5-1-2013

Simulation and Analysis of Insider Attacks

Christopher Blake Clark
University of Nevada, Las Vegas, cla05019@gmail.com

SIMULATION AND ANALYSIS

OF INSIDER ATTACKS

by

Christopher Blake Clark

Bachelor of Science in Computer Science
Brigham Young University, Idaho
2011

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2013

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Christopher Blake Clark

entitled

Simulation and Analysis of Insider Attacks

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

May 2013

Abstract

An insider is an individual (usually an employee, contractor, or business partner) that has been trusted with access to an organization's systems and sensitive data for legitimate purposes. A malicious insider abuses this access in a way that negatively impacts the company, such as exposing, modifying, or defacing software and data.

Many algorithms, strategies, and analyses have been developed with the intent of detecting and/or preventing insider attacks. In an academic setting, these tools and approaches show great promise. To be sure of their effectiveness, however, these analyses need to be tested. While real data is available on insider attacks (including logs of actions taken by the insider), the real data is limited in its usefulness. If the analysis being tested passes or fails in detecting the insider attack, how much can be attributed to the analysis's precision, the circumstances of the attack, or just luck? The ability to test an analysis against a wide range of data with circumstances that vary in complexity and circumstance would allow insight into strengths and weaknesses of the analysis. Data for multiples tests would also help in ruling out luck in the results.

To address this, I've built an insider attack simulator that generates test scenarios for analyses. Specifically, it generates logs of employee actions with both insider attacks and false positives hidden within the logs. This simulator allows for customization of the actions that are logged, the average behavior of individuals, the departments within the simulated company, and the abnormal events (including insider attacks) that take place. This thesis will discuss the nature of insider threats, the benefits of a simulator, how to customize the simulation, and how one can gain insight into analyses using logs generated by the simulator.

Table of Contents

List of Tables

List of Figures

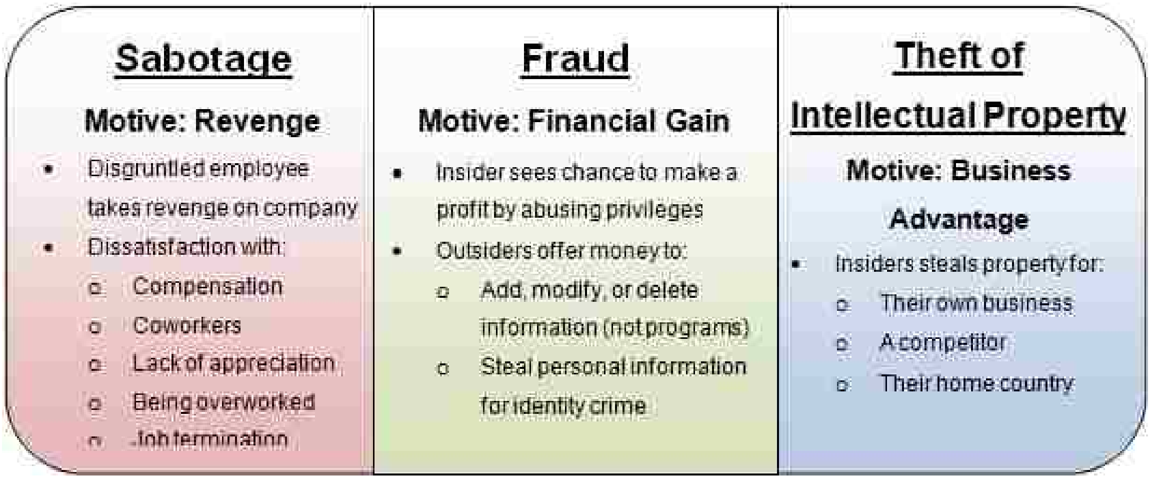
Chapter 1

Introduction

An insider is an individual (usually an employee, contractor, or business partner) that has been trusted with access to an organization's systems and sensitive data for legitimate purposes. A malicious insider abuses this access in a way that negatively impacts the company, such as exposing, modifying, or defacing software and data. This malicious insider is referred to as an insider threat, while the insider's actions are called insider attacks. In contrast to external attacks, where individuals try to gain access to the organization by illegal means, insider threats already have access to the organization and authority to use that access. In addition, many insiders have familiarity with the organization's security weaknesses. As a result, insiders are often able to evade and disable security, destroy backups, and do great damage without raising alarms.

1.1. Motivations behind Insider Attacks

Why would an individual attack their own company? Cappelli, Moore, Trzeciak and Shimeall (2009) suggest that insider attacks fall into three motivation-based categories: sabotage, fraud, and theft of intellectual property. Figure 1 depicts these three categories:



Sabotage

Motive: Revenge

- Disgruntled employee takes revenge on company
- Dissatisfaction with:
  - Compensation
  - Coworkers
  - Lack of appreciation
  - Being overworked
  - Job termination

Fraud

Motive: Financial Gain

- Insider sees chance to make a profit by abusing privileges
- Outsiders offer money to:
  - Add, modify, or delete information (not programs)
  - Steal personal information for identity crime

Theft of Intellectual Property

Motive: Business Advantage

- Insiders steals property for:
  - Their own business
  - A competitor
  - Their home country

Figure 1: Three categories of attacks and motives.

Sabotage is often caused by employees who become disaffected with their company, boss, or coworkers. Many occurrences in the work place can set an employee against their company and individuals within it, including dissatisfaction with compensation, arguments with coworkers, reprimands, or job termination. Further estrangement can occur when employees feel stressed, overworked, underappreciated, or isolated. Under such circumstances, an employee will often feel unfairly treated and may perform an insider attack out of anger or revenge. The main objective of these disaffected employees is to cause as much destruction as possible to those that they feel have wronged them. Cappelli et al. (2009) suggest that sabotage is often performed by employees with high technical skills and access to sensitive assets, including engineers and network administrators. These employees are often intimately aware of the systems they intend to sabotage, and their advanced technical ability gives them the means and know-how to cause incredible damage.

However, dissatisfaction is only one of the motivations behind insider attacks. Other insiders see an insider attack as an opportunity for financial gain (Cappelli et al., 2009). In many instances, these insiders are approached by individuals outside the company and offered large sums of money to steal personal information from the company for identity crimes. In other cases, they're asked to add, modify or delete company information. These initial offers may seem comparatively harmless to an employee, and the cash is enough incentive to cause the employee to rationalize their actions. These attacks usually don't happen once, but occur multiple times with each new attack growing greater in risk and reward. Sometimes this is against the employee's desires; in some circumstances, employees that perform these attacks for outsiders are later coerced into performing additional, more damaging attacks because the outsider threatens to turn them in or cause other serious problems if they don't comply. Other times, the successful attacks embolden the employee, and the employee's thefts become more brazen and damaging as they seek greater financial gain.

Money isn't the only prize sought by insiders; many insiders steal proprietary information from the company to gain business advantage for themselves, their family, a foreign country, or a company they intend to work for (Cappelli, Moore, Trzeciak & Shimeall, 2009). Individuals within the company may be offered large sums of money and future employment by those outside the company if they will pass along the company's proprietary information, including research and development, client lists, quotes, passwords, and software. Others steal these company assets and start their own business with them, often under the rationalization that they helped to develop the assets the stole, and therefore they have a right to take it with them.

As one may have already surmised, attacks can be part of one, many, or none of the categories above. Other motivations behind insider attacks include curiosity and bragging rights. Whatever the reason, motivations exist for insider attacks in any type of company.

1.2. Damage of Insider Attacks

Insider attacks have proven to be devastating to organizations, and many have caused damage so irreparable that affected companies are forced out of business. Insiders have two particular advantages when attacking their company that external attackers do not. First, insiders have a greater knowledge of the sensitive systems and data that they're attacking, including weaknesses in the security protecting those sensitive assets. With such knowledge, not only can insiders attack the company's system and data, but they can also target back-up systems and cripple the company's means of recovering from attacks.

Second, since insiders already have the authority and privileges to access targeted systems and data, their attacks are far more likely to go undetected. If performed discretely, the malicious actions of the insider may not be easily distinguishable from day-to-day activities. And again, since the insider has a greater awareness of the security measures in place to monitor and protect the data, the insider can disable the alerts, logs, and other means of recording or identifying attacks.

1.3. Can insider threats be thwarted?

Insider attacks can and have been thwarted, but it often takes greater planning, persistence, and effort to combat than fighting external attacks. Upcoming attacks can be detected by noting that a malicious insider's behavior before and during the attack differs from their prior behavior. Sometimes the changes in behavior are directly related to the attack itself; an insider that is emailing sensitive information to an outsider or to their personal email, for example, will have an increase in the number of emails sent outside the company with attachments. Another employee may have a sharp decrease in the number of emails they handle as their focus move away from legitimate work to the attack.

The attacks themselves may not involve a notable change in behavior; an employee may open a file they edit on a daily basis, but make malicious modifications instead. However, in cases where the attack itself involves little change in behavior, insiders on the path to committing an attack still display changes in behavior that indicate something is amiss. These changes may be subtle or even subconscious, and can include changes in:

- Productivity
- The amount of communication with those outside the company
- The attitude of the employee
- The time of day of the behavior.

By noting changes in behavior, the company may be able to recognize and prevent upcoming insider attacks. However, to recognize abnormal behavior, one must first know what the insider's normal behavior is. The normal actions of the insider need to be recorded over an extended period of time; it's for this purpose that companies keep logs of employee actions. Once sufficient data has been collected, the insider's current behavior can be compared to their earlier behavior, and the abnormality of current actions can be measured.

1.4. Logs and Analysis Programs

Many means of monitoring and recording the actions of insiders into logs have already been developed and put in use. By logging the actions of employees, a company has a record of the employee's earlier normal behavior, their newer abnormal behavior, and the actions taken to perform the insider attack itself. However, such data is hidden among the huge quantities of data generated by employee actions each day. Unless specific clues are given as to where to search, manually browsing a log of actions in search of abnormal behavior is far from effective. The solution is to create a program that automates the search for insider attacks within logs.

A few problems make this automation difficult. The data quantity problem is one of these. Specifically, the sheer quantity of data logged during an employee's stay with the company is far too big to be kept in active memory by the analysis program. As such, the analysis program needs to be able to read through the log in small portions, consolidating by keeping what's important for the analysis (whether that's the actual data, a summary of the data, or values it computes as it reads), and discarding the rest. If done correctly, the analysis will be able to perform its function and identify the insider attacks without having to perform multiple passes over the log.

Many algorithms, strategies, and analyses have been developed with the intent of detecting and/or preventing insider attacks. In an academic setting, these tools and approaches show great promise. To be sure of their effectiveness, however, these analyses need to be tested. While real data is available on insider attacks (including logs of actions taken by the insider), the real data is limited in its usefulness. If the analysis being tested passes or fails in detecting the insider attack, how much can be attributed to the analysis's precision, the circumstances of the attack, or just luck? The ability to test an analysis against a wide range of data with varying degrees of complexity is needed to gain this insight.

1.5. Simulator Overview

For this thesis, I build a program that simulates the behavior of employees, both good and malicious, generating a log of actions that stretches over several days. This log can then be used as input to the many algorithms and programs designed to detect insider threats. Many insights into the strengths and weaknesses of these tools can be gained as a result, including:

- What types of insider attacks did the analysis detect?
- What percent of attacks were discovered?
- What kinds of insider attacks did it fail to detect?
- Did the analysis trigger any false positives?
- Would a change in circumstance result in remarkably different results? In other words, how situation-dependent is the success of the analysis?

With my program's simulator, a large variety of logs can be generated to explore these questions. First, it simulates the behavior of a company's employees (both malicious and innocent) to generate a log of employee activity. It also keeps a separate log of when the insider attacks and false positives take place. Upon completion of the log, analyses can be run with the generated log as input. The results of each analysis can then be compared to the log of insider attacks to see how successful the analysis performed.

This process can be broken down into two phases: a simulation phase and an analysis phase. The steps to these two phases are shown in figure 2:

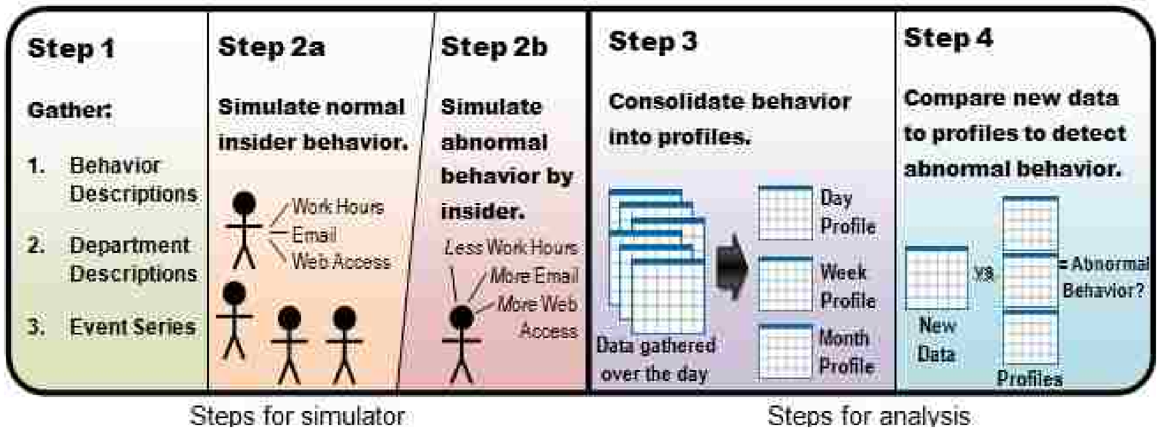


Figure 2: Steps for simulating and detecting insider attacks.

The first step to running the simulator is to determine the actions, behavior, and abnormal events (potentially representing malicious attacks) that the program will be simulating. Once this is complete, normal and abnormal behavior are simulated simultaneously. More specifically, normal behavior is generated until an abnormal event occurs. The encountered event is resolved which, depending on the settings of the event, could involve specific actions taking place or the behavior of an employee being modified. Once this has occurred, normal behavior continues until the next event, which is then resolved. This is repeated until all of the events have taken place and normal behavior has been generated up to the end of the simulation.

Upon completion of the simulation, a large log of actions performed by the employees will have been compiled. All actions, both normal and malicious, are recorded within this log, and it is now the job of the selected analysis to process that log and find malicious behavior. In the case of both example analyses included with the simulator (each based off of Kim, Sheldon, and Hively's profile analysis (2012)), the analysis performs a single sweep over the log, consolidating employee behavior into day, week, and month profiles. Simultaneously, the analysis compares each new day profile against the employee’s previous day, week, and month profiles. It generates an anomaly score indicating how much the behavior in that day’s profile differs from the behavior in the previous profiles. It records these anomaly scores in an output file, which can then be assessed by looking for the large anomaly scores or other patterns that indicate an insider attack. This assessment can be done automatically by an external program or it can be done visually by graphing the anomaly scores (using a program such as excel) to give a visual indication of where the abnormal behavior and insider attacks occurred.

The steps for simulation will be covered in detail over chapters 3-6 and the two examples analyses will be explained in chapter 7. A demonstration of how the anomaly scores can be assessed visually with graphs will be given in chapter 8.

Chapter 2

Previous Works

Kim, Sheldon, and Hively (2012) proposed a way to quantify insider threats by compiling daily actions of an employee into day, week, and month profiles. Each new day profile for an employee is compared to the previous day, week, and month profiles to detect abnormal behavior. Their analysis worked on the theory that the behavior of a malicious insider changes leading up to, during, and after an attack. Dissatisfaction, lower productivity, and other signs of discontentment often precede the attack, which can be detected early to prevent the insider attack. In addition, insiders often do preliminary abnormal behavior before the attack, such as investigating security or the target of the attack.

A few insiders may take steps to maintain normal behavior in an effort to cover up the attack, but abnormal behavior can still be detected by close inspection of a variety of daily actions; for instance, the insider may consciously make sure their email and work output remains steady, but neglect to correct their changes in behavior in regards to phone calls, idle time, or websites visited. Further, the attacks themselves often involve behavior that differs from daily, weekly, and monthly behavior; having the insider's prior behavior on hand allows for detection of this abnormal behavior by comparing new behavior to the employee's earlier behavior.

Kim, Sheldon, and Hively's insider threat analysis (2012) served as my introduction to insider threats, and as a result much of the theory and terminology of this thesis is heavily influenced by their paper. With permission from Yoohwan Kim, I have programmed and included two variations of their insider threat analysis. Chapter 7 discusses the implementation of these analyses.

Like Kim et al., Zhang and Zhao (2010) also elected to detect insider threats through behavior monitoring. In both Kim et al. and Zhang and Zhao's papers, they monitored behavior and created behavior models for the employees. However, Zhang and Zhao came up with a different approach to consolidating data. They explained that the sheer quantity of data involved in behavior monitoring requires some means of reducing the data's size, and they proposed using rough set theory along with machine learning to overcome this obstacle. The data size difficulties mentioned by Zhang et al. were encountered during my thesis as well, since the logs produced by even a small number of employees grew rapidly as the days progressed.

Hongbin Zhang (2009) divides insider attacks into low, medium, and high-level threats. Potential risks fall in the low-level category, abnormal (but not necessarily malicious) activities constitute medium-level threats, and malicious activities are labeled as high-level threats. Zhang then designs a defense model where each threat-level is addressed by different modules. Lower-level modules handle lower level threats, and pass higher-level threats up the higher modules. This allowed the modules to filter the threats so that the appropriate level of investigation, processing and action could be applied to each threat.

Killourhy and Maxion (2009) noted the difficulty in obtaining useful, real-world data of insider attacks, pointing out that privacy and security can prevent companies from handing over data showing how the insider successfully attacked them. The companies that do allow access to their logs and data usually sanitize the data first so that sensitive information is covered up or replaced with tokens. Killourhy and Maxion's paper addressed how different methods of sanitizing data can cause the sanitized data to be misleading. While the data simulated by my program won't provide greater accuracy than sanitized data, it does address the difficulty of obtaining data by allowing for the creation of a variety of insider attacks. Killourhy and Maxion's cautions about the accuracy of sanitized data also applies to my program's simulated data. While the simulated data is designed to provide insights, the artificial nature of simulated data, like sanitized data, can cause misleading interpretations if the differences between it and real-world data aren't considered.

Liu, Martin, Hetherington, and Matzner (2005) also pointed out the difficulty in getting real world data, and suggested creating datasets. They also proposed that when simulating insider threats, choose certain files to be "sensitive". Extra accesses to these sensitive files should be viewed with suspicion. They also notes the difficulties in avoiding false positives, particularly when an employee is sick, goes on vacation, or experiences a change in his or her job role.

Montelibano and Moore (2012) provide great detail on different forms of insider attacks. They claimed that if adequate controls are implemented for authorized access, acceptable use, and continuous monitoring, most insider attacks could be detected and prevented. No aspect of the company should be left out; the controls should be applied at the business, information, data, and application layers of company security.

Ali, Shaikh, and Shaikh (2008) used profiles to keep track of the suspicious behaviors of insiders. When an insider acts suspiciously, their profile is marked and watched more closely. Ali et al. (2008) provided great detail about different activities that are suspicious, including unusual login times and locations, unauthorized access attempts, installation of external drives or software, and abnormal idle time.

Cappelli, Moore, Trzeciak and Shimeall (2009) studied 250 insider threat cases and detailed the trends behind each insider attack, including motivation and planning. Their breakdown of insider attacks into the three categories of sabotage, financial gain, and business advantage was shown in figure 1 and described in chapter 1. They provided several examples of each type of insider attack, and concluded that all employees, not just technically advanced employees or employees with administrator privileges, have the capability and motivation to perform attacks.

Pramanik, Sankaranarayanan, and Upadhyaya (2004) discussed the Principle of Least Privilege, where documents are encrypted and the individuals that are allowed to decrypt them are set beforehand. However, they pointed out that least privilege isn’t sufficient security against the insider threat, nor is simply observing the insider’s actions. Instead, both actions and the context of those actions need to be monitored. Managers and coworkers that report the insider’s abnormal behavior can greatly assist any technical monitoring and help in identifying insider attacks.

Chapter 3

Defining Insider Actions and Behaviors

Before performing any simulation, information on what to simulate must first be gathered. The goals of this step are twofold: provide flexibility so that a wide variety of behaviors, companies, and insider attacks can be simulated, but also keep the program simple by making all but the most essential input optional. Only the most essential of details are required to get the simulator running, but the optional input allows for more realistic, useful scenarios to be logged. Each of the following sections will detail what bare minimum data is needed, as well as what optional data can be included to refine the simulator's results.

3.1. Actions and Behavior

3.1.1. Definitions

The user can specify the actions that are to be tracked in the simulator, as well as how often those actions should take place in a typical day. An action, as understood by the simulator, is an activity performed by one person that occurred at a specific time. Actions performed by multiple people are broken down into separate actions for each individual.

Behavior describes the trends in how often the action is performed in an eight-hour day, including the mean occurrences and the day-to-day deviation from the mean. Group behavior, such as department and company behavior, describes the typical behavior of an employee within the group, not the sum of behavior of all members within the group.

3.1.2. Required Data

Each behavior is defined using the following:

- Action Name
    - The name of the action type, i.e. “Email” or “Phone Call”
- Average
    - The mean number of times an employee performs the action in an eight-hour day
- The standard deviation (STD)
    - The daily variance or dispersion of the number of occurrences

The latter two values are used during the simulation to determine the number of times an employee performs the action. Each simulated day, the simulator calculates the number of actions performed by selecting a random, Gaussian-distributed number based on the average and standard deviation, as shown in figure 3.

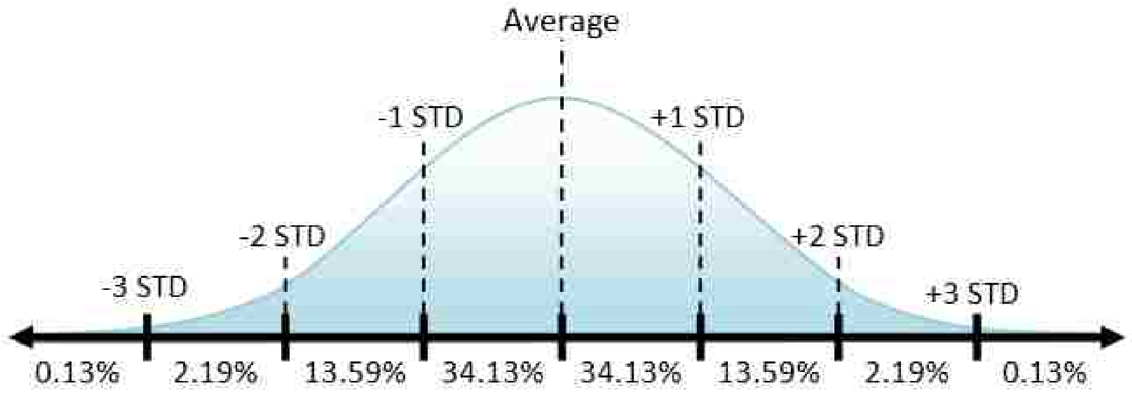


Figure 3: Sketch of a Gaussian function (bell curve).

The percentages below each section in figure 3 show the percentage chance that the Gaussian-distributed random number will be chosen from each section. This means a random, Gaussian-distributed number will fall within one standard deviation of the average 68% of the time, within two standard deviations 95% of the time, and within three standard deviations 99% of the time. As a result, the daily behavior in the simulator is fairly consistent (staying around the mean) while also having some fluctuation (measured by the standard deviation), much like many forms of real behavior.

The reason Gaussian-distributed random numbers are chosen over the more common linear-distributed random numbers is because the former better resembles real-world behavior. The number of occurrences of an action in a day is rarely a random, linear-distributed number between some minimum and maximum. Further, a behavior whose daily occurrence is best described by completely random numbers is too unpredictable for use in determining normal and abnormal patterns of behavior. Instead, using the Gaussian function generates a pattern of behavior where only occasionally there will be an abnormally large or small number of actions.

3.1.3. Company and Department-Level Behavior

To minimize the amount of repetitious data entered to summarize behavior, general behavior for the company is gathered first, and then greater detail can be provided at the department-level. In other words, the company behavior serves as the default behavior for departments. The program has the user first input each company behavior as describe above, where each behavior is given a name, a default average, and a default standard deviation. Once finished, the department behavior can then be specified with one of three options:

- Set the department behavior equal to the company behavior (this is done by default unless otherwise specified).
- Set the department behavior equal to a modified version of the company behavior.

  - To be specific, the department behavior is set equal to the company behavior, and then modifications specified by the user are applied to the department behavior.
  - For example, the sales department can be set to have twice the average number of email actions as the rest of the company.
- Use completely new values for the department behavior.

Figure 4 shows an example of how two departments use the company behavior to determine their own behavior:

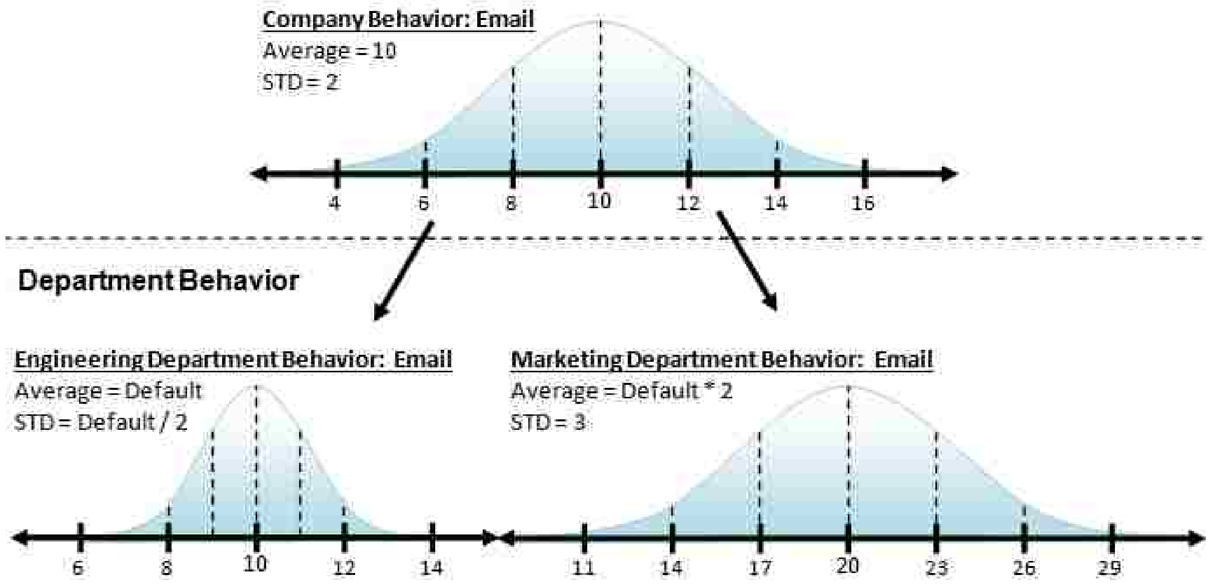


Figure 4: Example of department-level behavior being set based on the company behavior.

When setting the department-level behavior, the default value will be used unless (a) modifications are specified (as can be seen for the engineering department's STD and marketing department's average) or (b) a new, overriding value is given (as shown for the marketing department's STD).

There are two major benefits to defining departments based on company defaults. The first is convenience – the user doesn't have to enter department-level data except where it actually matters to them. The second benefit is that company-level changes in behavior won't require changing all of the departments; by making a change to the company behavior, the department behavior will be likewise changed (with any modifications applied to the new company behavior). However, if these benefits do not appeal, then a user can still override all behavior at the department-level.

3.1.4. Employee-Level Behavior

Employee behavior is handled differently than company or department behavior. Particularly, the user doesn't input each employee's behavior. Doing so would be exhaustive and time-consuming for companies with many employees, and it would result in employees having the exact same behavior for every single test. Instead, each employee behavior is randomly generated at the beginning of each simulation based on the department behavior and two variables: a standard deviation for randomly setting the average, and a standard deviation for randomly setting the standard deviation. To clarify, Figure 5 shows how a standard deviation is used to generate the employee's behavior's average:

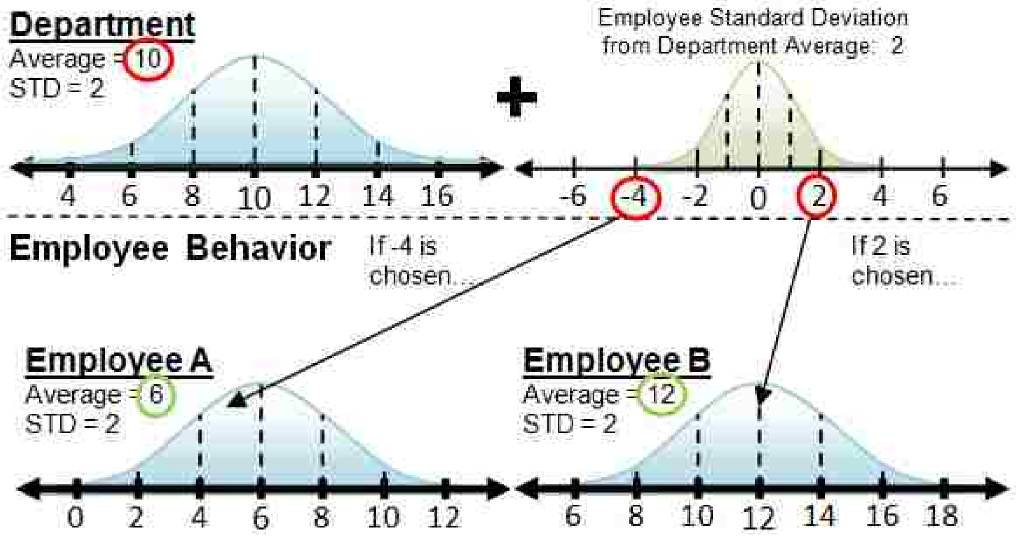


Figure 5: Illustration of how employee-level averages are generated.

When defining each department behavior, the user can specify the standard deviation of employee averages from the department average. If it not specified, the default standard deviation of 0 is used. For each behavior, the employee's average is set equal to the department average plus a random Gaussian-distributed number with an average of 0 and the previously mentioned standard deviation.

Setting this standard deviation for employee averages allows for the user to specify how similar or diverse employee behavior is. Should no such diversity be desired, the default standard deviation of 0 can be used, which would result in no variation in behavior between employees within the same department. In other words, all department employees will have averages matching the department's average. It should be noted that the average is prevented from going below 0, since an employee cannot perform an action a negative number of times.

Like the average for each employee behavior, the standard deviation of each employee behavior is set by adding a random Gaussian-distributed number to the department standard deviation. Figure 6 demonstrates how each employee behavior's standard deviation is set.

The standard deviation of each employee-level behavior starts the same as their departments. A random Gaussian-distributed number is then added to give each employee a unique standard deviation. The user can set the standard deviation of this random Gaussian-distributed number by setting the "Employee Standard Deviation from Department STD"; otherwise, the default value of 0 will be used.

To summarize, each department behavior has three standard deviations:

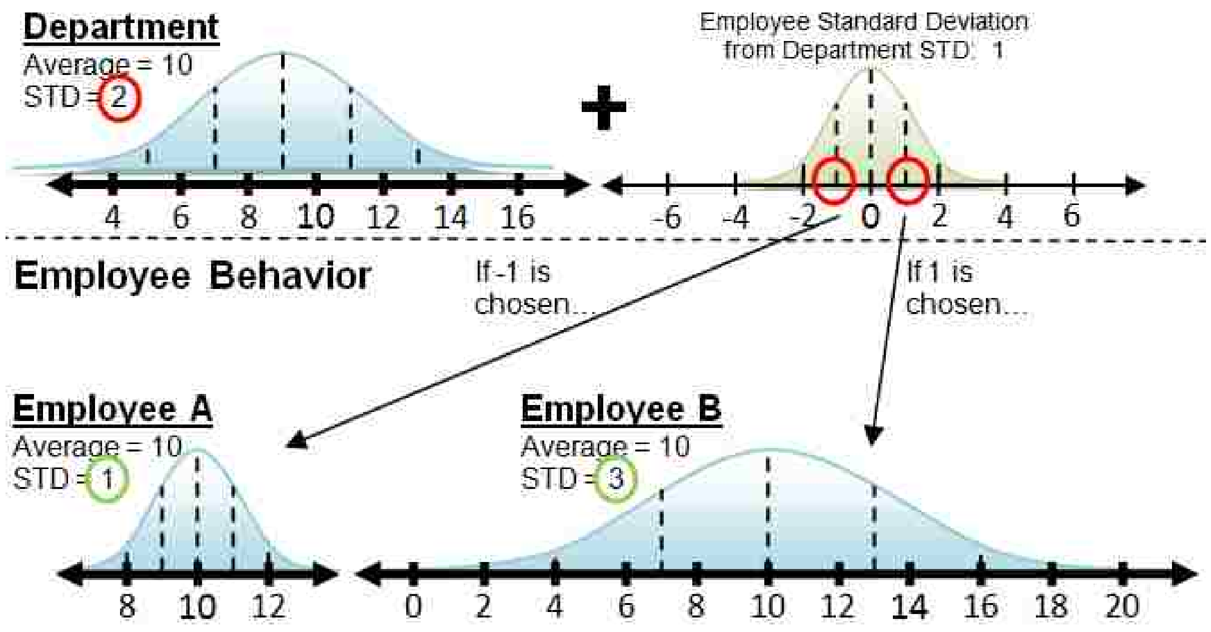


Figure 6: Example of how employee standard deviation is generated.

1. The default standard deviation of the number of actions performed each day by employees in the department.
2. A standard deviation used in determining each employee's average.
3. A standard deviation used in determining each employee's standard deviation.

3.2. Optional Input

In addition to the three required inputs of name, average and standard deviation, optional input can be provided by the user to improve the validity of the data, to set the deviation in employee behavior, and to provide greater detail about the actions. Some of these inputs have been mentioned already, and include:

- Employee Deviation from Department Average
- Employee Deviation from Department Standard Deviation
- Minimum and maximum occurrences
- Subgroups and Subtypes

The first two, the employee deviations, were just explained in the "3.1.4. Employee-Level Behavior". The remaining options will now be discussed.

3.2.1. Minimum and Maximum Occurrences

Sometimes, it doesn't make sense for an action to occur more or less than a certain number of occurrences in a day. For example, it doesn't make sense if the number of occurrences is less than 0, since an action cannot occur a negative number of times. Nor does it make sense for an action that takes an hour to perform to occur more than 8 times in an eight-hour period. As such, the option is available for the user to set the minimum and maximum number of times an action can occur in an eight-hour day (inclusively). The program enforces the bounds set by the minimum and maximum by replacing any out-of-bounds number of occurrences with the closest in-bounds value (the minimum or the maximum). If the random Gaussian-distributed value falls below the minimum, the minimum is used instead. Likewise, if the random Gaussian-distributed value rises above the maximum, the maximum is used instead. Figure 7 illustrates this:

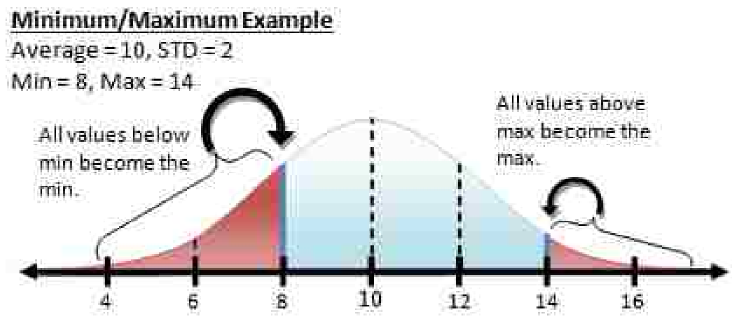

Minimum/Maximum Example
Average = 10, STD = 2
Min = 8, Max = 14
All values below min become the min.
All values above max become the max.
4
6
8
10
12
14
16

Figure 7: Example of the enforcement of minimum and maximum values.

Based on the example in figure 7, if the Gaussian-distributed random number 5 occurred, the closest in-bounds number, the minimum 8, would be used instead.

If no minimum is specified, the default minimum of 0 is used. If no maximum is specified, then the maximum is disabled.

3.2.2. Action Subgroups

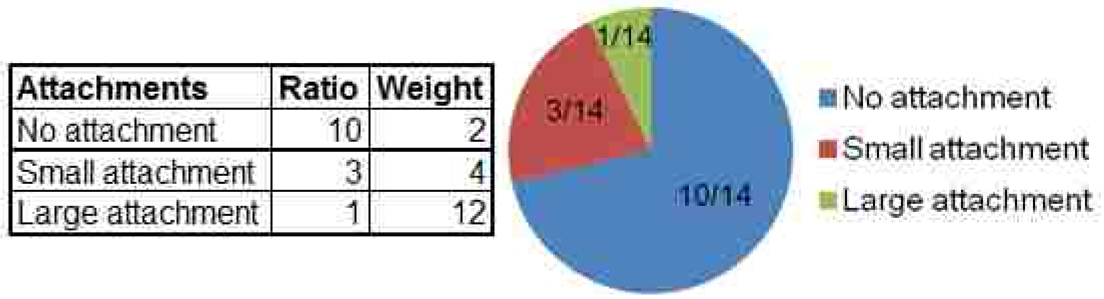
Sometimes greater detail about an action is needed besides knowledge that it occurred at a specific time by a specific employee. For instance, knowing whether a sent email had an attachment, what level of sensitivity it had, or whether it was sent inside the department, within the company, or outside the company can help in defining the threat-level of that email. The optional input of subgroups allow for this level of detail. Each action can have zero or more subgroups, and each subgroup is defined by one or more subtypes. Whenever an action occurs, one subtype is chosen for each of the action's subgroups and is written to the action log along with the action.

To illustrate, let's go back to our email example. Some possible subgroups for the "E-Mail" action could be:

1. The subgroup "Sensitivity"
    a. Subtypes "None, "Low", "Medium", and "High"
2. The Subgroup "Attachments"
    a. Subtypes "No attachment", "Small attachment", and "Large attachment".

During simulation, a subtype of each subgroup will be chosen and included in the log; in the case of the email action, "High" may be selected for the "Sensitivity" subgroup and "Large Attachment" for the "Attachments" subgroup. Another instance might have the subtypes "Low" and "Small Attachment". Through such subtype selections, two action instances can have entirely different threat levels based on the selected subtypes.

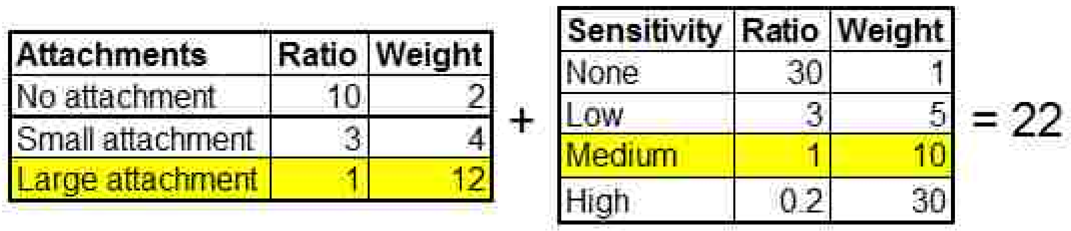
Each subtype has two fields set by the user: a ratio representing the likeliness of that subtype taking place (relative to the ratios of the other subtypes), and the weight or threat-level of that subtype. Figure 8 shows how the ratios affect the chance of a subtype being chosen:



Attachments	Ratio	Weight
No attachment	10	2
Small attachment	3	4
Large attachment	1	12

Figure 8: A pie chart showing how subtype selection is influenced by the subtypes' ratios.

The likeliness of a subtype being randomly selected is equal to its ratio divided by the total of the ratios. A subtype is chosen for each subgroup, and then the weights of the selected subtypes are added together to create the overall weight or threat-level of the action. If no subtypes are present, the default weight of the action is 1. Figure 9 demonstrates this:



Attachments	Ratio	Weight
No attachment	10	2
Small attachment	3	4
Large attachment	1	12

+

Sensitivity	Ratio	Weight
None	30	1
Low	3	5
Medium	1	10
High	0.2	30

= 22

Figure 9: The total weight of the action is the sum of the weights of its subtypes.

In addition to custom subgroups, a particular kind of subgroup can be added that is handled uniquely: the "Scope of Interaction" subgroup.

3.2.3. Scope of Interaction Subgroup

The scope of interaction subgroup is a special subgroup in that it always has the same subgroup name and three subtypes: “Local”, “Company”, and “Outsider”. Like a normal subgroup, each of these is assigned ratios and weights. However, in addition to these inputs, this subgroup takes an extra parameter that states whether the action is a subject-to-subject interaction (this is an action performed between two people) or a subject-to-object interaction.

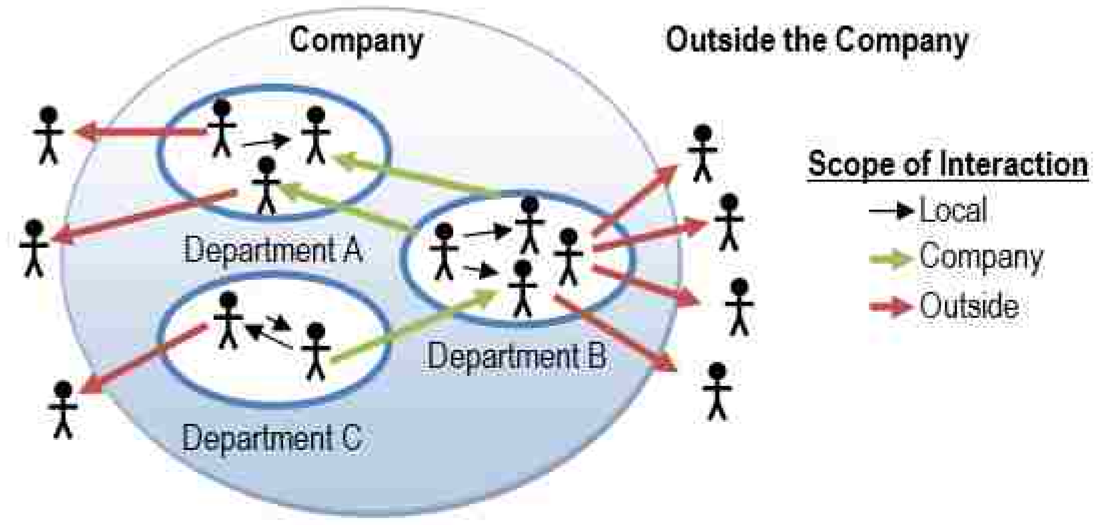


Figure 10: Person-to-person interactions where lines represent actions.

What’s unique about this subgroup is that when the simulator logs an action with this subgroup, it also logs an entity representing the person or object interacted with. In the case of subject-to-subject interactions, it will select a person based on which subtype was selected. As an example, if the subtype “local” was selected, then the simulator would log the name of a random employee within the local (employee’s) department. If “company” was selected, then a random employee within the company would be selected. While the two example analyses do not take the scope of interaction into account, the “Future Works” chapter explains how this feature can be built upon for analyses that take interactions into account.

Chapter 4

Customizing Insider Attacks and other Events

Given the behavior data of the company and departments, the simulator would be able to simulate a full log of normal activity with each employee performing according to their set behavior. However, the goal of this simulator is to simulate abnormal data alongside the normal so that analysis tools can try to detect the abnormal behavior. To achieve this goal, the simulator allows for the creation of event series that can represent abnormal behavior ranging from insider attacks to employee sick days.

Four forms of input are needed to create an event series:

1. The name of the event series
2. The number of employees to perform the event series
3. Whether the event series represents an insider attack
4. The events that should take place

For each event series, the user creates multiple events and sets when each event will take place relative to the start of the series. Each event does one of the following:

- Modifies the affected employee's current behavior
- Saves the employee's current behavior
- Loads a previously saved version of the employee's behavior
- Creates an action instance representing an action performed by the employee

During the simulation, each event series is assigned to a specified number of employees based on the Employees Affected input (the user may want x employees to get the "Sick at Home" event series during the simulation). At a random point during the simulation (different for each employee and event series), each event series will be triggered, and each event will be set in motion at their appointed time from the start of the series. Figure 13 illustrates this:

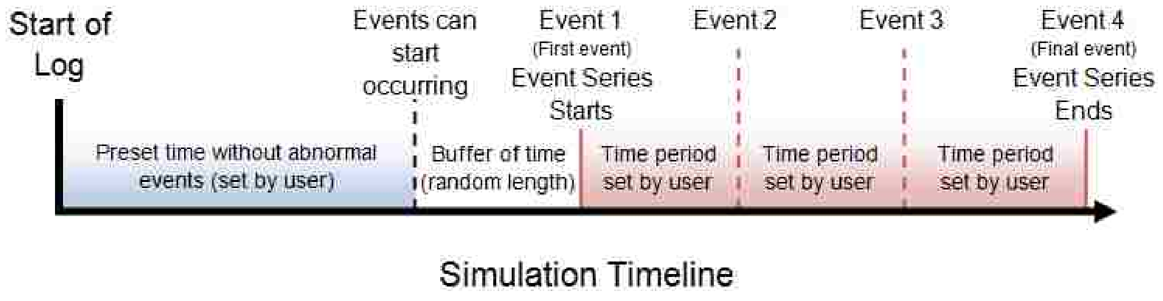

Start of Log
Events can start occurring
Event 1 (First event) Event Series Starts
Event 2
Event 3
Event 4 (Final event) Event Series Ends
Preset time without abnormal events (set by user)
Buffer of time (random length)
Time period set by user
Time period set by user
Time period set by user
Simulation Timeline

Figure 11: An example of how an event series are spaced in the employee's timeline.

Each log begins with a preset time for normal behavior, during which no event series can take place. This gives the analyses time to determine each employee's normal behavior. In addition, a random buffer of time is placed between the end of the preset time and the start of the first event series so that the series' start time is unpredictable. Once an event series begins, each event in the series occurs after the time interval specified by the user.

All of the events occur sequentially with buffers of random length in between – this prevents each event series from overlapping with others. This is enforce because, while parallel events may work in a few situations, in many other situations parallel events would conflict or not make sense (an event series representing an insider doubling their work effort would conflict with an event series representing the employee staying home sick wouldn't make sense occurring at the same time). If two events need to occur in parallel, it's recommended to create one event series with the events of the two combined.

A separate log is created during the simulation detailing when each event series took place, the name of the employee that performed the series, and whether the series was an insider attack. This additional log can then be used as an answer sheet to compare the analyses results to and thus verify whether a given analysis was able to successfully notice malicious insider attacks after processing the simulator's log.

Chapter 5

Using the Simulator Interface

The previous chapters discussed the various input a user can provide to customize the behavior and events that will be simulated by the program. This chapter explains how a user can operate the program's interface to enter and save input. The general flow of input for the simulator is the following:

1. The simulator either starts with new data files or loads existing data from comma-separated-value (CSV) files.
2. The user can use the program's interface to add, modify, or delete behavior, departments, or event series. All changes will be saved to the CSV files.
3. The simulator is run to generate a log of action instances. An additional, separate log is generated detailing who performed each event series and at what times (which can be used later to verify if the analyses detected the insider attacks).
4. An analysis is chosen and ran on the generated log, generating a file with the daily and anomaly scores.

The interface provides the means to do all four steps. However, the majority of the interface focuses on step 2, since setting the behavior and events is the most complicated step for the user.

5.1. The Interface Overview

Upon starting the program, the program's interface appears in a window as seen in figure 12. At the top of the window is a message box (figure 12, #1), which is where feedback to the user from the program appears. This feedback will include error messages, success messages, and messages detailing where generated logs can be found.

A tabbed panel takes up the majority of the interface, and clicking a tab will bring up one of five interfaces in the area labeled tab-specific interface (figure 12, #3). When the program starts, all but the first tab will initially have grayed-out text indicating that the interfaces are disabled. These tabs will become available as the data the interfaces depend on becomes available.

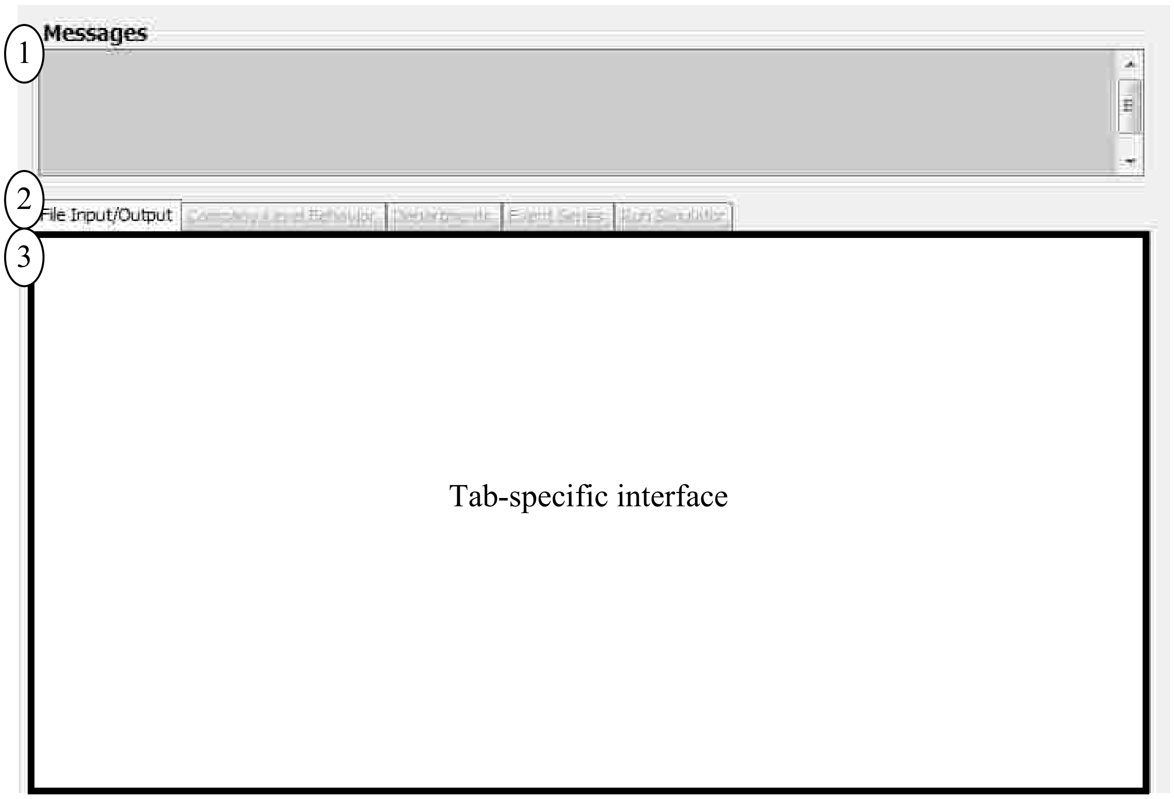


Figure 12: The windowed interface.

5.2. Data Dependencies

The user interface is broken into five sub-interfaces or sections based on the data they generate or the role they perform:

1. File Data
2. Behavior Data
3. Department Data

4. Event Series Data
5. Run Simulator/Analysis

As mentioned in the last section, only the first section's tab is available when the program starts and the other tabs are disabled (represented by the grayed-out text). This is because the first tab, the File Data section, is depended upon by all other sections. The reason for this is that the remaining sections need to know which files to save changes, new logs, and scores to. It also makes sense to load existing data first so that the remaining tabs can display the existing data for modification or deletion. In general, each section's tab won't become enabled until the data that section depends upon is received and valid.

Figure 13 shows the data dependencies in the program (excluding the dependencies on File Data since it has already been established that all other sections depend on it):

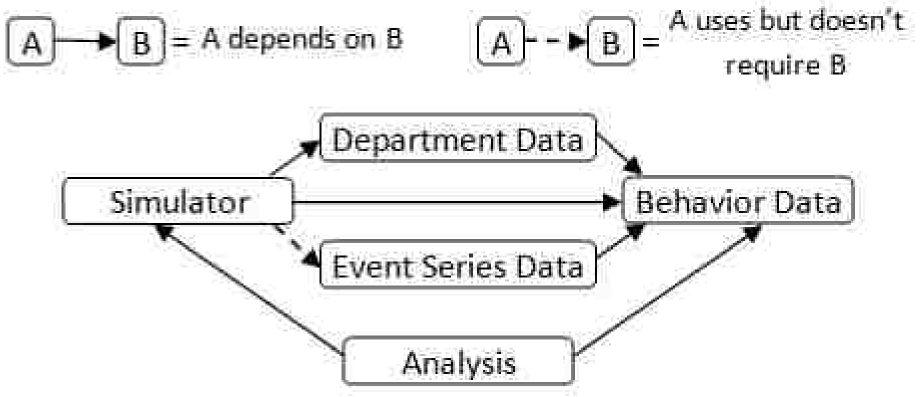


Figure 13: The program's data dependencies.

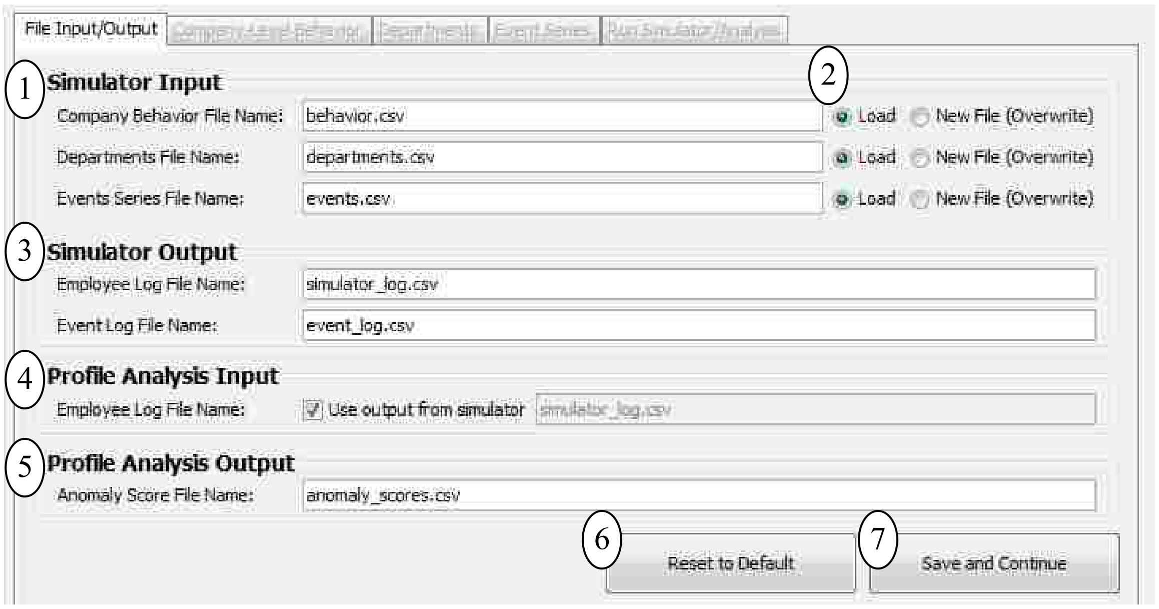
The following list explains the above dependences:

- Department Data
    - The department behavior can't be defined until Behavior Data is complete.

- Event Series Data
    - All events either cause or change behavior, and thus can’t be defined until Behavior Data is complete.
- Simulator
    - The simulator requires Behavior Data and Department Data to create the simulated log.
    - While event series data can be included (for the generation of abnormal behavior), no event series are required for the simulator to run.
- Analysis
    - Each analysis reads the log generated by the simulator and generates anomaly scores based on the weights of the behavior subtypes (thus using Behavior Data to generate scores).

5.3. File Interface

The file interface allows for the user to specify the files to read for input as well as the files to write or append data, logs, and scores to. This is the only interface initially available to the user, because the files for loading and saving behavior need to be specified before any data can be added, modified, deleted, or simulated. Each field is initialized with a default file name. Use of this file name will result in the program loading and saving from a file with the given file name from the same directory that the simulator program is running from. The user can access other files by adding the file path before the name. For instance, if the desired file was called “example.csv” in the folder “example_files”, then the user would put “./example_files/example.csv” into the field. The file interface is shown in figure 14:


File Input/Output | Company-Level Behavior | Departments | Event Series | Run Simulator/Analyses

1 **Simulator Input**

Company Behavior File Name: behavior.csv [✓ Load] New File (Overwrite)

Departments File Name: departments.csv [✓ Load] New File (Overwrite)

Events Series File Name: events.csv [✓ Load] New File (Overwrite)

2

3 **Simulator Output**

Employee Log File Name: simulator_log.csv

Event Log File Name: event_log.csv

4 **Profile Analysis Input**

Employee Log File Name: [✓] Use output from simulator simulator_log.csv

5 **Profile Analysis Output**

Anomaly Score File Name: anomaly_scores.csv

6 Reset to Default

7 Save and Continue

Figure 14: The file input/output interface.

As already mentioned, the simulator uses three forms of data as input: company-level behavior, department data (including department-level behavior), and event series data. This data is saved to three separate CSV files (figure 14, #1). Whenever data is added, modified, or deleted by the user through the interface, the old data in these files will be overwritten with the updated data. The user can elect to either load each files or to start new files (this will overwrite any files at the given locations) (figure 14, #2).

The simulator generates two logs; one is the main log where the actions of the simulated employees are written to, and the other is a log detailing when each event series took place and who performed it. The user can specify where to write these logs to (figure 14, #3).

The program's analyses read the main log generated by the simulator as input. By default, the analyses read from the same file that the simulator writes to, although the user can specify a different file to use as input (figure 14, #4). Upon reading and analyzing the log, the profile analysis will write the daily scores and anomaly scores to the output file specified by the user (figure 14, #5).

Default file name are provided by the program, and pushing the "Reset to Default" button will restore all default names (figure 14, #6). Once the desired file names are entered, pressing the "Save and Continue" button will cause the following to happen:

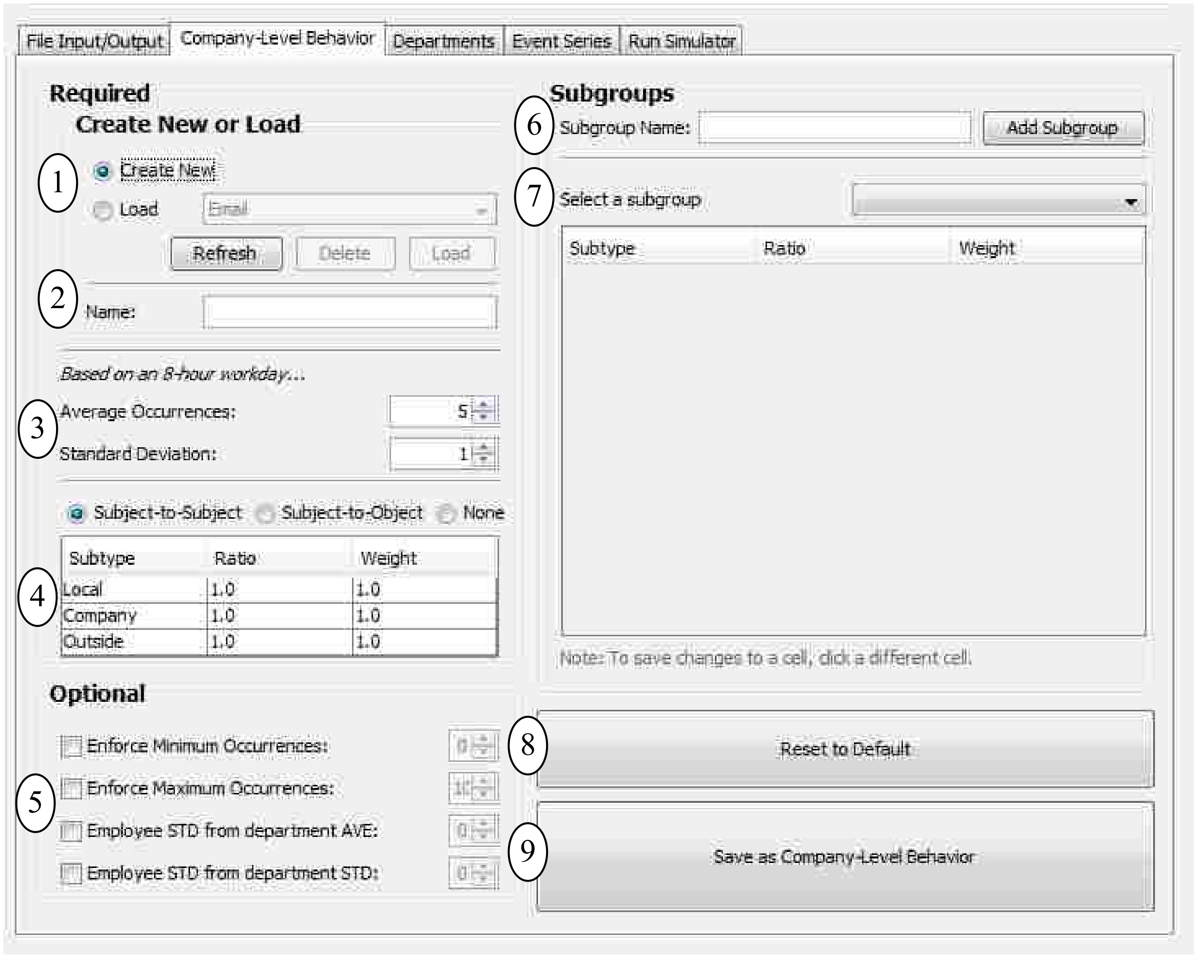
1. The file names are saved.
2. The data in the simulator input files is either loaded or cleared (depending on if "Load" or "New" is selected in figure 14, #2).
3. Each interface checks to see if the data it depends on is now in memory (see dependencies section). If so, the interface's tab is enabled. Otherwise, its tab is disabled.

If an error occurs while reading the files (such as if the file had incorrect format or wasn't accessible), then an error will appear in the message box (figure 12, #1). Otherwise, the message box will state that the files were processed successfully.

5.4. Company Behavior Interface

The company behavior interface allows for the creation, modification, and deletion of company-level behavior. As mentioned before, department data, event series data, and the simulator all are dependent on the existence of company behavior, and so all three tabs will remain disabled until at least one company behavior is defined.

Figure 15 shows the company-level behavior interface:



[✓ Create New] Load: Email

Name:

Based on an 8-hour workday…

Average Occurrences: 5

Standard Deviation: 1

[✓ Subject-to-Subject] Subject-to-Object None

Subtype	Ratio	Weight
Local	1.0	1.0
Company	1.0	1.0
Outside	1.0	1.0

Enforce Minimum Occurrences: 0

Enforce Maximum Occurrences: 10

Employee STD from department AVE: 0

Employee STD from department STD: 0

Subgroup Name:

Select a subgroup

Subtype	Ratio	Weight

Note: To save changes to a cell, click a different cell.

Figure 15: Company-level behavior interface.

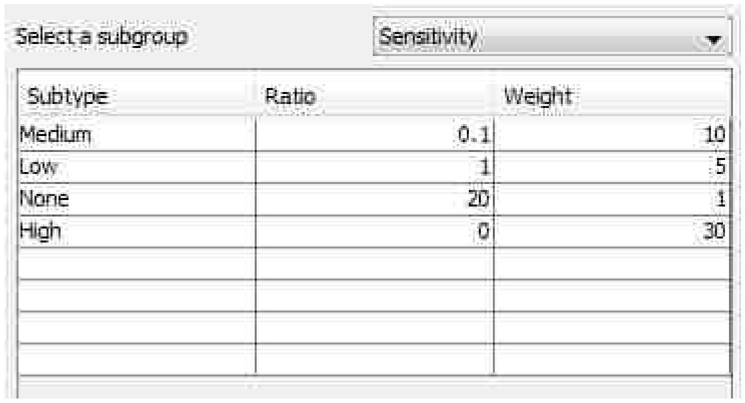
The user can choose to create, load, or delete behavior (figure 15, #1). Loading a behavior will update fields 2-7 with the loaded behavior's data. If modifications are made to the loaded data and the "Save as Company-Level Behavior" button (figure 15, #9) is clicked, the behavior will be overwritten with the new data. Field #2 is where the name of the action described by the behavior is specified, and the fields at #3 set the average number of occurrences and standard deviation of the company-level behavior.

The table in figure 15 labeled #4 sets the scope of interaction subgroup, which is treated different from regular subgroups because (1) the subtype names never changes and (2) the subgroup also keeps track of whether it is a subject-to-subject or subject-to-object behavior. If the user doesn't wish for this subgroup, they can disable it by selected the "None" radio button.

The fields labeled #5 in figure 15 are optional fields that, if desired, can be enabled and filled out. The minimum and maximum prevent the number of occurrences of the action to take place outside the range they specify. The employee standard deviation (STD) fields determine how much the employee's behavior deviates from the department and company behavior (see "Customizing Simulated Behavior – Optional Behavior Data" for more details). Only checked fields will be saved for the behavior.

Subgroups can be created by entering the subgroup name and pressing the "Create Subgroup" button at figure 15, #6. Once this is done, the named subgroup will be available in the dropdown box at #7. Selecting that subgroup will make the subtype table appear with default values in cells. A subtype can be added to the table by writing its name, ratio and weight into a row. At least two subgroups need to be defined, and the combined ratio of all of the subtypes must be a positive integer.

Figure 16 shows what this table looks like if four subtypes are added:


Select a subgroup: Sensitivity

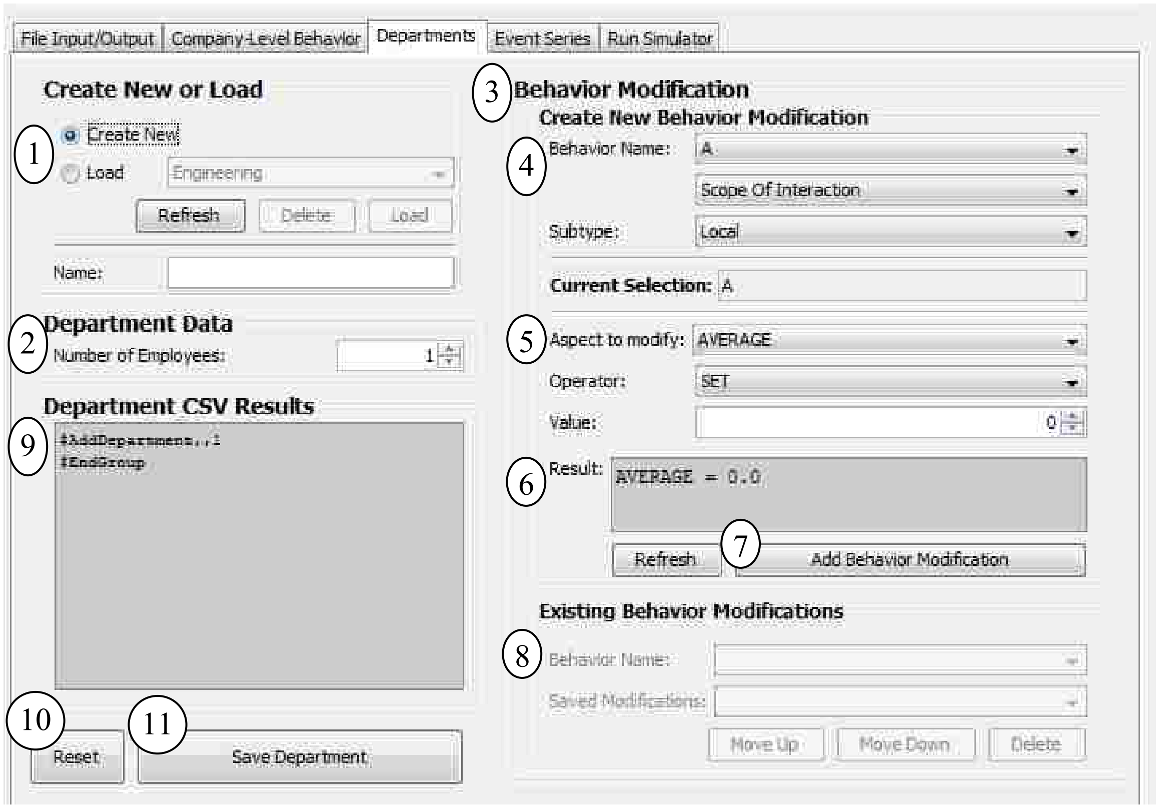
Subtype	Ratio	Weight
Medium	0.1	10
Low	1	5
None	20	1
High	0	30

Figure 16: Table of subtypes for the subgroup "Sensitivity".

If at any point the user wants to reset the values to their default settings, the user can click the “Reset to Default” button (figure 15, #7). On the other hand, the “Save as Company-Level Behavior” button (figure 15, #8) saves all changes to the file specified earlier in the “File Input/Output” panel. If the “new” radio button is selected, it will save the data as a new behavior, overwriting any behavior with the same name. If a behavior had been loaded and the “behavior” radio button is still selected, then the loaded behavior will be overwritten with the new data (and renaming the behavior if a new name was entered).

5.5. Department Interface

The department interface is used to create, modify, and delete departments. The interface is shown in figure 17:



Load: Engineering

Name:

Number of Employees: 1

```
#AddDepartment,,1
#EndGroup
```

Behavior Name: A

Scope Of Interaction

Subtype: Local

Current Selection: A

Aspect to modify: AVERAGE

Operator: SET

Value: 0

Result:

```
AVERAGE = 0.0
```

Figure 17: The department interface.

The creation, modification, and deletion of departments is performed in the same way company-level behavior is created, modified, and deleted (figure 17, #1). If saved departments exist, the user can select one and load that department to the interface, which will update all of the fields in the panel with the department's data. Saving after the load (without clicking "Create New" or resetting) will cause that department's former behavior to be overwritten with the new data.

Besides the name, the only other required data for the department is the number of employees the department has (figure 17, #2). Some caution should be taken when selecting this number: when the simulator runs, it will create the number of employees specified and generate behavior for every single one of them. In addition to increasing the time for the simulation to finish, a large number of employees could result in a very, very large log of actions.

By default, department-level behavior matches the company-level behavior. The panel on the right side of the department interface (figure 17, #3) allows the user to change the department-level behavior (figure 17, #4) by either adding modifications to the company's default or setting a new value for it (see "3.1.4. Employee-Level Behavior" for more details). To do either, the user selects the aspect of the behavior that they want to set or modify (figure 17, #5). There are six aspects of each behavior that can be modified at the department level:

1. Average
2. Standard Deviation
3. Employee Deviation from the Department Average
4. Employee Deviation from the Department Standard Deviation

5. Subgroup Standard Deviation (how much the subgroup's subtype ratios vary from employee to employee)
6. Subtype Ratio (the ratio of a specific subtype)

Once one of the six aspects is chosen, the user can then decide how to change this value at the department level. They do this by choosing an operator and a value (figure 17, #5). The five operators include:

1. Set
    a. Sets the aspect to the value, i.e. NewAspect = Value.
2. Add
    a. Adds the value to the aspect, i.e. NewAspect = OldAspect + Value.
3. Subtract
    a. Subtracts the value from the aspect, i.e. NewAspect = OldAspect - Value.
4. Multiply
    a. Multiplies the value by the aspect, i.e. NewAspect = OldAspect * Value.
5. Divide
    a. Divides the value by the aspect, i.e. NewAspect = OldAspect / Value.
    b. The value is not allowed to be 0 (to avoid any divide-by-zero errors).

Once the aspect, operator and value have been chosen, the user can add the modification by clicking on the "Add Modification" button (figure 17, #7). Modifications can be selected in the "Existing Behavior Modifications" drop box (figure 17, #8) and reordered or deleted through the buttons below the drop box.

The department CSV message box displays the department as it will appear in the CSV file. This box shows the department behavior modifications, including the order they take place. The reset button (figure 17, #9) allows the user to set the department data back to the default values and delete all behavior modifications. Finally, the "Save Department" button (figure 17, #10) provides the means to save the department to the CSV file. The save button performs the following when pressed:

1. Verifies that the department data is complete and valid.
2. Makes sure every behavior modified by the department is defined.
3. Saves the data to the CSV file.

5.6. Event Series Interface

Similar to the last two interfaces, the event series interface is designed to allow the user to create, modify, and delete event series to be used in the simulator. Figure 18 shows the event series interface.

The interface for creating a new event series or loading an existing one is the same as the ones used for creating and loading behavior and departments (figure 18, #1). To reiterate, the user can either create a new event series, or load an existing event series. Doing the latter will update all of the fields in the interface with the loaded event series' values. The name of new event series is also set here.

The user can specify the number of employees that will perform this event series (figure 18, #2) as well as whether the series represents an insider attack (figure 18, #3). After these are set, the only step remaining is to add events. For each event to be added, the user specifies how long after the start of the event series that the event should take place (figure 18, #4). It should be noted that if an event isn't set to take place on day zero when the event series is saved, then all of the events will be moved back enough days so that the earliest event takes place on day zero.

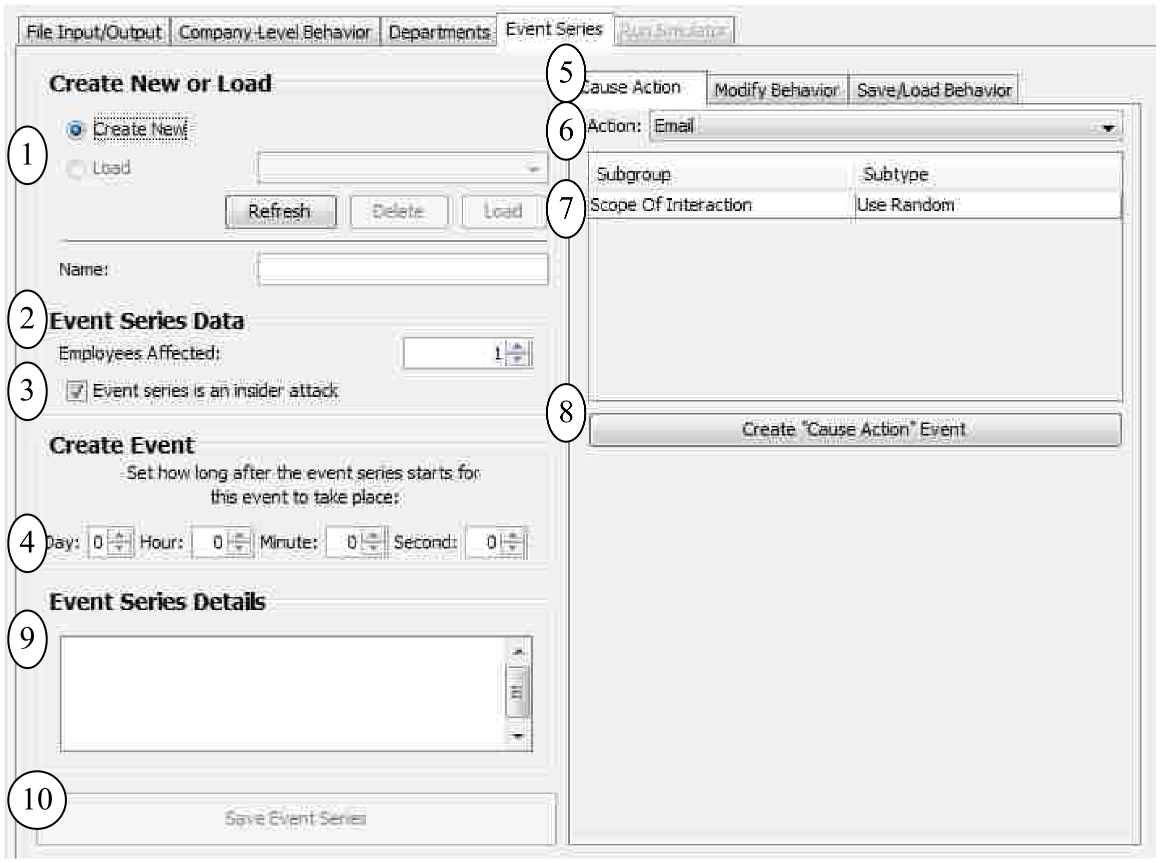
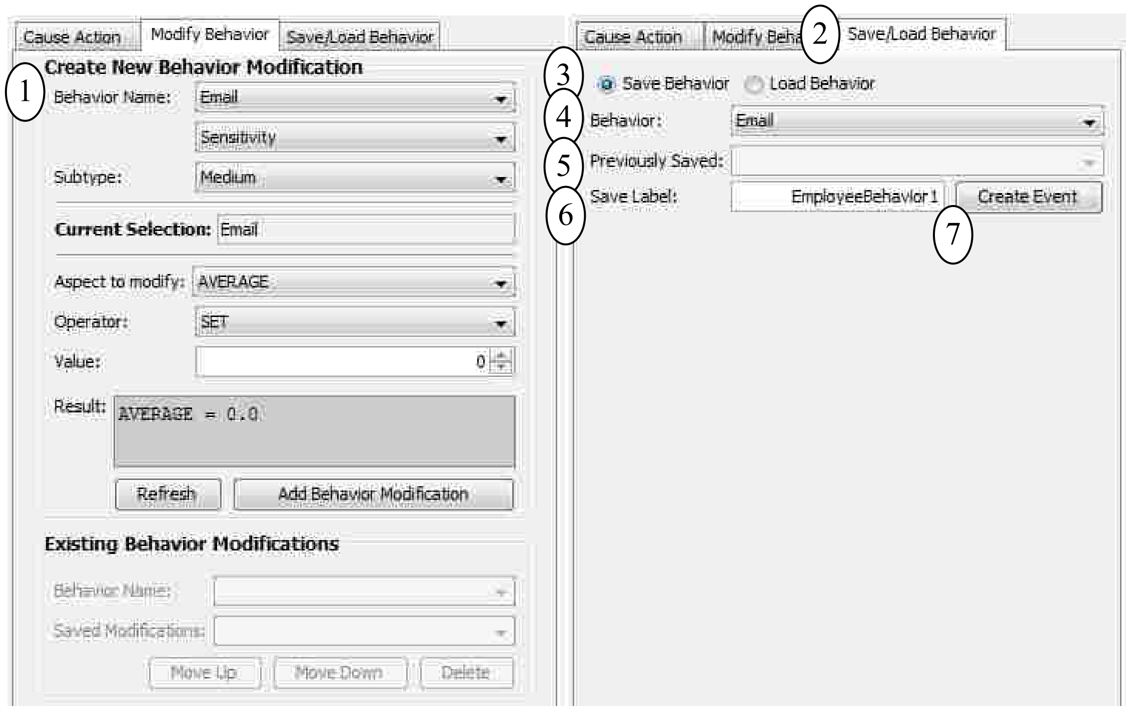


Figure 18: The event series interface.

The user can then choose to create one of four events to take place at the specified time:

1. Cause an action to be performed by the affected employee (figure 18, #5).
2. Cause the behavior of the affected employee to change (figure 19, #1).
3. Save a behavior (figure 19, #2).
4. Load a behavior (figure 19, #3).

Figure 18 shows the interface for the "Cause Action" event type, while figure 19 shows the interfaces for the other three event types:



Behavior Name: Email
Sensitivity
Subtype: Medium
Current Selection: Email
Aspect to modify: AVERAGE
Operator: SET
Value: 0
Result: AVERAGE = 0.0

[✓ Save Behavior] Load Behavior
Behavior: Email
Previously Saved:
Save Label: EmployeeBehavior1

Figure 19: Interfaces for the "Modify Behavior", "Save Behavior", and "Load Behavior" event types.

To create a "Cause Action" event, the user selects the "Cause Action" tab (figure 18, #5). The user first selects what action the employee should perform; doing so will cause a table to appear below it with a row for every one of the action's subgroups (figure 18, #6). The user can then specify which subtypes they want the caused action to have by selecting the cells to the right of each subgroup to make a dropdown menu appear. The user can then select the desired subtype for the subgroup, or select "use random" if they want the subtype to be randomly selected based on the subtype ratios. During the simulation, when the event is triggered, the action specified will be logged with the event time and with the selected subtypes.

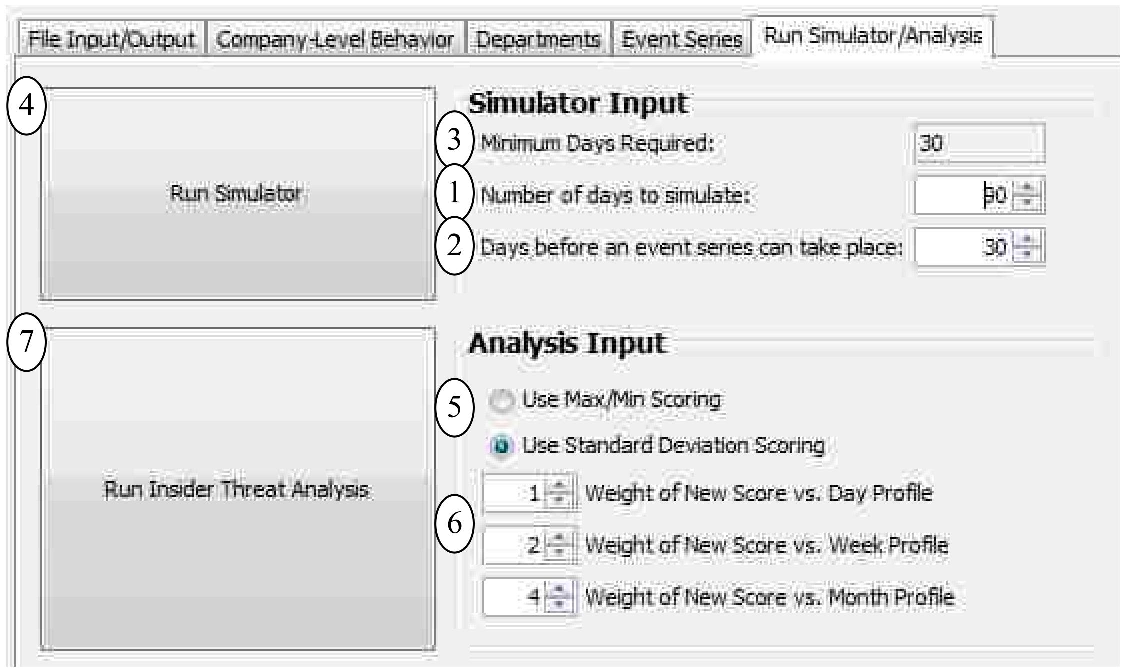
The "Modify Behavior" event (figure 19, #1) has the same interface as the behavior modification in the department interface. The only two differences between them is (1) when the modification takes place and (2) the target of the modification. The department behavior modifications take place at the beginning of the simulation and are used to set the department behavior. In contrast, “Modify Behavior” events change the behavior of the employee at the time specified in the event series. Since setting the behavior modifications for events is the same as department behavior modifications, the department interface section can be referenced on how to create behavior modifications.

The “Save Behavior” and “Load Behavior” events share an interface accessed through the “Save/Load Behavior” tab (figure 19, #2). The user selects whether they want to save current behavior or load a previously saved behavior (figure 19, #3). The behavior to save or load is selected next (figure 19, #4). If “Load Behavior” is selected and the selected behavior has been previously saved, then previously saved behavior will be selectable in a drop-down menu (figure 19, #5). If “Save Behavior” is selected, then the user can enter a name to save the behavior under (figure 19, #6). This name will appear in the drop-down menu for future “Load Behavior” events. The user can then save the event by pressing the “Create Event” button.

The user can create as many events as desired. All data about the event is displayed in the CSV message box as it is added (figure 18, #9). When the event series meets the user’s satisfaction, they can save the event series with the “Save Event Series” button.

5.7. The Run Simulator/Analysis Interface

The final interface allows the user to set some final parameters before running the simulator or an analysis. First, the user can choose how long they want the simulation to run for (figure 20, #1) and how many days should pass before any events take place (figure 20, #2). The minimum number of days that the simulator needs to run in order for



Simulator Input

Minimum Days Required: 30

Number of days to simulate: 60

Days before an event series can take place: 30

Analysis Input

○ Use Max/Min Scoring

● Use Standard Deviation Scoring

1 Weight of New Score vs. Day Profile

2 Weight of New Score vs. Week Profile

4 Weight of New Score vs. Month Profile

Figure 20: The "Run Simulator/Analysis" interface.

all event to be able to sequentially complete is shown in the field labeled #3; the number of days to simulate cannot be less than this number. The "Run Simulator" button (figure 20, #4) runs the simulator for the set number of days, and will display a message stating where the completed log can be found in the message box if successful (figure 12, #1). If an error occurs, an error message will be shown in the message box instead.

Once the simulator has been successfully run, an analysis can be run on the generated log. The user can determine the analysis to use (figure 20, #5) as well as the weights to be used for the day, week, and month profiles (figure 20, #6). Each profile's anomaly score will be multiplied by the given weight, and then divided by the total of the weights. For example, if the anomaly scores of the day, week, and month profiles were 5, 9.5, and 8 respectively and the respective weights were 1, 2, and 4, then the resulting anomaly score would be:

```
((5 * 1) + (9.5 * 2) + (8 * 4)) / 7 = 8
```

All weights are required to be non-negative. Since the weighted scores are summed and divided by the total, at least one weight must be positive. Once these details are set, the user can run the analysis by pressing the “Run Analysis” button (figure 20, #7). Like the simulator, it will display a message stating where the completed anomaly scores file can be found in the main message box (figure 12, #1). If an error occurred during the analysis, an error message will be displayed instead.

5.8. Tips for Entering Data

Mind the data dependencies when modifying data. If a behavior is modified or deleted, it should be ensured that doing so won’t break any department or event series data. In addition, if the analysis tries to analyze a previously generated simulator log and encounters a behavior, subgroup, or subtype that is no longer found in the behavior data (whether because the behavior was modified or deleted, or because a different behavior file is being used), then an error will be thrown and the analysis won’t complete.

It’s also worth reiterating that each time data is saved in the interface, the program will attempt to update the corresponding CSV file. If that file is currently being accessed by another program or it is otherwise inaccessible, the file will not be updated and an error message will be displayed in the message box.

Finally, while the interface is useful, there will be times where editing the CSV files directly could prove easier or more efficient than using the interface. For more information on the CSV files and their formats, please refer to Appendix A.

Chapter 6

Simulated Action Generation

Now that the ways of providing input to the simulator has been covered, the steps taken by the simulator to generate the employee action log will be considered. By the time the simulator runs, the following data must be available to it:

- Company-level behavior descriptions
- Department descriptions
- Event series descriptions (if any)
- Total number of days to simulate
- Number of days to simulate before any event series can start taking place

Using this input, the simulator will generate employees and simulate their behavior, logging their actions over the log's duration. The general outline of what the simulator does is as follows:

1. Compute the behavior of each department.
2. Generate employees for each department.
3. Assign each event series to the desired number of employees.
4. Do the following for each employee:
    a. For each event series assigned to the employee…
        i. Determine when the series will begin.
        ii. Write the event series, its start time, and the employee performing the series to the event series log.
    b. Create a queue of event instances sorted by time of occurrence.
    c. For each event instance…

i. Generate the employee's actions up to that event instance.

ii. Resolve the event.

d. Generate the employee's actions for the remainder of the simulated time.

5. Output the generated log.

6.1. Computing Department Behavior

The process for generating the department behavior has been described earlier in chapter 3. In brief, the department's behavior begins as a copy of the company's behavior, and then the department behavior modifications are applied to the department's behavior. Figure 4 illustrated this and is shown again as figure 21:

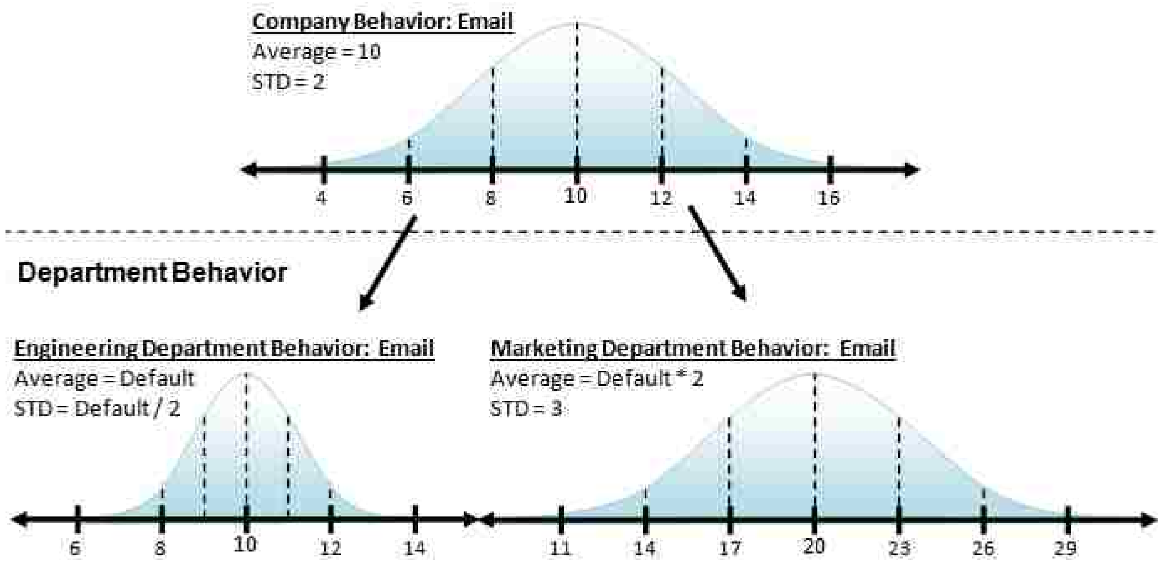


Figure 21: How department behavior is generated based on company behavior.

Each department can have 0 or more behavior modifications assigned to it. Each behavior modification will specify:

- The name of the behavior to modify
- The aspect of the behavior to modify (i.e. average, standard deviation, etc)
- The operator to perform on the aspect (i.e. set, add, multiply, etc)

- The value to use in the operation (i.e. the value to add or multiply by)

The behavior of the department is initially set to match the company-level behavior. Then each behavior modification is performed sequentially from the beginning of the list of modifications to the end. For more information on behavior modifications, including what aspects can be modified, how the operators and values are applied, and how to set the order of behavior modifications, refer to “5.5. Department Interface”.

6.2. Generate Employees for Each Department

Once each department’s behavior is set, the company’s employees are generated. The employee average is determined using the following equation:

```
Employee_Average = Department_Average
+ Department_Employee_STD_From_Department_Average
* random.nextGaussian()
```

The command “random.nextGaussian()” returns a Gaussian-distributed number with an average of 0 and a standard deviation of 1. By multiplying this by the standard deviation of employees from the department average and adding the department average, the employee’s average becomes a Gaussian-distributed number with an average determined by the department’s average and a standard deviation equal to the department’s standard deviation of employees averages from the department average. The employee’s standard deviation is calculated with a similar formula:

```
Employee_STD = Department_STD
+ Department_Employee_STD_From_Department_STD
* random.nextGaussian()
```

These formulas are used to set the behavior of each behavior for each employee.

6.3. Assign the Event Series to the Employees

Once all of the employees and their behavior have been generated, the assignment of event series to employees needs to take place. Each event series specifies how many employees should be assigned to perform its events. An error is thrown and the simulation aborted if the number of employees to be affected is less than the number of employees in the company. Otherwise, if n employees are to be affected, then n employees are selected at random and assigned the event series. No employee will be assigned the same event series twice, although it is possible that one employee could be assigned every event series. It is for this reason that the length of time for the log to run is required to be large enough for an employee to perform every single event series in sequence.

6.4. Determine the Start Time of Each Event Series for Each Employee

When all event series have been assigned to the appropriate number of employees, the start time of each event series for each employee is determined. This is calculated independently for each employee, so employees with the same event series will still have the series starting at different times. Figure 22 demonstrates how the event series are distributed throughout the simulation timeline:

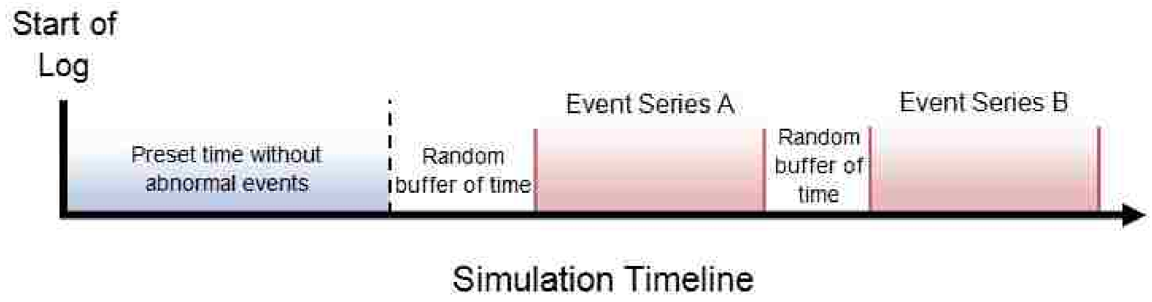


Figure 22: The distribution of multiple event series.

All of the events will occur sequentially and in a random order with time buffers of random length in between. The total amount of time available for time buffers is calculated by taking the total time span of the log and subtracting the time span of each event series as well as the preset time before events can take place. For n event series, the buffer is randomly split into n+1 time spans: one before each event, and one between the last event and the end of the log. Once the sequence of events and buffers is set, the start time of each event is calculated.

Each event series is written to a CSV file so that the user can verify when every event series took place and who performed it. Appendix B shows the format of these files.

6.5. Create an Event Instance Queue

For each employee, the simulator will start at day 0 and simulate normal behavior until the first event occurs. It will then resolve the event and continue simulating the employee's behavior until the next event. To make this easier, all of the events in the event series are placed into event instances (an event instance is an object that holds an event, a time of occurrence and the name of an employee). These event instances are then sorted by when they occur and placed into a queue. Figure 23 demonstrates how this combining and sorting takes place (below the event name is four numbers that represent the day, hour, minute and second at which the event takes place):

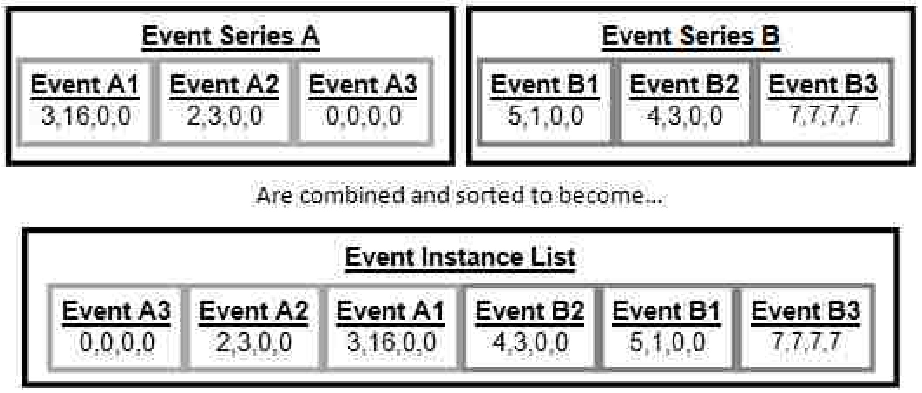


Figure 23: Illustration of how the event instance list is built.

6.6. Simulate Behavior between Event Instances

If a simulator has n events, then there are n+1 periods of time that the program needs to generate simulated behavior for: the time periods before each event, and the time between the last event and the end of the program. The logs for each of these periods will be called sub-logs. Starting with the first day, the simulator calculates how many working hours the employee puts in for that day (the default work hours of the employee are 8 AM to 4 PM). To do so, it calculates the following:

- The actual start time, calculated as the later time of the following:
    - The start of that day's work day
    - The start of the sub-log day
- The actual end time, calculated as the earlier of the following:
    - The end of that day's work day
    - The end of the sub-log day

The default beginning of the sub-log day for a day x is day x, hour 0, minute 0, and second 0 and the default end of the sub-log day is day x, hour 23, minute 59, second 59. The first sub-log day is an exception and instead shares the same start time as the sub-log, which means that the first sub-log day of a sub-log that starts on day v, hour x, minute y, second z also starts on day v, hour x, minute y, second z. Similarly, the last sub-log day is also an exception and shares the same end time as the sub-log, which means that the last sub-log day of a sub-log that ends on day v, hour x, minute y, second z also ends on day v, hour x, minute y, second z.

Figure 24 shows a few possible examples:

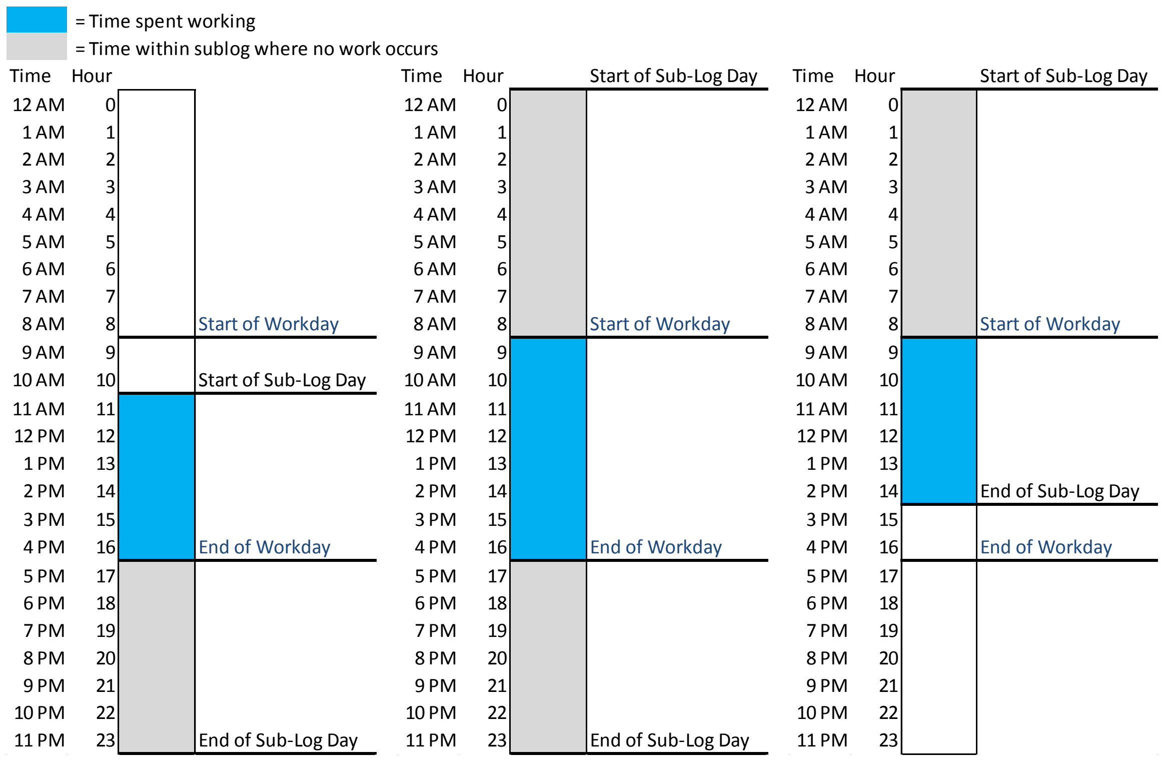


Figure 24: Three examples of how the time spent working in a day is calculated.

Using the above example in figure 24, suppose the sub-log's time period started on day 2, hour 10 and ended on day 4, hour 14. The start of the actual work for day two (left example in figure 24) would be the later time between 8 AM (the start of the workday) and 10 AM (the start of the sub-log day), which is 10 AM. The end of actual work would be the earlier of 4 PM (the end of the work day) and 11:59 AM (the end of the sub-log day), which would be 4 PM. This means that the employee spent 6 hours working on day 2. Day 3's work starts at hour 8 and ends at hour 16, since neither the start nor the end of the sub-log day occur on day 3 (see middle example of figure 24). Finally, day 4's work day would start at hour 8 but end at hour 14, when the sub-log also ends.

Once the number of hours worked is calculated, the number of occurrences of each action for that time period is computed. The formula for this is:

```
Occurrences = (Hours_Worked / 8) *
(Average + Standard_Deviation * random.nextGaussian())
```

The hours worked is divided by 8 because the average and standard deviation are based on the number of occurrences in an eight-hour day. If the resulting number of occurrences falls outside the bounds set by the behavior's minimum and maximum, the closest in-bounds value is returned instead (so if -2 is generated and the minimum is 0, then 0 is used instead). To simulate each occurrence, four steps occur:

1. A random time between the actual work hours is selected.
2. For each of the behavior's subgroups, a subtype is selected randomly based on the subtypes' ratios (see figure 8 in Chapter 3 for more details).
3. An action instance is created. Each action instance logs the action performed, who performed it, the time selected in step 1, and the subgroup-subtype pairs from step 2.
4. The action instance is added to a list of action instances that are sorted by the time of occurrence. This list of action instances will later be written to the simulator log.

6.7. Resolve Each Event

After each sub-log (excluding the final sub-log) completes, the event that follows the sub-log is resolved. What this event does is based on the type of event.

If the event is a "Cause Action" event, then an action instance is generated based on the event's parameters. The cause action event specifies the action that occurs, the time the action takes place, and some of the subgroup-subtype pairs. Any subtypes that aren't specified will be selected randomly based on the subtype ratios. This created action instance is then saved in the same list as the other action instances.

If the event is a "Modify Behavior" event, then each behavior modification specified in the event is applied to the employee's behavior. Each behavior modification will state what's being modified (the behavior average, standard deviation, or subtype ratio), how it's being modified (one of the following operators: Set, Add, Subtract, Multiply, or Divide), and by what value. For instance, if the average is going to be modified with the operator Subtract and the value 2, then the following is performed:

```
New_Average = Old_Average - 2
```

Once each modification is performed, the event is resolved.

The "Save Behavior" event saves an event under a given name so that it can be loaded later using the given name. Specifically, it takes the behavior and saves it to a hash map using a key that combines the behavior name and the given name. Load events also have a specified behavior and a name. When a load event occurs, the same hash map is queried with a key also based on the behavior's name and the given name. If found, the employee's saved behavior is restored, overwriting the employee's behavior.

The load/save events allow the user to save an employee's behavior, make behavior modifications, run the simulator with the altered behavior, and then restore the old behavior. For instance, to simulate a sick day, an employee's behavior can be saved using a "Save Behavior" event. All of the employee's behavior could then be set to not occur with "Modify Behavior" events. After a day of running the simulator with no action taken by the employee, the former behavior is restored with a "Load Behavior" event.

6.8. Writing the Action Instances to a CSV File

Once all actions instances are complete, they are written to the simulator log file in CSV format (see Appendix B). If no errors occur, then a message is displayed to the user stating that the simulator ran successfully as well as where the resulting log can be found.

Chapter 7

The Analyses

Once the simulator has completed and a log of actions has been generated, analyses can be run on the log to find the hidden insider attacks. To demonstrate how this can be done, two examples analyses have been programmed and added to the simulator. Both of them are based on the profile analysis suggested by Kim, Sheldon, and Hively (2012) as described in Chapter 2. To summarize, they believe that an insider's day-to-day behavior changes before, during, and after performing an insider attack. If their new behavior can be effectively compared to their old behavior, the abnormal behavior surrounding an insider attack can be found and the insider attack discovered.

The question then arises on how to store and use old behavior. According to Kim et al. (2012), the raw data generated by all of the insiders of a company in a single day would be quite large. Within a matter of weeks, the data would become far too large for efficient storage retrieval or comparisons. Their solution was to consolidate the old behavior into action summaries, or profiles. These profiles would be significantly smaller than the data they summarize, making them far easier to store and use.

Three profile types were proposed: day profiles, week profiles, and month profiles. Each day's activities would be consolidated into a day profile. Seven day profiles would be used to create a week profile, and four week profiles would be used to create a month profile. This would mean that a total of twelve profiles would be kept for comparison to new behavior as opposed to a month's worth of actions.

7.1. Computing Activity Scores

Profiles generate activity scores (not to be confused with the anomaly score) based on the weights assigned to each action's subtypes. Each action instance read from the simulator log is read into a day profile and the weights of the action instance's subtypes are added to the profile's overall activity score (or 1 is added if no subtypes are given). In short, the activity score of a profile is the sum of the weights of the action subtypes read into the profile. At this point, the anomaly score is generated (the following section will cover how this is done). Once the anomaly score is generated and saved, the day profile's activity score is passed to the week profile. The week profile collects seven scores (one from each day within the week), and then generates its own activity score (the average of the seven day activity scores). The week profile then passes its generated activity score to the month profile, which, like the week profile, collects four week activity scores before generating its own (the average of the four week activity scores).

7.2. Computing Anomaly Scores

Up to this point, the two example analyses have the same steps. Both of them generate activity scores in the exact same way. How they use and interpret these scores differs. The first examples analysis, the one that's most closely based on Kim, Sheldon, and Hively's proposed profile analysis, will be described first. This analysis has been nicknamed the "Max/Min Analysis" based on its formula for generating anomaly scores. A discussion of the second analysis, nicknamed the "Standard Deviation Analysis", will follow.

7.2.1. The Max/Min Analysis

The Max/Min Analysis computes a new day profile's anomaly score by comparing the day profile's activity score to the activity scores of the previous day profile, week profile, and month profile. A ratio is generated by each comparison, in which the larger of the activity scores is divided by the lower activity score. For instance, if the new day profile's activity score is 20 and the previous day profile's score was 23, then the ratio generated is 23/20. If the week profile's score was 37 and the month profile's score was 15, then the two generated ratios would be 37/20 and 20/15, respectively, since the lower activity score is always the divisor.

The previous day profile, week profile, and month profile each have a weight associated with them that's set by the user before the analysis runs (their default weights are 1, 2, and 4 respectively). Each previous profile's generated ratio is multiplied by this weight, added together, and then divided by the sum of the weights. This is done so that profiles with higher weights have a higher influence on the resulting anomaly score. Finally, 1 is subtracted from the result so that perfectly normal behavior results in an anomaly score of 0 while abnormal behavior produces positive integers.

Suppose we had the following variables:

- A_n is the new day profile's activity score
- A_d is the activity score of the previous day profile
- A_w is the activity score of the previous week profile
- A_m is the activity score of the previous month profile
- W_d is the day profile weight
- W_w is the week profile weight
- W_m is the month profile weight

The formula for calculating the anomaly score is the following:

$$\text{Anomaly Score} = \frac{\frac{\max(A_n, A_d)}{\min(A_n, A_d)} * W_d + \frac{\max(A_n, A_w)}{\min(A_n, A_w)} * W_w + \frac{\max(A_n, A_m)}{\min(A_n, A_m)} * W_m}{W_d + W_w + W_m} - 1$$

For instance, if the activity scores for the previous day profile, week profile, and month profile are 25, 35, and 40, then a new score of 26 would produce the following ratios:

Day-Profile Ratio:	26/25
Week-Profile Ratio:	35/26
Month-Profile Ratio:	40/26

If we were to use the default weights of 1, 2, and 4 for the scores of day profile, week profile, and month profile respectively, then the resulting anomaly score would be:

$$\text{Anomaly Score} = \frac{\frac{\max(26, 25)}{\min(26, 25)} * 1 + \frac{\max(26, 35)}{\min(26, 35)} * 2 + \frac{\max(26, 40)}{\min(26, 40)} * 4}{1 + 2 + 4} - 1$$

$$= \frac{\frac{26}{25} * 1 + \frac{35}{26} * 2 + \frac{40}{26} * 4}{7} - 1 = \frac{\frac{26}{25} + \frac{35}{13} + \frac{4}{3}}{7} - 1 \cong 9.89$$

There are a few difficulties that arise with this analysis, especially when the minimum value approaches or equals zero. If the minimum equals zero, then a divide-by-zero exception occurs. Further, as the minimum approaches zero, the anomaly score jumps dramatically. 8.1.3 will explore the reason behind this in greater detail. The program's work-around is to add 1 to both the maximum and minimum if the minimum is less than 1.

7.2.2. The Standard Deviation Analysis

The standard deviation analysis is a derivative analysis of the max/min analysis. In addition to keeping track of the average of activity scores, the profiles also keep track of the standard deviation of activities during their time. The anomaly score generated by a new score is calculated as the number of standard deviations the new score is away from the profile's average. To make this work, the day profile is replaced with a 3-day profile. Each day, the average and standard deviation of activity scores over the past three days is calculated and becomes the activity score and standard deviation of the 3-day profile. These values are then passed to the week profile, which uses the average of these values for its own average and standard deviation. It then passes the week's average and standard deviation to the month profile. After collecting four such values, the month profile generates its own average and standard deviation.

Each previous profile (the 3-day profile, the week profile, and month profile) will all have an activity score (the average of the activity scores within their duration) and a standard deviation. Similar to the max/min analysis, each profile also has an associated weight set by the user. The default weights of the 3-day profile, week profile, and month profile are 1, 2, and 4 respectively.

Suppose we had the following variables:

- A_n is the new day profile's activity score
- A_d is the activity score of the previous 3-day profile
- A_w is the activity score of the previous week profile
- A_m is the activity score of the previous month profile
- Std_d is the standard deviation of the previous 3-day profile
- Std_w is the average standard deviation of the previous week profile
- Std_m is the average standard deviation of the previous month profile
- W_d is the day profile weight

- W_w is the week profile weight
- W_m is the month profile weight

The formula for calculating the anomaly score A_{score} is the following:

$$\text{Anomaly Score} = \frac{\frac{\text{abs}(A_n - A_d)}{\text{Std}_d} * W_d + \frac{\text{abs}(A_n - A_w)}{\text{Std}_w} * W_w + \frac{\text{abs}(A_n - A_m)}{\text{Std}_m} * W_m}{W_d + W_w + W_m}$$

Basically, the number of standard deviations the new score is away from each profile's averages is multiplied by their appropriate weights, summed together, and divided by the total of the weights.

7.3. Were the Abnormal Actions Noticed?

The anomaly scores can be evaluated to see if they detected the change in behavior of the malicious insiders. This can be done by graphing the anomaly scores and looking for patterns indicating an insider attack. These patterns could be consistently high anomaly scores or large anomaly score spikes. The patterns to look for will depend on the analysis. The results can then be checked against the log of insider attacks to see if the attacks were noticed. If the insider attacks were not discovered, then the weights can be adjusted to improve detection, or the formulas reconsidered. Running the analyses on several logs would be wise to get reliable results. Chapter 8 will show how the anomaly scores can be graphed to gain insights into the analysis.

Chapter 8

Example Runs

To demonstrate how the simulator provides insights into insider attack analyses and tools, the two analysis programs mentioned in the last chapter were run on logs generated by the simulator that represented increasingly complex circumstances. Twelve sets of runs were performed based twelve sets of circumstances. The first logs were run with the most basic of circumstance, and subsequent runs were modified to better match real-life scenarios.

8.1. Twelve Example Runs

For all runs, the company consisted of one department with one employee (this was done for simplicity, and so that only one graph would be needed). There was no deviation between the company, department, and employee behavior (the employee deviation for the average and standard deviation was set to 0); in other words, the employee's average and standard deviation will match the company behavior exactly.

Each log was generated over a 90 day period. Each analysis needed about 30 days worth of data to establish "normal" behavior. Because of this, all events were set to occur at a random time after 30 days had passed. The anomaly scores of the first 30 days will be generated and shown, but only scores from day 30 and on will be considered valid.

Seven actions were chosen to be monitored for the runs:

1. Email
2. Phone Calls
3. Instant Messages

4. File Access
5. Web Site Visits
6. Upload File
7. Log-in/out

These actions are detailed in appendix C, including subgroups and weights. To demonstrate the difference in results between monitoring one action and monitoring many, many of the runs were performed twice: once with only the email behavior being monitored, and once with all seven being monitored. Similarly, the subgroups and weights will be applied to some of the runs. These weights were chosen arbitrarily as rough estimates on how often each subtype took place and the relative danger of each subtype.

The default profile weights are used for both analyses, where the day, week, and month-level profiles are 1, 2, and 4 respectively. Please refer to chapter 7 for more details on how these weights are applied.

8.1.1. Run Set #1: No Events, Weighted Subtypes, or Standard Deviation

This first set of runs was performed to demonstrate the anomaly scores created if an employee has 100% consistent behavior for the duration of the entire log. During these runs, the employee performed the exact same number of actions every day. In the case of the email-only runs, the employee sent five emails every day without fail or deviation. With the others, the employee performed the same number of each action every day, with a total of 61 actions per day. As one may expect, an employee that is 100% consistent generates an anomaly score of 0 at all times, as seen in the following graphs. Figure 25 shows the results of the runs. The red charts show the actual number of occurrences while the blue charts show anomaly scores:

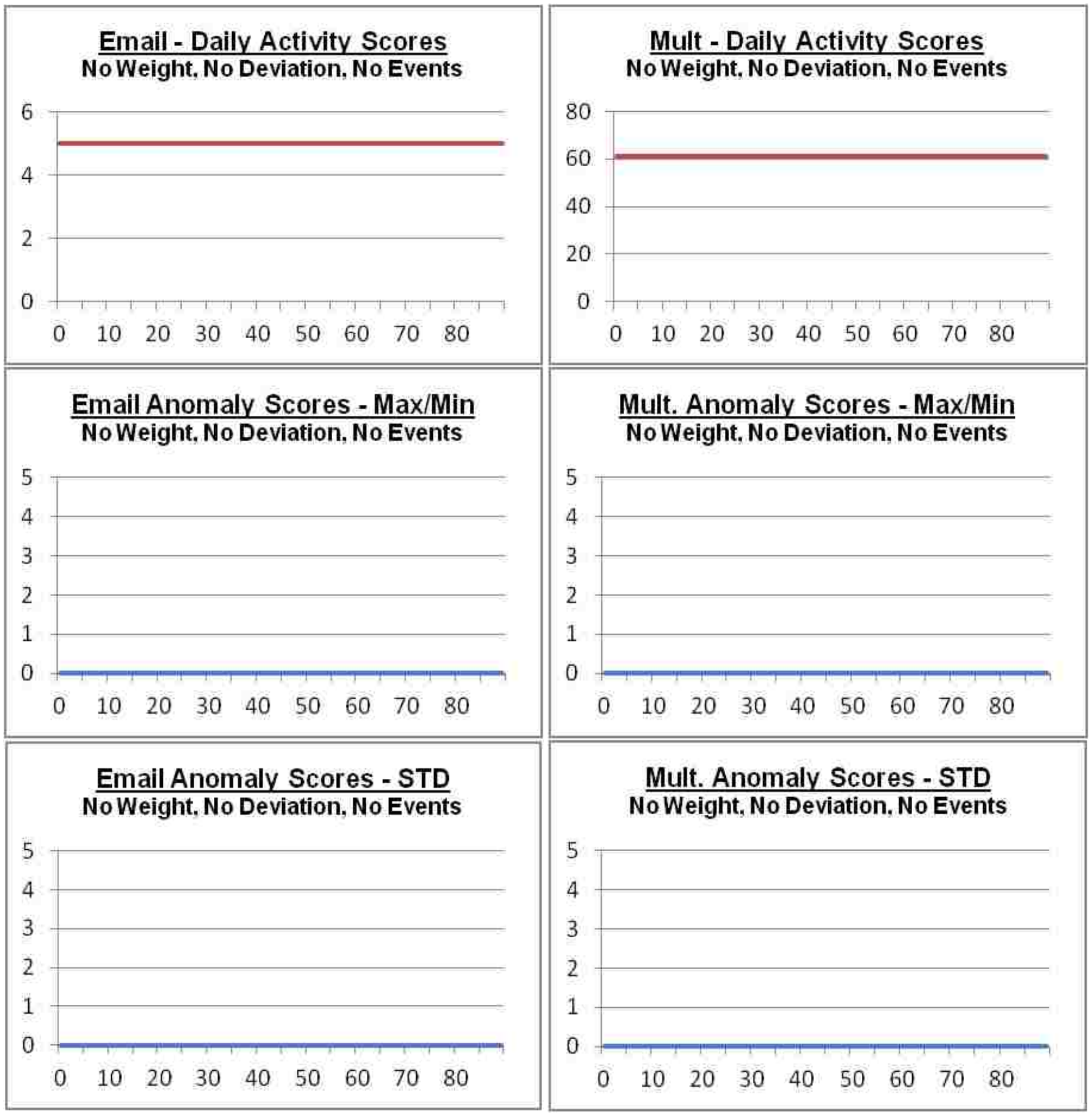


Figure 25: Anomaly scores of an employee who performs the same number of each action every day.

This run set was a control run to make sure that the analyses were not generating anomaly scores when they shouldn't exist, as well as a base case to compare the next sets of runs to. It establishes that consistent normal behavior should result in anomaly scores close to 0.

8.1.2. Run Set #2: Small Standard Deviation

The second set was very similar to set #1, but with a conservative standard deviation added to each action (Appendix C describes these standard deviations). Overall, the behavior of the employee is still reasonably consistent, and the average number of occurrences remains the same. The following charts show the results:

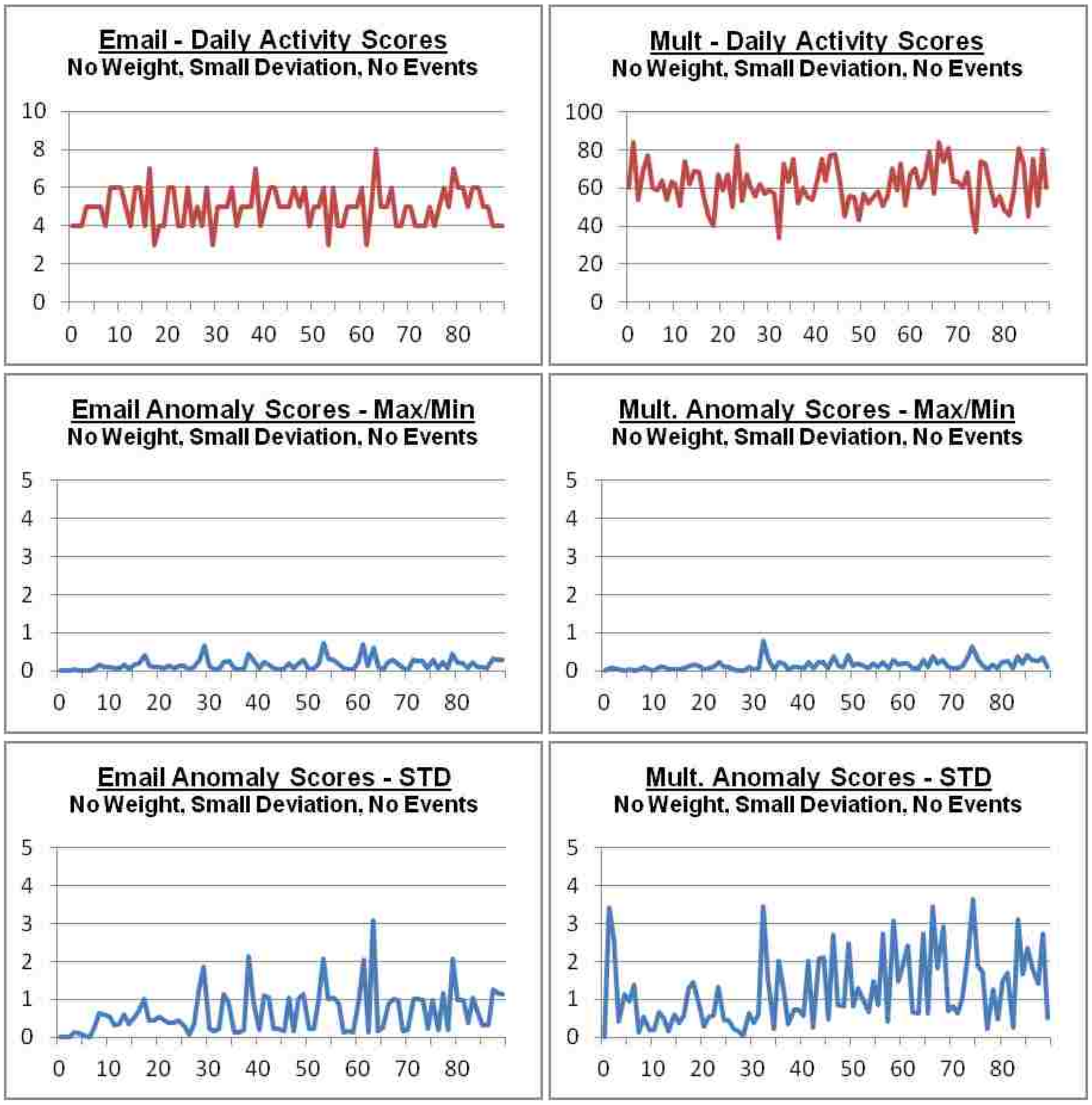


Figure 26: Occurrences and anomaly scores when there is a small deviation, no weights, and no events.

The conservative standard deviation does cause some jitter for both the max/min analysis and the standard deviation analysis. This jitter is normal and can be expected since the behavior of an employee does vary from day-to-day. The insider attacks will need to stand out from this day-to-day jitter in order for either analysis to be effective.

8.1.3. Run Set #3: Large Standard Deviation

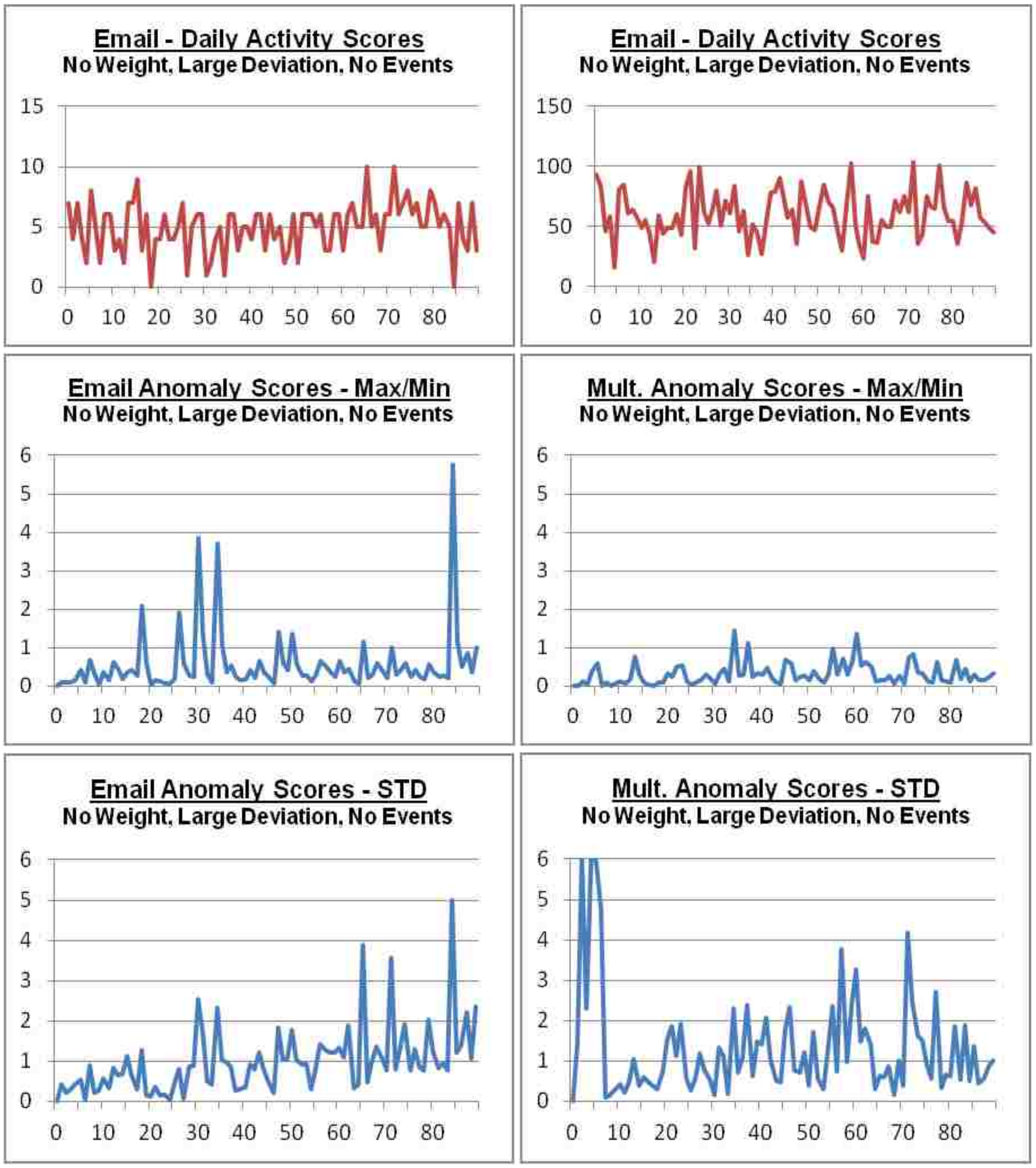


Figure 27: Occurrences and anomaly scores when there is a large deviation, no weights, and no events.

Set #3 had all of the same settings as set #2, but with the standard deviations doubled. This run is of particular interest because it exposed a weakness in the max/min analysis.

The weakness of the max/min analysis can be seen in the email-only graph on days 18 and 84. On these two particular days, the day's score is 0. When calculating the overall score, the larger score is divided by the smaller score, which in this case is 0. To avoid dividing by zero, the program adds 1 to both scores. However, this doesn't take care of the real problem: lower scores create larger jumps in the anomaly score than higher scores. The following graph illustrates the anomaly score generated by a new activity score when the activity score of all of the profiles is 5:

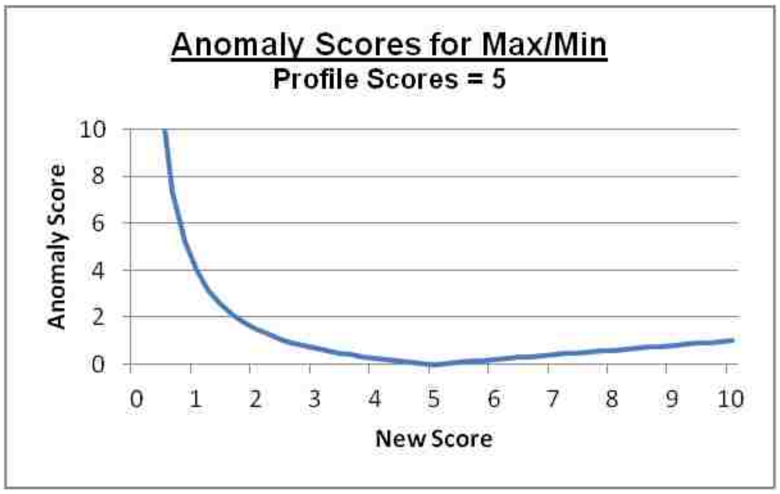


Figure 28: Graph of the anomaly score generated by the max/min analysis.

The graph shows that as the new score near zero, the anomaly score dramatically increases. As a result, performing less of an action in a day generates a greater anomaly score than an increase of equal magnitude. While this may seem appropriate for good actions (doing less of a good thing should sound more of an alarm than doing more), this doesn’t work for actions where high occurrences suggest an attack (such as uploading sensitive files).

When multiple activities are influenced, there’s a far smaller chance of the new anomaly score approaching zero. While lower scores still generate slightly higher anomaly scores, the difference isn’t nearly as influential as it was for just email alone.

8.1.4. Run Set #4: Weighted Subtypes Added and No Standard Deviation

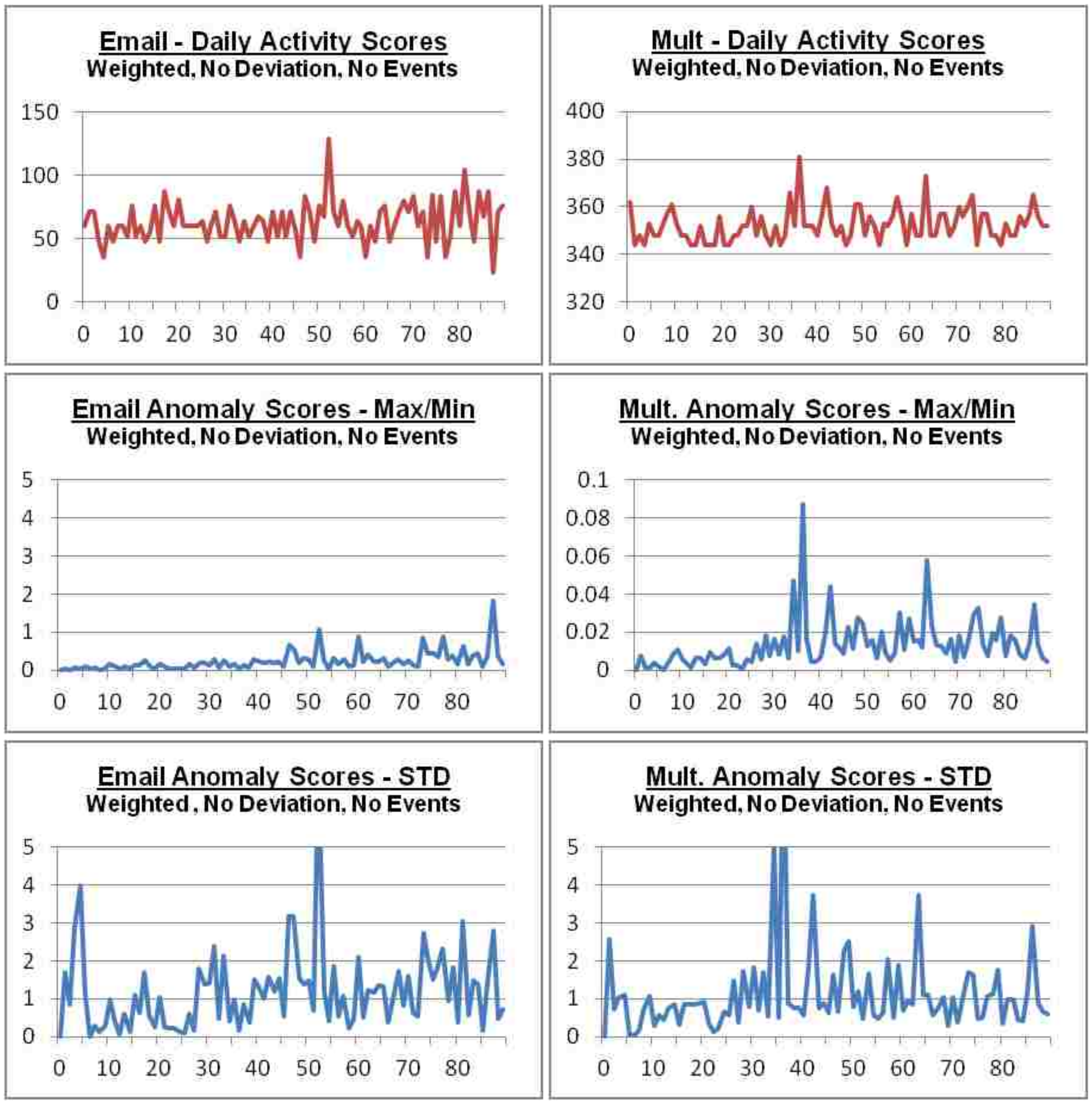


Figure 29: Graphs of the occurrences and anomaly scores when there is no deviation or events, but each instance of an action has its subgroups selected and the weight of the subgroup added.

The same number of occurrences of each action takes place in this set as occurred in run set #1 (the completely stable graph). However, for each instance of an action, a subtype is selected for each subgroup and the weight of each subtype is added to the activity score. This takes into account the fact that not all occurrences of the same action are equal – more threatening actions cause a greater increase in score than others. In the workforce, sensitive actions (ones that create high scores) do take place for valid reasons. Unfortunately for the two analyses, these valid but weighty actions generate jitter and large spikes (as seen on day 53 for the email run and days 34-36 for the all-actions run).

8.1.5. Run Set #5: Weighted Subtypes and a Small Standard Deviation

Set five combines the small standard deviation of run set #2 with the weighted subtypes of set #4. This produces variance in both the number of actions and the type of actions performed each day, similar to what occurs in an actual work environment. The graphs on the following page show the results.

It's interesting to note that the max/min analysis had significantly less jitter in it, but it did have a spike around day 55 when the daily score dropped particularly low. At this point, these graphs do not indicate how good or bad either analysis is on its own, but simply establishes what "normal" jitter appears like. The anomaly scores of insider attacks will need to be distinguishable from this jitter in order for the analyses to notice it.

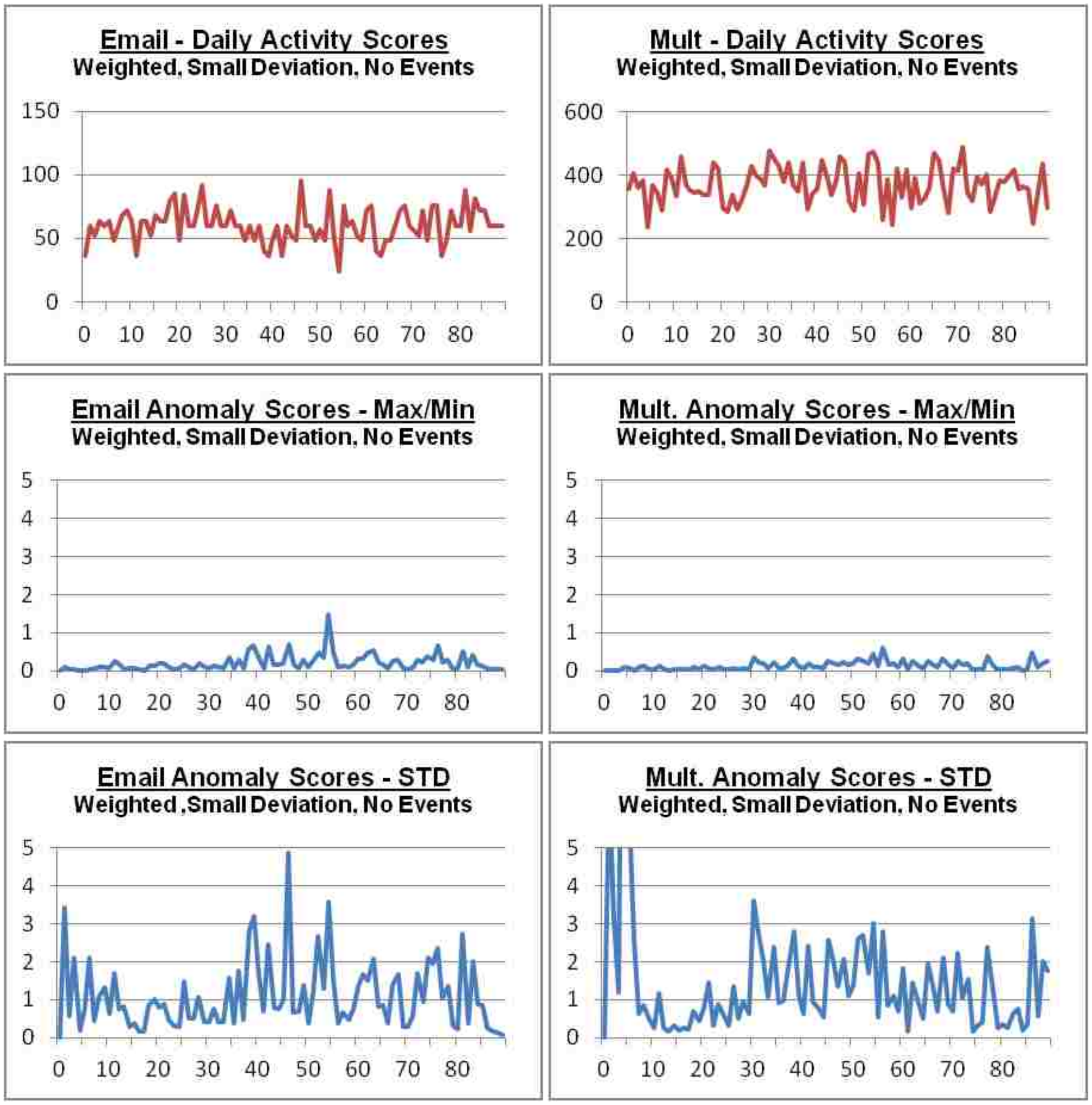


Figure 30: Occurrences and anomaly scores when there is a small standard deviation, no events, and each instance of an action has its subgroups selected and the weight of the subgroup added.

8.1.6. Run Set #6: Single Malicious Email with Weighted Subtypes but no Standard Deviation

Set #6 has the same settings as set #4, but it differs because a single event has been added. On day 36, the employee sends a highly sensitive email with a large attachment outside of the company (which may represent espionage or theft of company data). The following figure shows the results:

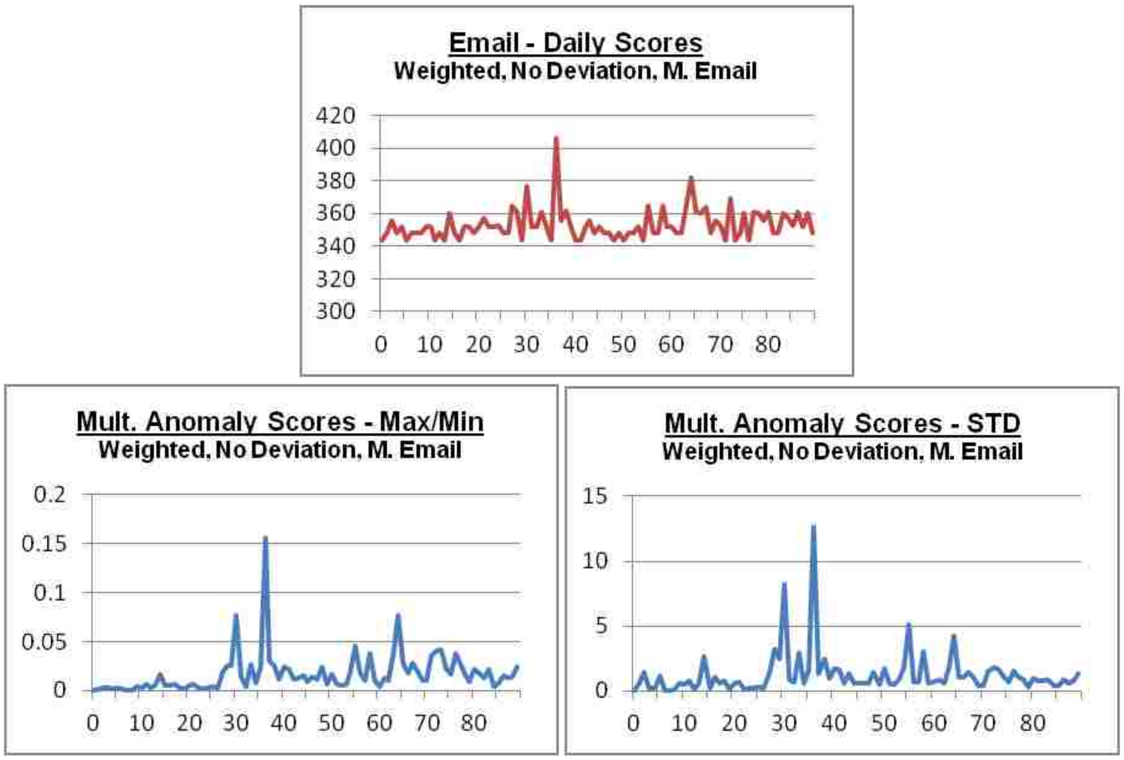


Figure 31: Daily and anomaly scores with no standard deviations and a single malicious email (highly sensitive email with a large attachment sent outside of the company) occurring on day 36.

The good news is that, with the weights associated with the high-risk subtypes, the malicious email caused a spike in the graph of the daily score. This spike was picked up by both analyses, both of which created large anomaly scores on day 36 as well. However, we've seen large spikes take place in previous sets, which means that a single malicious action may or may not be recognized from false positives.

8.1.7. Run Set #7: Single Malicious Email with a Small Standard Deviation

Set #7 has the same settings as set #6 but now the actions have a small standard deviation (the same standard deviations from set #2).

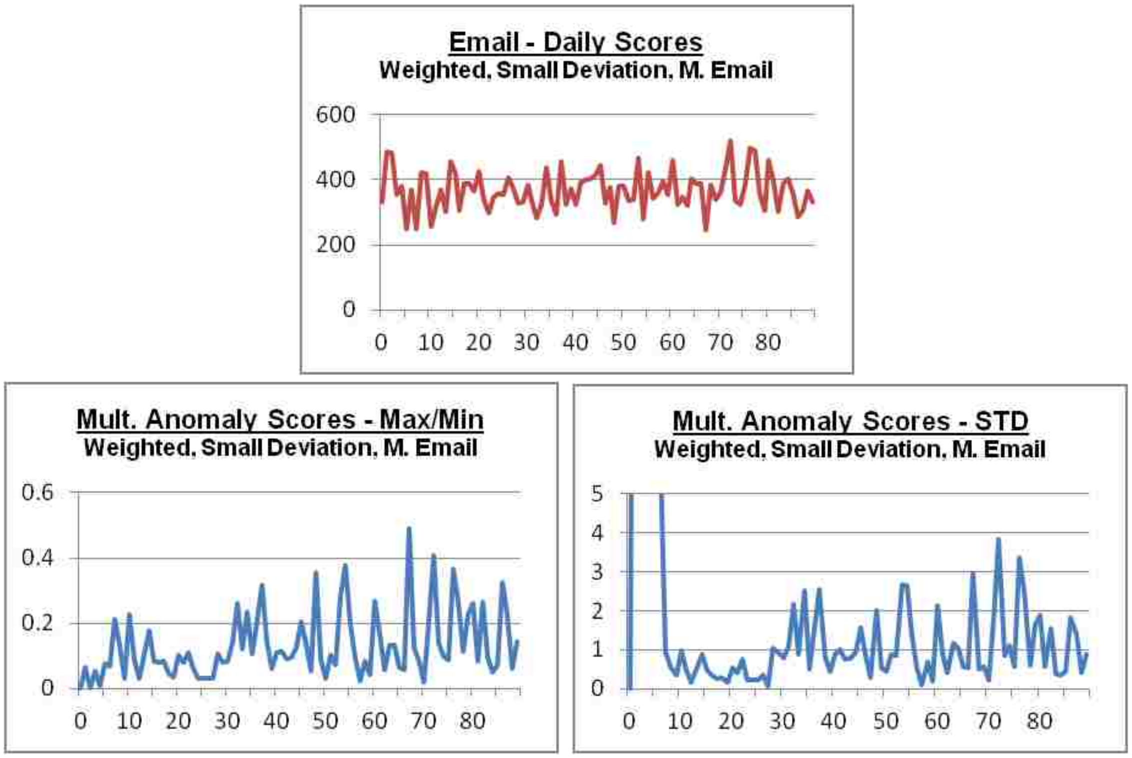


Figure 32: Daily and anomaly scores with small standard deviations and a single malicious email (highly sensitive email with a large attachment sent outside of the company) occurring on day 34.

The malicious email was sent on day 34. However, that day turned out to also be a day when little other activity took place. The low score from the low activity largely cancelled out the score increase from the malicious email. This suggests a potential flaw exists in the scoring system: opposite activity (low good activity and high bad activity) can cancel each other out and appear to be normal behavior.

8.1.8. Run Set #8: Disaffected Employee with No Weighted Subtypes or Standard Deviation

Run sets 8-11 involve an insider attack where an employee becomes disaffected, performing less over the weeks. The actual attack itself doesn't take place in this event series; the purpose of these runs is to see if the analyses can detect the insider attack based on the changes in behavior leading up to the attack alone. On day the first day of the event series, the average occurrences of all actions drop by 20% except for web sites visited, which increases by 20%. This occurs again 10 days later as well as 20 days later for a total of three drops. This represents three different events that caused the employee to work less and surf the web more. In the real world, such changes can come as a result of negative events in the workplace (a fight with a coworker or news of layoffs). Alternatively, the insider may be planning an insider attack and subconsciously starts working less over the twenty day

If there is no standard deviation or weight, each change in behavior is very clearly visible and quickly picked up by both analyses:

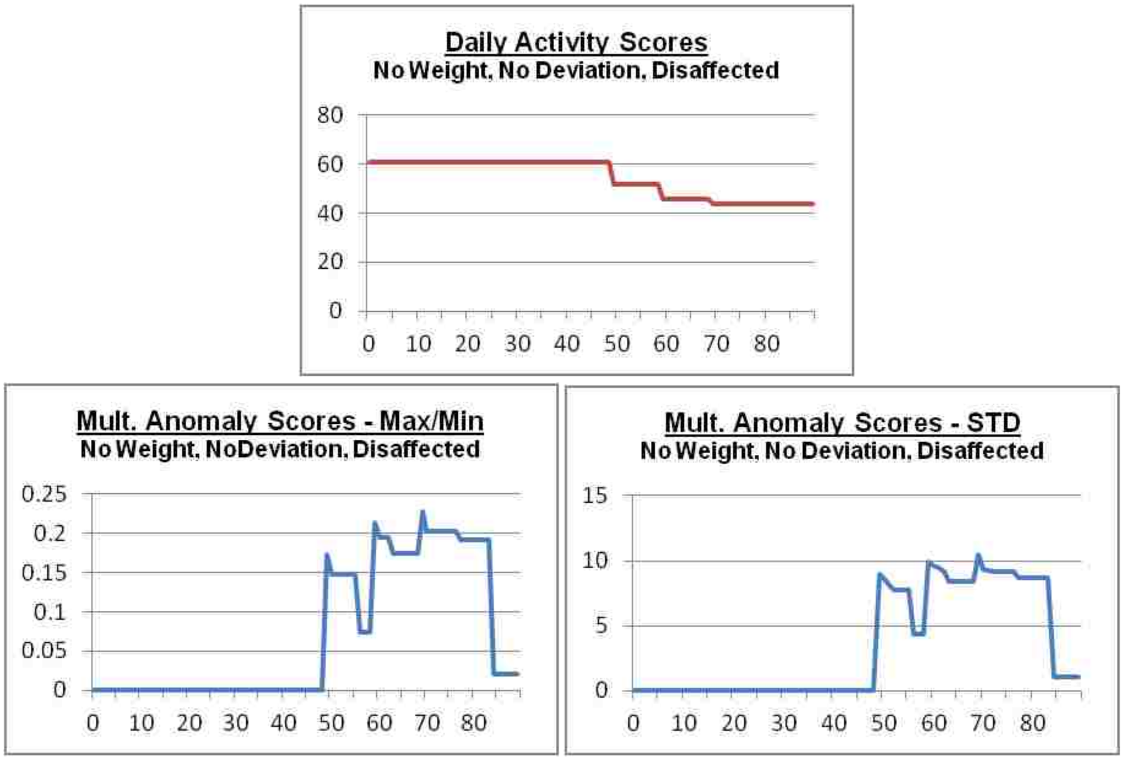
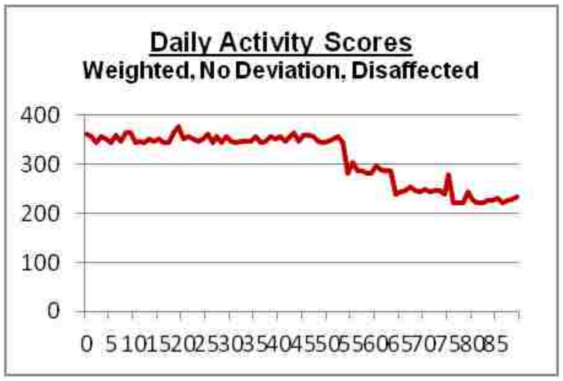
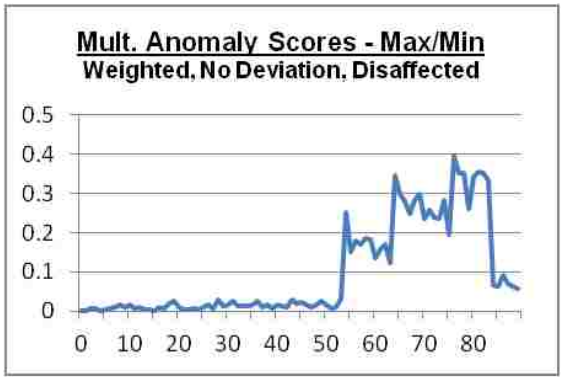


Figure 33: Employee daily and anomaly scores of a disaffected employee when no weights or standard deviation is present.

8.1.9. Run Set #9: Disaffected Employee with Weighted Subtypes and No Standard Deviation

When weights are added to the equation, a bit of jitter is generated, but the behavior changes are still very apparent in the graphs of the daily activity scores and the anomaly scores:





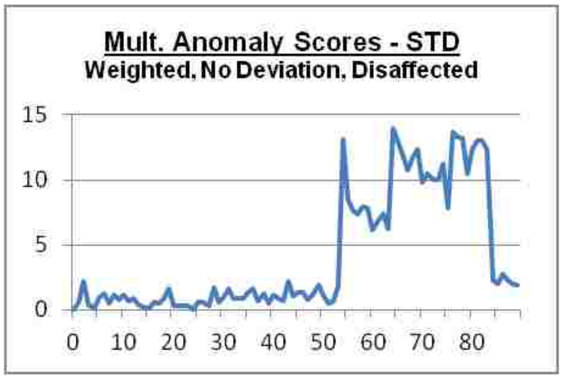


Figure 34: Employee daily and anomaly scores of a disaffected employee when weighted subtypes are selected but no standard deviation is present.

8.1.10. Run Set #10: Disaffected Employee with a Small Standard Deviation and No Weighted Subtypes

While the jitter in run set #7 didn't hamper the analyses in picking up the change in behavior, the jitter from each action having a small distribution did largely hide the abnormal changes in behavior. As can be seen in figure 35, the drop in the average behavior is visible on the daily score graphs, but isn't as apparent on the anomaly score graphs:

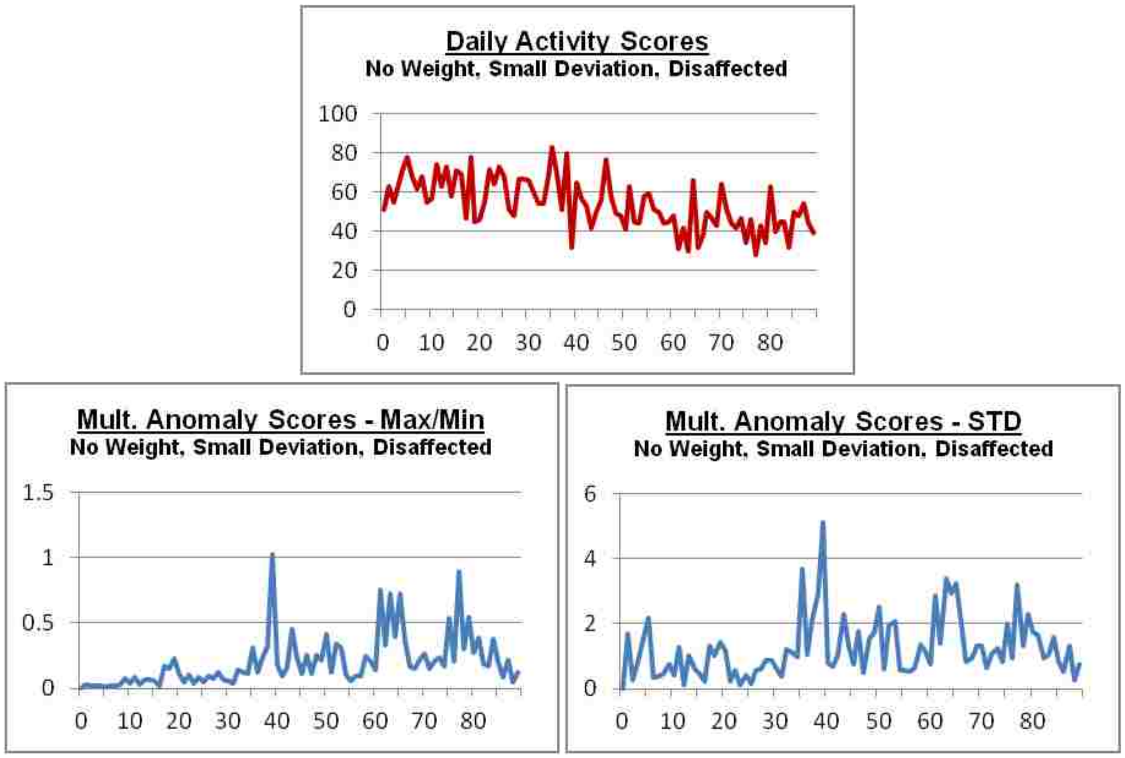


Figure 35: Daily and anomaly scores of a disaffected employee when no weights are present but each action has a small standard deviation.

The drops in behavior occurred on day 39, day 49, and day 59. Both graphs include spikes on day 39 and 59, but day 49 seems to go by without noticing. Unfortunately, these results suggest that a change in behavior could go unnoticed among the static caused by the standard deviation.

8.1.11. Run Set #11: Disaffected Employee with Weighted Subtypes and a Small Standard Deviation

The biggest hope for each analysis is that they can detect abnormal behavior in spite of the normal, reasonable variances in behavior that occur in the workplace. Set #11 has the reasonable changes in the number and type of action occurrences. The three drops in behavior occur on days 48, 58, and 68 for this run. These drops in behavior are quite visible on the actual daily scores. The anomaly scores do something interesting during this period of time – instead of simply creating spikes, the scores rise away from zero and remain away for some time after each event, creating a jittery arc. This is different from the usual jitter created by normal behavior, suggesting that the abnormal behavior could still be caught by the analyses.

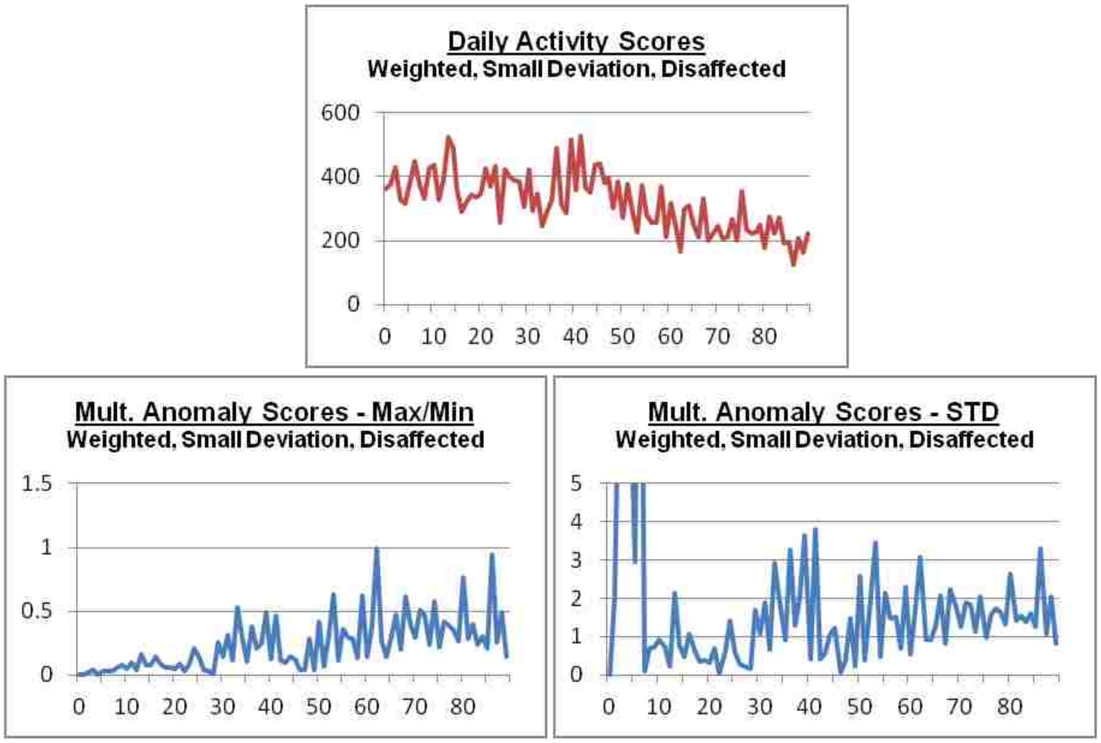


Figure 36: Daily and anomaly scores of a disaffected employee when each action has a small standard deviation and weighted subtypes.

8.1.12 Run Set #12: Sick Employee with Weighted Subtypes and a Small Standard Deviation

For completion sake, a final set of runs was performed with a false positive present. In this case, the false positive is a sick day, in which the employee doesn't perform any work at all. The following graphs show what happens:

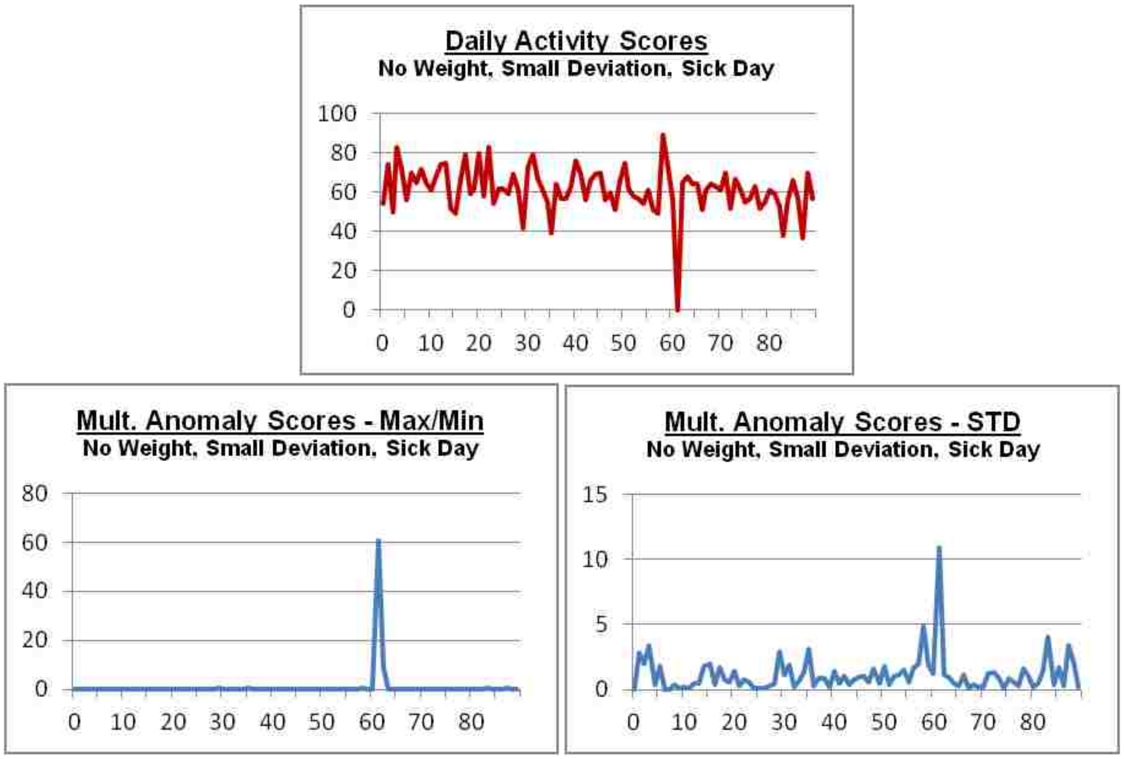


Figure 37: Daily and anomaly scores of a normal employee when each action has a small standard deviation and no weighted subtypes. The employee has one sick day on day 61.

The sick day does cause a huge anomaly score for both analyses, especially for the max/min analysis (reiterating the problem where the max/min anomaly score grows dramatically for abnormally low scores). However, the behavior goes back to normal and doesn't create the long-term increase in anomaly scores (as seen in run set #11) that suggest an insider attack. While such spikes would probably warrant investigation, the return to normal behavior would suggest this was an abnormal day and not a long-term change in behavior (which an insider attack would likely cause).

8.2. Insights from Example Run Sets

The examples just provided were designed to be a demonstration of how the simulator can be used for gaining insights into an analysis. While I made some suggestions and observations with the above graphs, they in no wise prove or disprove the effectiveness of the analyses. Some steps to take to increase the confidence in critiques of the analysis include:

1. Quantitatively set out beforehand what constitutes an insider threat. This could be a spike of a specific height, a length of time with the anomaly score above a given value, or some other metric.
2. Perform each run several times. The above examples only had a single run for each set of circumstance, and false conclusions can be arrived at without a larger pool of tests.

The ideas presented in the "Future Works" chapter could also be implemented to increase confidence in the results.

Chapter 9

Future Works

While working on the simulator and analysis programs, many ideas surfaced that could greatly improve the flexibility, accuracy, and user-friendliness of the program. The usefulness of each idea may differ depending on the analysis being used or the needs of the user. I will be breaking the future works into three categories: the simulator, the analyses, and the user interface.

9.1. Future Works for the Simulator

The simulator's current design makes it easy to implement actions where the threat level is predominantly based on the number of instances of the action. However, there are certain actions where the number of occurrences isn't the only important variable; there are actions where attributes, such as duration, intensity, or size of the action, play a large factor in determining the threat level of the action. For instance, the "time idle" action could benefit from having a "duration" attribute, where the duration of the idle time can be specified for each action instance (such as 15.3 minutes for one, and 52.1 minutes for another). The program only partially permits for actions to have attributes because the subgroup-subtype ratios and weights allow for these attributes to be recognized in categories (for instance, having a "duration" subgroup with subtypes "small", "medium", or "large"). However, these aren't as good or precise as actual numbers. Adding the ability for actions to have attributes would allow for greater accuracy in measuring each action instance's threat-level.

The simulator demonstrates how a subject or object can be selected based on the scope-of-interaction subgroup. The simulator has a very basic implementation for more of a proof of concept than for actual usefulness. Whenever an employee performs an action that has a scope-of-interaction subgroup, another person is chosen randomly based on the subtype (local, company, or outside) for subject-to-subject actions. Subject-to-object actions are even more basic since they only return the word “object”. This is because the current implementation has no list of available objects to interact with. This could be refined so that specific actions by an employee are performed with specific people or objects in the company based on the role of the person/object. To be specific, a list of people/objects can be available for each action, with each entity having a specified likeliness of being selected. Further, events could then be set so that the insider interacts with a specific person or object. Objects can be given greater detail than simply a name, including sensitivity and access rights. This idea could really be fleshed out to make the entity interacted with a strong variable in the threat-level of each action.

Behavior of an employee is currently based on company and department behavior. A possible future work would be to base the behavior of employees on their role instead of their department. This way, the team leaders of the engineering department will behave according to their roles, while senior engineers and intern engineers can have their own role-based behavior. The number of each role can be specified for each department instead of just the number of employees. Giving employees roles would also help with choosing which employees can be interacted with for specific interactions, as mentioned in the previous paragraph.

Insider attacks are represented by event series in the simulator. While the start of the event series is selected randomly by the simulator, the events within the series itself occur in the order and at the time specified by the user. Allowing for some variability to the time between each event may allow for less predictable insider attacks.

One of the events currently creatable by the simulator is the “Change Behavior” event, in which one of the employee’s behaviors immediately change as specified. For some insider attacks, a gradual change would make more sense than an immediate change. This could be provided by adding a “Change Behavior over Time” event that would allow the user to specify a change that slowly occurs over a given time. Having both gradual and immediate behavior changes would allow for greater flexibility, but some significant changes within the simulator would need to be made to make it possible.

9.2. Future Works for Analyses

While the focus of my thesis has been on the simulator and not as much as the analyses, some ideas came to mind on how the two analyses included in the program could be improved. Some of these were mentioned in the final section of the previous chapter. One of the first ideas that came to mind addressed a problem with the scoring system. As the current implementation goes, the scores of each action are added up, and then a single anomaly score in generated based on the sum. However, this meant that the abnormally low score of one behavior cancels out the abnormally high score of another behavior. One change that could address this would be to keep track of the new, daily, weekly, and monthly scores of the behaviors individually, generate an anomaly score for each behavior, and then add the anomaly scores together (as opposed to one anomaly score for the sum of activity scores).

One change worth investigating is the difference between adding and multiplying an action instance’s subtype’s weights. Currently, the activity score generated by an individual activity is 1 if it has no subtypes or the sum of the subtypes' weights. For instance, if an email action has high sensitivity (weight = 10), a large attachment (weight = 5), and is sent to someone outside the company (weight = 3), the score generated by the simulator would be 10 + 5 + 3 = 18. This means that if a similar email was sent but was sent to someone within the insider's department (weight = 1), the score would be 10 + 5 + 1 = 16 – a two point difference for a rather significant change in the danger of the email. However, if instead of adding weights together, the program multiplied them, the results of the two emails would be 10 * 5 * 3 = 150 and 10 * 5 * 1 = 50, a very large difference. The downside to multiplication is that there will be an even greater jump whenever a non-malicious but high-weight activity occurs, causing a lot more spikes. As a result, the subtle changes in behavior could be drowned out by the large spikes. The weights would definitely need to be adjusted to prevent this. However, this is speculation – implementing such a change could better measure the threat-level differences between action subtypes.

The max/min analysis can be improved by addressing the flaw where lower daily scores create abnormally large anomaly scores would need to be addressed (see figure 28 in chapter 8.1.3). The current workaround implemented in the program added 1 to both the maximum score and minimum score if the minimum score was less than 1. This could be taken further by adding a specific value to both scores irrespective of how low it is. For instance, the following figure shows the anomaly scores generated by new numbers when the profile scores are all 5.0 and a value is added to both the smaller and larger number:

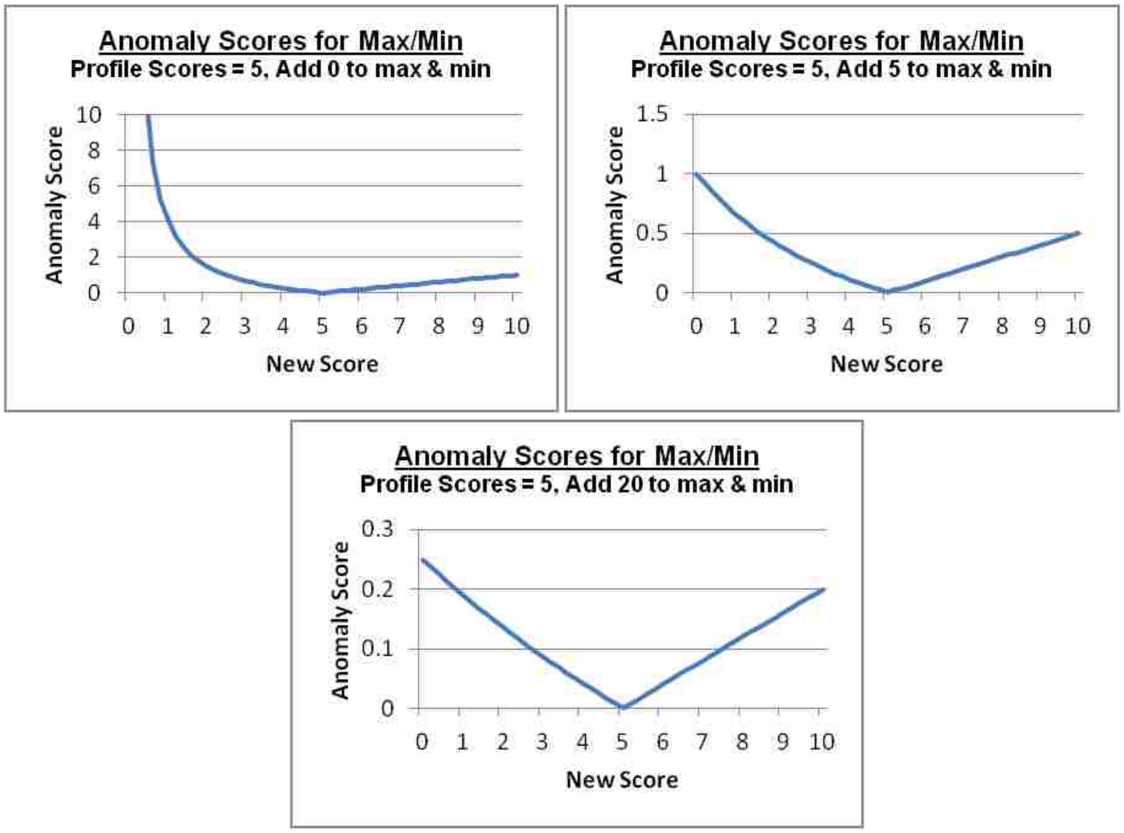


Figure 38: Anomaly scores generated by max/min analysis when all profile scores are equal 5 and an offset is added to both the larger and smaller number.

The offsets reduce the extra anomaly score for lower numbers but they don't remove it completely. The max/min formula would likely need to be modified to cause both increases and decreases in the behavior score of the same magnitude to generate the same anomaly scores.

The standard deviation analysis was developed to find a new analysis without this flaw, but it too had a similar flaw when a previous profile's standard deviations approached zero. Figure 39 shows how the size of the profile's standard deviation influences the anomaly score:

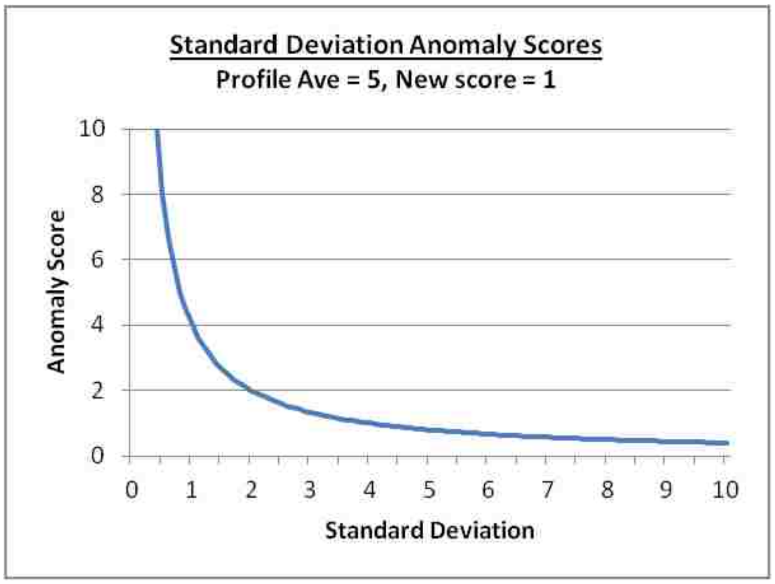


Figure 39: Anomaly score's generated by the profile's standard deviation.

Suppose someone is fairly consistent in their behavior, causing the profile standard deviations to be less than 1. If they did one extra behavior on one day, the spike such behavior would cause would overwhelm all other behavior (covering up both normal and abnormal behavior). The workaround in the program was to set the minimum each profile's standard deviation can be to 1. However, future work may produce a better means of addressing the spikes.

9.3. Future work for User Interface

The user interface was added to the simulator to make the creation of behavior, department, and event series data easier. The look of the interface definitely could be improved; the objective of getting all functionality to fit onto the interface took precedence over concerns about appearance. While I did group together similar data, the interface itself could be cleaned up to be easier on the eyes. Such a cleanup would also make it easier to find specific buttons and fields.

Three particular features stand out to me that, if added, would improve the graphic user interface:

1. A means of modifying and deleting existing, individual events within an event series
2. A better way to display department and event series data
3. A panel displaying results after an analysis has been run

The first feature is rather self-explanatory; while event series as a whole can currently be added to or deleted, the individual events currently can only be added; they cannot be modified or deleted.

The second feature would involve replacing the display boxes that show the comma-separated-value data of the department or event series with a panel that clearly shows the currently set values (included saved behavior for departments and saved events for the event series). This would greatly improve the readability of departments and event series with a lot of data. For the event series, the first and second feature could be combined so that the displayed events can be selected for modification and deletion.

The third feature would be a panel that displays the results from a run analyses. What is displayed would differ depending on the analysis performed. For the two analyses that I implemented, this panel could include graphs of the activity and anomaly scores. These and other analyses could take advantage of the event series logs that detail all event series that occurred and calculate how many of the malicious event series were detected.

Chapter 10: Conclusion

Analyses have been developed for detecting insider attacks over the past years and have been tested on existing insider attack logs and other data. However, without the ability to see if the insider attacks work under a variety of circumstances, the strengths and weaknesses of these analyses have remained untested. My simulator has allowed for the creation of logs of actions taken by a simulated group of employees. One of the simulator's greatest strengths is that is allows for the customization of a wide variety of scenarios to test analyses under. Users can customize the actions that are monitored, the behavior of the company, the size and behavior of the company's departments, the randomization of employee behavior, and the series of abnormal events (both malicious and innocent) that should take place over the time period of the log.

The benefit of such customization is that running the analyses on logs of simple actions can reveal strengths and weaknesses in analyses that may otherwise go unseen when tested under complex, real data. Testing an analysis on actual data provides limited insight – did it work or didn't it. By starting with simple logs and building up to more complicated logs that better reflect normal behavior, the user can determine the circumstances under which the analysis works effectively as well as learn the point of complexity or the specific circumstances in which the analysis struggles. A demonstration was provided of two example analyses being run on a variety of logs with the circumstances of the logs increasing in complexity. It was shown that insights can be gained fairly quickly upon running the analyses on the logs, even when the simulated data was unrealistically simple.

Overall, the simulator has made great strides in generating simulated behavior and has demonstrated its usefulness in studying analyses. Future works have been mentioned that can improve the simulator's flexibility, accuracy, detail, and ease of use. With the simulator, designers of analyses will no longer have to wait for more real data to become available, but will be able to create all of the data they need.

Appendix A

The Comma-Separated-Value Files

The data used by the simulator for defining behavior, departments, and event series is saved in comma-separated-value (CSV) files, where the data is stored with commas separating each field. To add or modify data, the user can either open and modify the CSV files directly (which can be done with a text program like Notepad or a spreadsheet program like Microsoft Excel) or take advantage of the user interface provided with the simulator. This appendix will explain how the CSV files are formatted, as well as how they're used by the program.

A.1. General CSV Format

For all of the CSV files, data is separated from one another by either commas or a new line. For instance, if I wanted the three values "Hello", 2, and "World" to be stored, both of the following would work:

```
Hello,2,World
```

or

```
Hello,2
World
```

Spaces should not be put on either side of commas; otherwise extra spaces will be added to the strings. For instance, entering the three values as "`Hello, 2, World`" would result in the three values being "Hello", "_2", and "_World" (where _ represents a blank space). The "_2" would not be interpreted as an integer of value 2, but as a line of text.

Also, the program is designed to toss out empty fields. Thus “Hello,,World,,,” would be parsed into two lines of text: “Hello” and “World”.

A.2. How Text is Interpreted

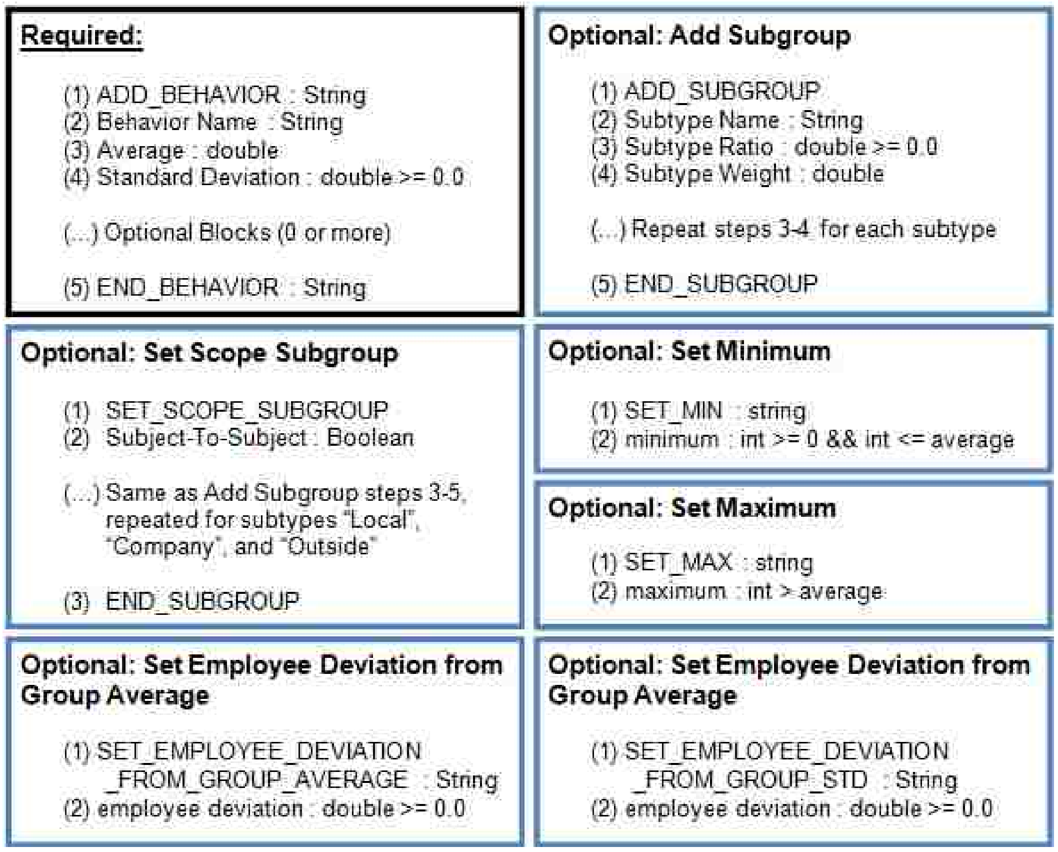
All values stored in a CSV file, including numbers, and text, are stored as text separated by commas and newlines. The program reads each value in and, depending on what it is expecting, tries to interpret the value as one of the following:

- String
    - A sequence of characters
- Command String
    - A string representing a command. Every command has a string associated with it. When a command is expected, the new value is matched against the strings of all commands that would be valid at the time. If a match isn’t found, an error is thrown. These commands and their associated strings are set in the program’s enumeration files.
    - For example, suppose the ADD and SUBTRACT commands’ strings are “#Add” and “#Subtract”. If one of these two commands is expected by the program, then it will read in the next value as a string and compare it to “#Add” or “#Subtract”. If neither one matches the string, then an error is thrown. Otherwise, the string is interpreted as its matching command (so the “#Subtract” string would be interpreted as the command SUBTRACT).
- Integer
    - A whole number; a number without a decimal point (5, -13, 0)

- Double
    - A floating-point number; a number with a decimal point (5.1, -13.8, 0.0)
- Boolean
    - Either the value TRUE or FALSE

A.3. Company Behavior CSV File

When reading a CSV file, the program reads data from the file and compares it to what is expected. If the program is at a point where optional data can be entered, it sees if the next data matches an optional command or the next required line. If the optional command is encountered, then the program keeps reading in data until all expected data for that option is gathered, and then it again checks if the next line is an optional command or the next required line.



Required:

(1) ADD_BEHAVIOR : String
(2) Behavior Name : String
(3) Average : double
(4) Standard Deviation : double >= 0.0

(...) Optional Blocks (0 or more)

(5) END_BEHAVIOR : String

Optional: Add Subgroup

(1) ADD_SUBGROUP
(2) Subtype Name : String
(3) Subtype Ratio : double >= 0.0
(4) Subtype Weight : double

(...) Repeat steps 3-4 for each subtype

(5) END_SUBGROUP

Optional: Set Scope Subgroup

(1) SET_SCOPE_SUBGROUP
(2) Subject-To-Subject : Boolean

(...) Same as Add Subgroup steps 3-5, repeated for subtypes "Local", "Company", and "Outside"

(3) END_SUBGROUP

Optional: Set Minimum

(1) SET_MIN : string
(2) minimum : int >= 0 && int <= average

Optional: Set Maximum

(1) SET_MAX : string
(2) maximum : int > average

Optional: Set Employee Deviation from Group Average

(1) SET_EMPLOYEE_DEVIATION _FROM_GROUP_AVERAGE : String
(2) employee deviation : double >= 0.0

Optional: Set Employee Deviation from Group Average

(1) SET_EMPLOYEE_DEVIATION _FROM_GROUP_STD : String
(2) employee deviation : double >= 0.0

Figure 40: Order of data expected in the company behavior CSV file.

To write a behavior CSV file, start by providing the required data (steps 1-4 in figure 40's box labeled "REQUIRED"). At the location of "(…) Optional Commands", you can insert 0 or more of the optional blocks of data. For instance, the following is an example of how a behavior with two optional blocks of data would be formatted in a CSV file (all strings starting with '#' are string commands):

```
#AddBehavior,Email,5.0,1
#AddSubgroup,Sensitivity
None,100,1
Low,1,3
Medium,0.1,6
High,0.001,20
#EndSubgroup
#SetMax,12
#EndBehavior
```

Figure 41: Example of comma-separated-value data for adding the behavior "Email".

In this example, each optional data block is surrounded by a blue box to show where it starts and end. The program would read each data value, compare it to what's expected, and continue to the next.

A demonstration of how this would be done for the example in figure 41 follows. The first expected value is "(1) ADD_BEHAVIOR : String", or the "ADD_BEHAVIOR" command. The program would read the string "#AddBehavior" and compare it to "ADD_BEHAVIOR" command's string (in this case, they match). The program will then move to the next expected value, "(2) Behavior Name : String". It reads the string "Email" and saves it as the department name. The next expected value is "(3) Average :

double", so it would read "5.0" and parse it as a double value (throwing an exception if the text couldn't be interpreted as a floating-point number). It would similarly read in "1" for "(4) Standard Deviation : double >= 0.0", throwing an exception if it can't be parsed as a floating-point number or if the parsed value isn't greater-than-or-equal-to 0.0.

At this point, the program is going to see if an a string matching an optional command is encountered or if a string matching "(5) END_BEHAVIOR : String" is encountered. An optional command is encountered, and so it goes and interprets the expected values of the optional block in the same manner the required values were. It repeats until "(5) END_BEHAVIOR : String" is encountered, then ends.

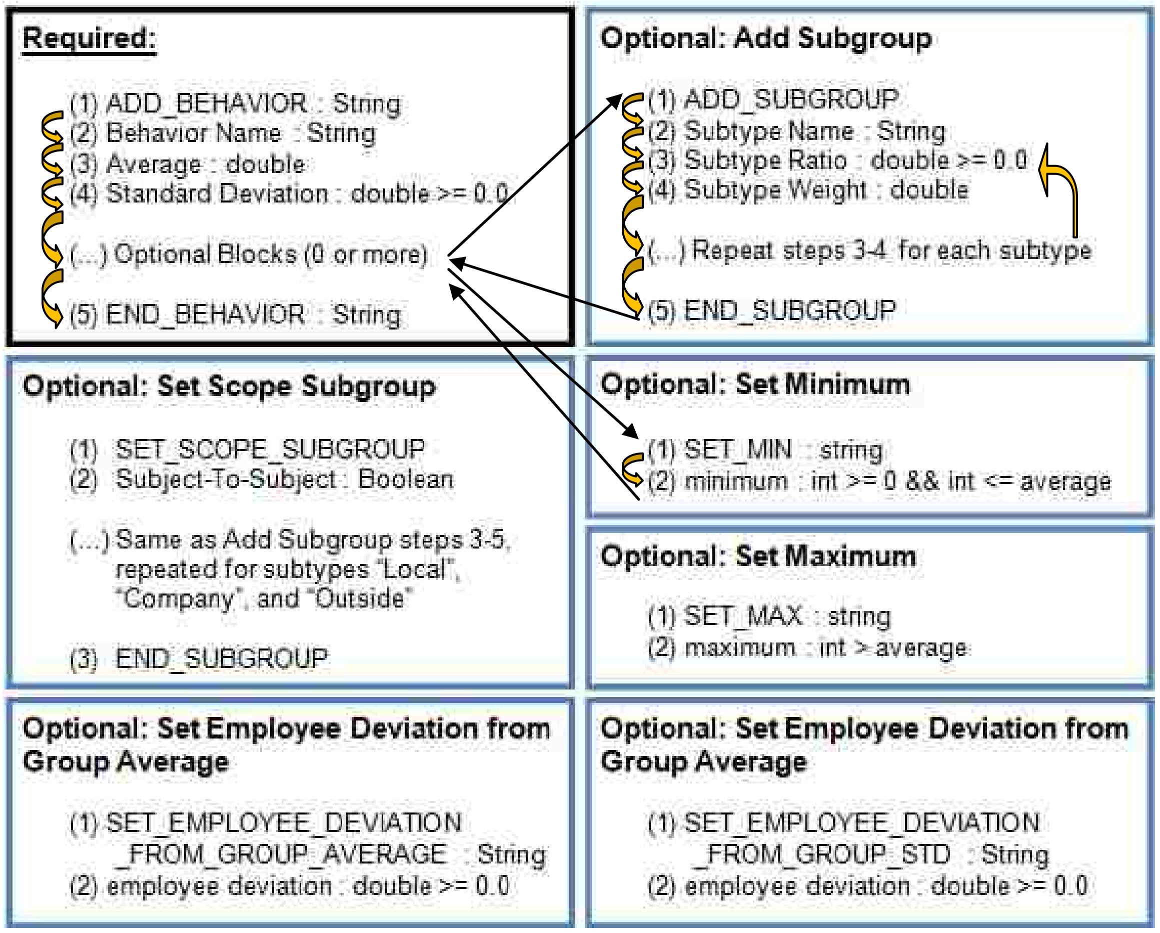
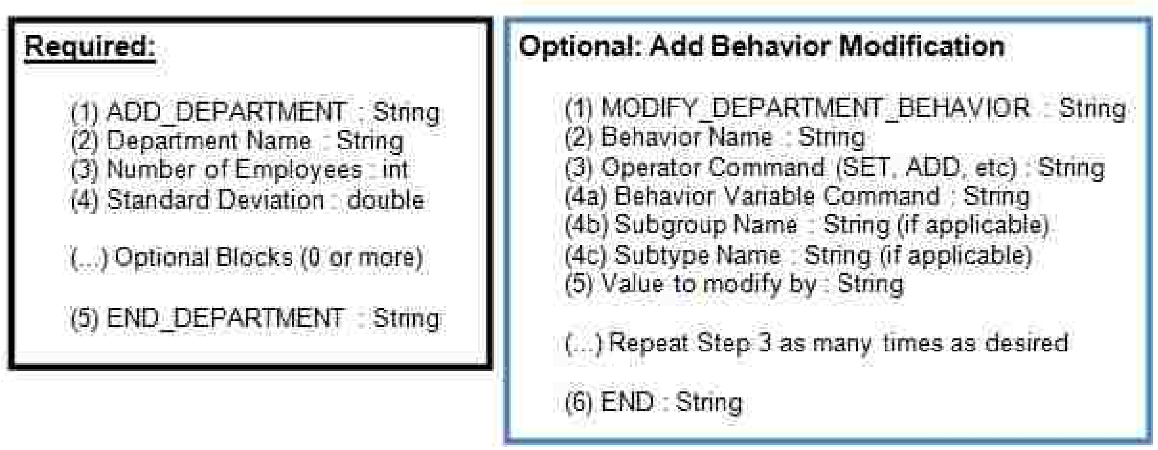

Required:
(1) ADD_BEHAVIOR : String
(2) Behavior Name : String
(3) Average : double
(4) Standard Deviation : double >= 0.0
(...) Optional Blocks (0 or more)
(5) END_BEHAVIOR : String
Optional: Add Subgroup
(1) ADD_SUBGROUP
(2) Subtype Name : String
(3) Subtype Ratio : double >= 0.0
(4) Subtype Weight : double
(...) Repeat steps 3-4 for each subtype
(5) END_SUBGROUP
Optional: Set Scope Subgroup
(1) SET_SCOPE_SUBGROUP
(2) Subject-To-Subject : Boolean
(...) Same as Add Subgroup steps 3-5, repeated for subtypes "Local", "Company", and "Outside"
(3) END_SUBGROUP
Optional: Set Minimum
(1) SET_MIN : string
(2) minimum : int >= 0 && int <= average
Optional: Set Maximum
(1) SET_MAX : string
(2) maximum : int > average
Optional: Set Employee Deviation from Group Average
(1) SET_EMPLOYEE_DEVIATION _FROM_GROUP_AVERAGE : String
(2) employee deviation : double >= 0.0
Optional: Set Employee Deviation from Group Average
(1) SET_EMPLOYEE_DEVIATION _FROM_GROUP_STD : String
(2) employee deviation : double >= 0.0

Figure 42: Illustration of how the example in figure x traversed the expected data.

A.4. Department CSV File

The following figure presents the expected data for a CSV file:



Required:

(1) ADD_DEPARTMENT : String
(2) Department Name : String
(3) Number of Employees : int
(4) Standard Deviation : double

(...) Optional Blocks (0 or more)

(5) END_DEPARTMENT : String

Optional: Add Behavior Modification

(1) MODIFY_DEPARTMENT_BEHAVIOR : String
(2) Behavior Name : String
(3) Operator Command (SET, ADD, etc) : String
(4a) Behavior Variable Command : String
(4b) Subgroup Name : String (if applicable)
(4c) Subtype Name : String (if applicable)
(5) Value to modify by : String

(...) Repeat Step 3 as many times as desired

(6) END : String

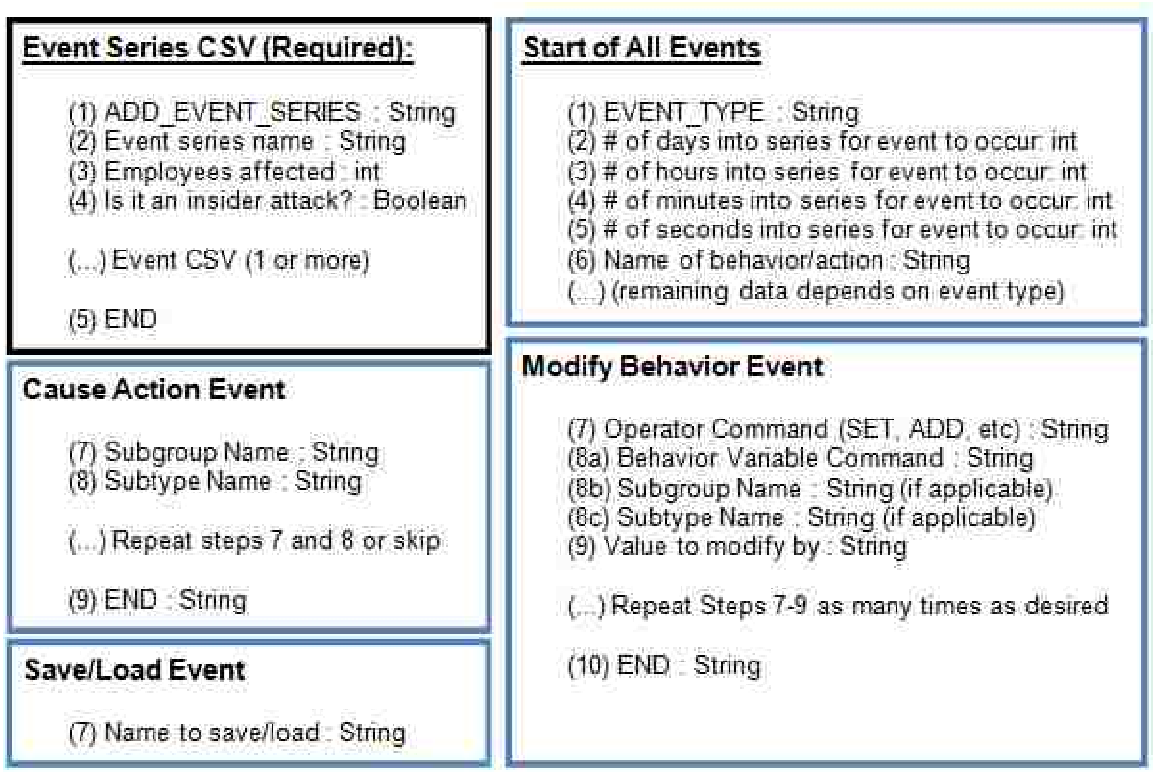
Figure 43: Order of data expected in the department CSV file.

The order of data for department CSV data is similar to the order of data for the company behavior CSV data. The department starts off with four required pieces of data, and then any number of behavior modifications can be entered. 4b and 4c are only included under the following conditions:

1. The behavior variable (or the aspect of the behavior being modified) is an aspect of one of the behavior's subgroups (such as subgroup standard deviation). If so, include the subgroup (4b) to modify but leave out the subtype (4c).
2. The behavior variable is an aspect of one of the behavior's subtypes (such as subtype ratio). If so, include the subgroup (4b) and subtype (4c).

A.5. Event Series CSV File

The event series begins with the even name, number of affected employees, and either a Boolean value representing whether the event series represents an insider attack (with the value "true" being an insider attack). From there, one or more events are given. Each event, regardless of its type, starts the same way. The command specifying the type


Event Series CSV (Required):

(1) ADD_EVENT_SERIES : String
(2) Event series name : String
(3) Employees affected : int
(4) Is it an insider attack? : Boolean

(...) Event CSV (1 or more)

(5) END

Cause Action Event

(7) Subgroup Name : String
(8) Subtype Name : String

(...) Repeat steps 7 and 8 or skip

(9) END : String

Save/Load Event

(7) Name to save/load : String

Start of All Events

(1) EVENT_TYPE : String
(2) # of days into series for event to occur: int
(3) # of hours into series for event to occur: int
(4) # of minutes into series for event to occur: int
(5) # of seconds into series for event to occur: int
(6) Name of behavior/action : String
(...) (remaining data depends on event type)

Modify Behavior Event

(7) Operator Command (SET, ADD, etc) : String
(8a) Behavior Variable Command : String
(8b) Subgroup Name : String (if applicable)
(8c) Subtype Name : String (if applicable)
(9) Value to modify by : String

(...) Repeat Steps 7-9 as many times as desired

(10) END : String

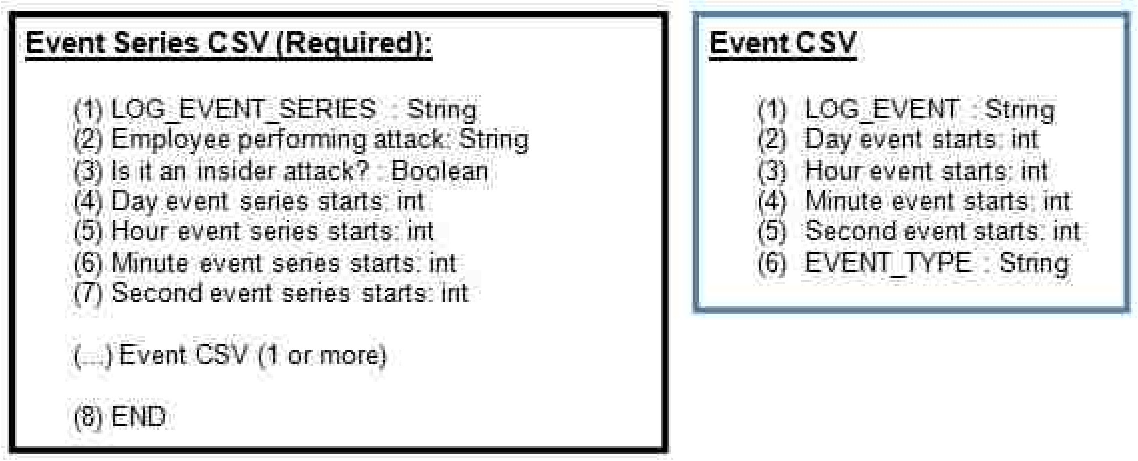
Figure 44: Order of data expected for an event series.

of event is given, followed by when the event takes place and what behavior is involved. After that, the data unique to each type of event is given. The cause action event allows for the subtypes of the caused action to be specified (one subtype for each subgroup), although no subtypes have to be given. The modify behavior event is parsed in a manner very similar to the department-level behavior modification. Finally, for both the load and save events, the only extra information needed is the name the behavior should be loaded or saved under.

Appendix B

CSV Output Files

B.1. Event Series Log CSV Output File



Event Series CSV (Required):

(1) LOG_EVENT_SERIES : String
(2) Employee performing attack: String
(3) Is it an insider attack? : Boolean
(4) Day event series starts: int
(5) Hour event series starts: int
(6) Minute event series starts: int
(7) Second event series starts: int

(...) Event CSV (1 or more)

(8) END

Event CSV

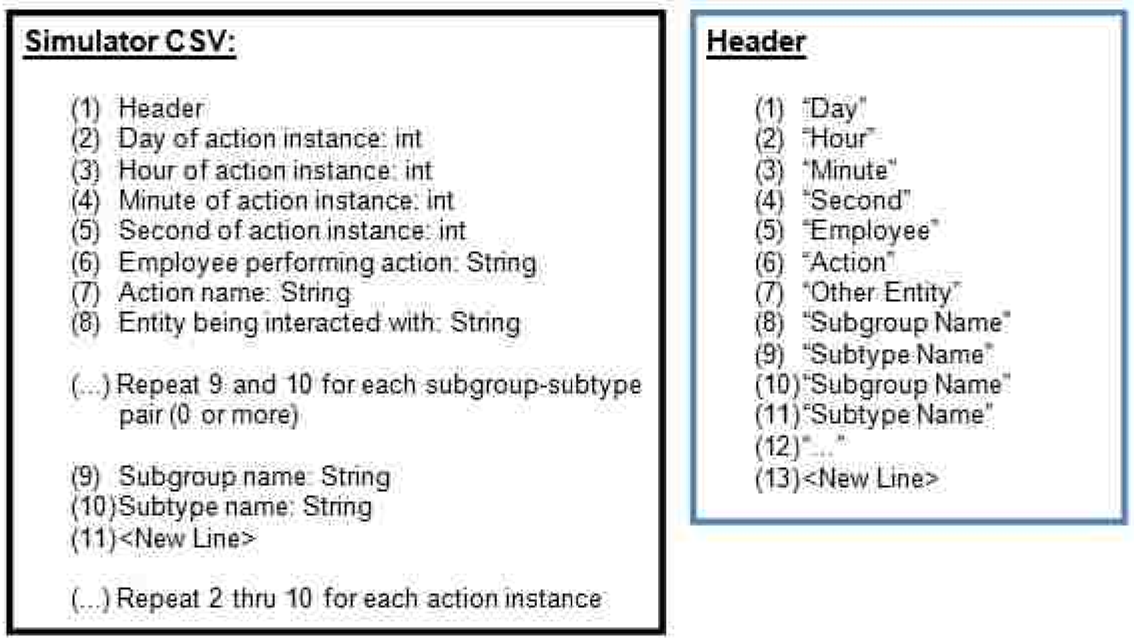
(1) LOG_EVENT : String
(2) Day event starts: int
(3) Hour event starts: int
(4) Minute event starts: int
(5) Second event starts: int
(6) EVENT_TYPE : String

Figure 45: Order of event series log data when written to an output file.

Once all of the event series have been applied to employees and their times are set, the program outputs each event series to the event series log CSV output file. This file is used to check when each event series took place and who did it. It also allows the user to verify if the analyses were able to detect all of the insider attacks as well as if the analyses had any false positives.

B.2 Simulator Log CSV Output File

The simulator log is unique in that it starts with a header line, and all of the data for each action instance is written on one line each (see figure 46). For each action instance, the time data occurs first, followed by the employee who performed the action, the action name, and the entity the employee interacted with (either a person or the string "Object"). For each subgroup-subtype pair, the subgroup name is written followed by the subtype. This is repeated for all pairs, and then a new line is added to signal the end of the action instance.



Simulator CSV:

(1) Header
(2) Day of action instance: int
(3) Hour of action instance: int
(4) Minute of action instance: int
(5) Second of action instance: int
(6) Employee performing action: String
(7) Action name: String
(8) Entity being interacted with: String

(...) Repeat 9 and 10 for each subgroup-subtype pair (0 or more)

(9) Subgroup name: String
(10) Subtype name: String
(11) <New Line>

(...) Repeat 2 thru 10 for each action instance

Header

(1) "Day"
(2) "Hour"
(3) "Minute"
(4) "Second"
(5) "Employee"
(6) "Action"
(7) "Other Entity"
(8) "Subgroup Name"
(9) "Subtype Name"
(10) "Subgroup Name"
(11) "Subtype Name"
(12) "..."
(13) <New Line>

Figure 46: Order of data written to the simulator log CSV output file.

Appendix C

Tables of Input Behavior

Behavior		**Average**	**Small Std**	**Large Std**
Email		5	1	2
Subgroup		**Subtype**	**Ratio**	**Weight**
Scope	Local		10	1
	Company		3	3
	Outside		1	10
Sensitivity	None		20	1
	Low		1	5
	Medium		0.1	10
	High		0.01	30
Attachments	No Attachment		20	1
	Small Attachment		1	5
	Large Attachment		0.1	10

Table 1: Details of "Email" action and behavior.

Behavior		**Average**	**Small Std**	**Large Std**
Phone Calls		8	2	4
Subgroup		**Subtype**	**Ratio**	**Weight**
Scope	Local		3	1
	Company		2	2
	Outside		1	3

Table 2: Details of "Phone Call" action and behavior.

Behavior		**Average**	**Small Std**	**Large Std**
Instant Messages		13	3	6
Subgroup		**Subtype**	**Ratio**	**Weight**
Scope	Local		3	1
	Company		2	2
	Outside		1	5

Table 3: Details of "Instant Messages" action and behavior.

Behavior	**Average**	**Small Std**	**Large Std**
File Access	20	8	16
Subgroup	**Subtype**	**Ratio**	**Weight**
Scope	Local	3	1
	Company	2	2
	Outside	1	5
Sensitivity	None	20	1
	Low	1	5
	Medium	0.1	10
	High	0.01	30

Table 4: Details of "File Access" action and behavior.

Behavior	**Average**	**Small Std**	**Large Std**
Web Site Access	10	4	12
Subgroup	**Subtype**	**Ratio**	**Weight**
Scope	Local	0	1
	Company	2	1
	Outside	1	3

Table 5: Details of "Web Site Access" action and behavior.

Behavior	**Average**	**Small Std**	**Large Std**
File Uploads	2	2	4
Subgroup	**Subtype**	**Ratio**	**Weight**
Scope	Local	3	1
	Company	2	2
	Outside	1	5
Sensitivity	None	20	1
	Low	1	5
	Medium	0.1	10
	High	0.01	30

Table 6: Details of "Web Site Access" action and behavior.

Behavior	**Average**	**Small Std**	**Large Std**
Log In/Out	3	1	2
Subgroup	**Subtype**	**Ratio**	**Weight**
Scope	Local	50	1
	Company	10	3
	Outside	1	10
Abnormal Time	On Time	50	1
	Abnormal Time	1	10

Table 7: Details of "Log In/Out" action and behavior.

Bibliography

Ali, G., Shaikh, N. A., & Shaikh, Z. A. (2008). Towards an automated multiagent system to monitor user activities against insider threat. *Biometrics and Security Technologies, 2008. ISBAST 2008. International Symposium on,* 1-5.

Cappelli, D., Moore, A., Trzeciak, R., & Shimeall, T. J. (2009). Common sense guide to prevention and detection of insider threats 3rd edition – version 3.1. *CMU SEI,*

Killourhy, K. S., & Maxion, R. A. (2007). Toward realistic and artifact-free insider-threat data. *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual,*87-96.

Kim, Y., Sheldon, F., & Hively, L. M. (2011). Anomaly detection in multiple scale for insider threat analysis. *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research,* Oak Ridge, Tennessee. 77:1-77:1. doi: 10.1145/2179298.2179386

Liu, A., Martin, C., Hetherington, T., & Matzner, S. (2005). A comparison of system call feature representations for insider threat detection. *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC,* 340-347.

Montelibano, J., & Moore, A. (2012). Insider threat security reference architecture. *System Science (HICSS), 2012 45th Hawaii International Conference on,* 2412-2421.

Pramanik, S., Sankaranarayanan, V., & Upadhyaya, S. (2004). Security policies to mitigate insider threat in the document control domain. *Computer Security Applications Conference, 2004. 20th Annual,* 304-313.

Zhang, H., Ma, J., Wang, Y., & Pei, Q. (2009). An active defense model and framework of insider threats detection and sense. *Information Assurance and Security, 2009. IAS '09. Fifth International Conference on, , 1* 258-261.

Zhang, T., & Zhao, P. (2010). Insider threat identification system model based on rough set dimensionality reduction. *Software Engineering (WCSE), 2010 Second World Congress on, , 2* 111-114.

Vita

Graduate College

University of Nevada, Las Vegas

Chris Clark

Degrees:

Master of Science in Computer Science, Pending

University of Nevada, Las Vegas

Bachelor of Science in Computer Science, 2011

Brigham Young University, Idaho

Projects:

Class Scheduler using Genetic Algorithms, 2011

Analyzer of Oil Trades, 2011

Simulation of Particles in Chaos Theory, 2005

Teaching Experience:

Lab Instructor for Freshman Computer Science Students, 2012