file == "profileLogA.csv")
                a = v;
        else if(file == "profileLogB.csv")
                b = v;
        else if(file == "profileLogS.csv")
                s = v;
        else
                m = v;

        cout << "OK" << endl;
        in.close();
}

//Output all user normal profile log activity to the screen
void PROFILEHANDLER::printNormalProfileLogs()
{
        for (int z = 0; z < a.size(); z++)
        {
                for (int k = 0; k < a[z].size(); k++)
                {
                  cout << a[z][k] << " ";
                }
                cout << endl;
        }

        for (z = 0; z < b.size(); z++)
        {
                for (int k = 0; k < b[z].size(); k++)
                {
                  cout << b[z][k] << " ";
                }
                cout << endl;
        }

        for (z = 0; z < s.size(); z++)
        {
                for (int k = 0; k < s[z].size(); k++)
                {
                  cout << s[z][k] << " ";
```

```
                }
                cout << endl;
        }

        for (z = 0; z < m.size(); z++)
        {
                for (int k = 0; k < m[z].size(); k++)
                {
                   cout << m[z][k] << " ";
                }
                cout << endl;
        }
}

int main()
{
        //Create instance of BIDE class
        BIDE run = BIDE();

        run.printHeader();

        cout << "Intrusion Detection Engine Starting Up.....OK" << endl;
        ACTIVITYHANDLER observedActivity = run.startActivityHandler();

        PROFILEHANDLER userProfiles = run.startProfileHandler();

        //observedActivity.printMonitorVector();

        //userProfiles.printNormalProfileLogs();

    observedActivity.clearObservedLogs();

        run.startSimulation(observedActivity,userProfiles);

        //observedActivity.printObservedLogs();

        //run.printStatistics();

        return 0;
}
```

REFERENCES

[1] Anderson, Frivold, Valdes, “Next-Generation Intrusion Detection Expert System (NIDES) A Summary”, In http://www.sdl.sri.com/papers/4/s/4sri/4sri.pdf, Computer Science Laboratory, SRI International, May 1995.

[2] Barbara, Couto, Jajodia, Popyack, Wu, "ADAM: Detecting Intrusions by Data Mining", In Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, West Point NY, June 2001, pp. 11-16.

[3] Denning, "An Intrusion-Detection Model", In Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, April 1986, pp. 118-132.

[4] General Accounting Office, “Information Security: Computer Attacks at Department of Defense Pose Increasing Risks”, In GAO/AIMD-96-84, May 1996.

[5] Ghosh, Wanken, Charron, "Detecting Anomalous and Unknown Intrusions Against Programs", In Proceedings of the 1998 Computer Security Applications Conference, IEEE CS Press, Los Alamitos CA, 1998, pp. 259–267.

[6] Goregaoker, “A Method for Detecting Intrusions on Encrypted Traffic”, In Technical Report TR-010703, Computer Science Department, Florida State University, 2001.

[7] Hallivuori, Kousa, “Denial of Service Attack against SSH Key Exchange”, Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, November 2001.

[8] Ilgun, “USTAT: A Real-Time Intrusion Detection System for UNIX”, In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland CA, May 1993, pp. 16-28.

[9] Jones, Li, “Temporal Signatures for Intrusion Detection”, In Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans LA, December 2001, pp. 252-261.

[10]Kohout, Yasinsac, McDuffie, "Activity Profiles for Intrusion Detection", In Proceedings of the North American Fuzzy Information Processing Society-Fuzzy Logic and the Internet (NAFIPS-FLINT 2002), New Orleans LA, June 2002.

[11]Lane, Brodley, "Temporal Sequence Learning and Data Reduction for Anomaly Detection", In Proceedings of ACM Transactions on Information and System Security, Vol. 2, No. 3, August 1999, pp 295-331.

[12]Lee, Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems", In Proceedings of ACM Transactions on Information and System Security, November 2000, pp. 227-261.

[13]Lee, Stolfo, “Data Mining Approaches for Intrusion Detection”, In Proceedings of the 7th USENIX Security Symposium”, San Antonio TX, January 1998, pp. 26-29.

[14]Lindqvist, Porras, “Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)”, In Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland CA, May 1999, pp. 146-161.

[15]Lowe, “Some New Attacks upon Security Protocols", In Proceedings of the 9th IEEE Computer Security Foundations Workshop, March 1996, pp. 162-169.

[16]Lunt, Jagannathan, “A Prototype Real-Time Intrusion-Detection Expert System”, In Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, Oakland CA, April 1988, pp. 59-66.

[17]Melendez, “The Monitor and Principals”, In Technical Report TR-010701, Computer Science Department, Florida State University, 2001.

[18]Patel, “Knowledge Base for Intrusion Detection System”, In Technical Report TR-011203, Computer Science Department, Florida State University, 2001.

[19]Porras, Neumann, “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances”, In Proceedings of the National Information Systems Security Conference, October 1997, pp. 353-365.

[20]Presidential Commission on Critical Infrastructure Protection. Commission Report “Critical Foundations: Protecting America’s Infrastructures”, In http://www.pccip.gov/report_index.html, October 1997.

[21]Syverson, “A Taxonomy of Replay Attacks”, In Proceedings of the 7th Computer Security Foundations Workshop, June 1994, pp. 131-136.

[22]Vigna, Kemmerer, “NetStat: A Network-Based Intrusion Detection Approach”, In Proceedings of the 14th Annual Information Theory: 50 Years of Discovery Computer Security Application Conference, Scottsdale AZ, December 1998, pp. 25-34.

[23]Wagner, Schneier, “Analysis of SSL 3.0 Protocol”, In Proceedings of the 2nd USENIX Workshop on Electronic Commerce, Oakland CA, November 1996, pp. 29-40.

[24]Woo, Lam, “Authentication for Distributed Systems”, In Computer, Vol. 25 No. 1, January 1992, pp. 39-52.

[25]Yasinsac, "An Environment for Security Protocol Intrusion Detection", In The Journal of Computer Security, Vol. 10, No. 1-2, January 2002, pp. 177-188.

[26]Ye, Chen, “An Anomaly Detection Technique Based on a Chi-Square Statistic for Detecting Intrusions into Information Systems”, In Proceedings of Quality and Reliability Engineering International, Vol. 17, No. 2, 2001, pp. 105 - 112.

BIOGRAPHICAL SKETCH

Tysen Leckie is a graduate student at Florida State University working on the Master of Science in Computer Science degree. He is a member of the ACM, IEEE and IEEE Computer Society.