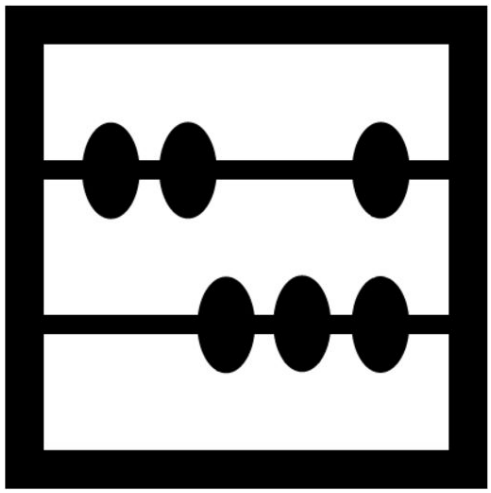


ABACUS

(Anomalous Behavior Analysis for Corporate User Systems)

Team May1613

Zachary Bales, Claude Cullen, Douglas Johnson, Fata Nadarevic, Eric Roberts, Alexander White

TABLE OF CONTENTS

1 Introduction

1.1 Problem Statement

PricewaterhouseCoopers (PWC) presented us with the challenge of creating a system for use in the corporate computing world that detects unusual user behavior. This would be used as a way to mitigate security issues and potentially head off problem behavior before it became a big issue. PWC requested that we also address mitigating false positive detection, which is something that is a constant issue for any security monitoring program.

PWC wanted this system to have a way to engage and educate users on how risky their digital behavior is. This would be in the form of user education. Their key concept was to have a "security score" similar to a credit score to keep an individual informed as to their security status. Users may be unintentionally taking part in bad behavior, and letting them know where they are going astray will improve the overall security of a system. Given these few guidelines, PWC let us define the parameters of the system. And what follows is our solution to the given design challenge.

1.2 Our Solution

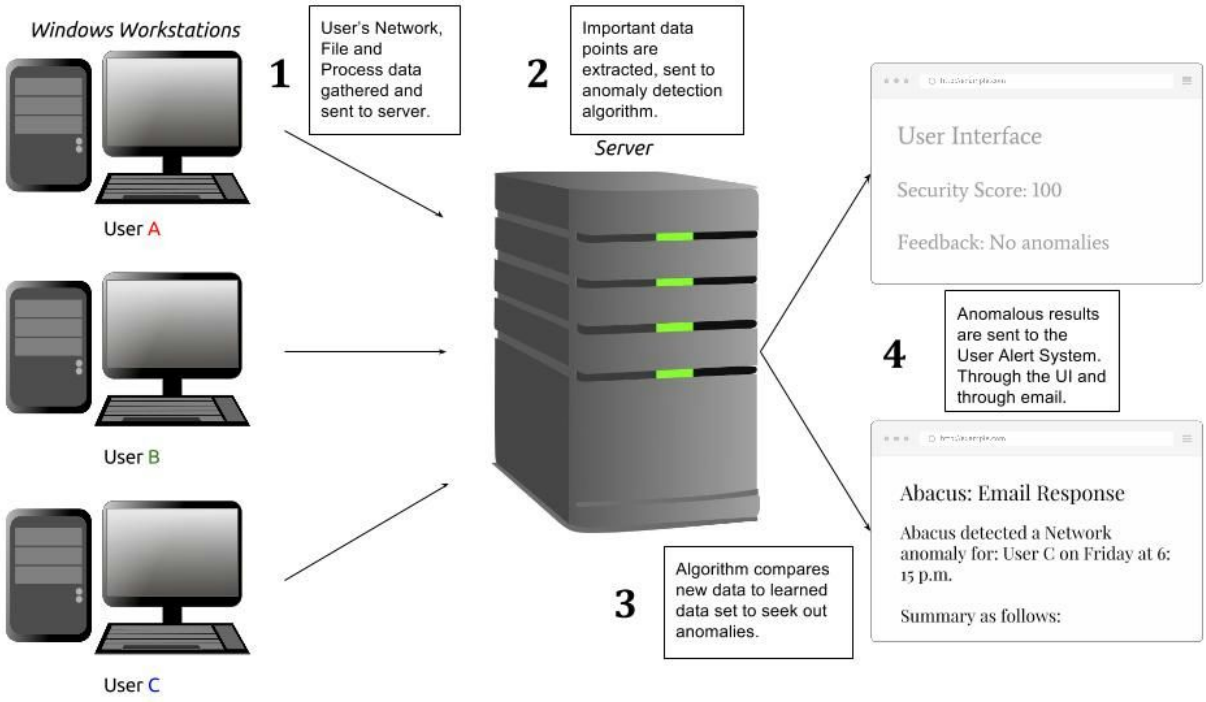
We decided that our program would follow the client / server architecture model in order to monitor multiple user's systems in a typical work environment. A user feedback system with the "user security score" keeps users informed about their security status via emails, and helps to mitigate false positives.

Three main types of data are gathered from the users of the system, file, network, and process system activity. This data is analyzed on the server side after features are extracted from it. The data is learned, stored on a database, and compared against new data through a machine learning algorithm to detect anomalies.

2 Project Design Analysis

2.1 Functional Decomposition

Our anomalous user activity detection framework is split into two main subsystems: the server subsystem, and the client subsystem. The client subsystem is responsible for logging system usage data and sending this data to the server subsystem. The server subsystem then proceeds to analyze the data in order to generate a profile for the user and detect anomalous behavior, which shall be defined more rigorously later in this document. When anomalous behavior is detected the server notifies the user and the system administrator.



2.2 Client System Modules

The client side of the program is installed on to the user's computer at their organization and works unseen in the background monitoring the user's system and activities. There is feedback to the user about their behavior to encourage secure computer use and notifications to inform the user and others of security issues. The main systems of the client side include three Data

Collection modules (for file, network and process data), and the data sender. A breakdown of their operations follows:

2.2.1 Data Collection

As stated previously, data collection on the client side involves different modules that observe different aspects of a user's activity. This is one of the main parts of our program that observes user activity, and it was important that it be unobtrusive, but also capable of overseeing most, if not all, of the computer's activities. Because security risks come in many forms, it is essential to gather as much useful data as possible to facilitate the eventual analysis of that data, so the program can yield more accurate results in the event of abnormal behavior.

The data collection module is split into three submodules: file data collection, network data collection, and process data collection. Each submodule is responsible for collecting its respective type of data.

- **File Data Collection Module** - This submodule collects data about the user's interaction with files. This includes how a file was changed (whether it was changed, created, renamed, or deleted), the name of the file that was changed, and the time it was changed. We collect this data whenever there is a file create, delete, update, or rename event on the system.

- **Network Data Collection Module** - This submodule is responsible for collecting data related to network usage on the system. It collects information about packets that are sent and received including packet size, IP address, and sent/arrival time. Upon receiving or sending a packet, an event is triggered which tells our system to gather and save the necessary data about the packet.

- **Process Data Collection Module** - The process data collection submodule is responsible for gathering data about the processes on the user system. Whenever a process is started or stopped, our system is notified. It then collects data about this process, including the time it was started or stopped, the process name, and the memory used by the process.

2.2.2 Server Communication Module

Once data is collected by the data collection module, it is sent to the server communication module where it is formatted properly and then sent to the server subsystem for analysis. The server communication module takes in a list of data points, formats it as a JSON object, and sends it to the server for analysis with an HTTP post request.

2.3 Server System Modules

The server side of the program is where most of the heavy lifting is done. This is where the data accumulates, where it is parsed and then interpreted to glean information about security risks on a client by client basis. From this interpreted data, the program determines the user's security score and provides that information to the user. This is also where, when an anomaly occurs, the program can send out notifications via email to users and supervisors within the company. The server is split into four key modules which are: the Runner and Receiver, Feature Extraction, Anomaly Detection, and Database Interaction. An analysis of their make up follows:

2.3.1 Receiver & Runner

This is the module which receives the three main types of data (file, network, and process) from all the clients currently connected to the system, and runs the feature extraction and anomaly detection process on that data. It also is responsible for sending user data to the database and toggling learning mode. This module primarily is responsible for delegating responsibilities to the other modules in response to incoming data.

2.3.2 Feature Extraction

The data collection system of the client side is sending an extremely large amount of data on many aspects of a user's behavior from one client computer, let alone multiple machines networked to this single instance of the program. Because of this, feature extraction becomes an important process in determining anything useful from all that data. The system builds a user profile on the database from data gathered on the client side during a given time, an engaged "learning mode." From this profile, the program learns what normal behavior for a given user will be. In the language of machine learning, feature extraction clears away the majority of the

data that is deemed to be extraneous. From each type of data (file, network, process), relevant information “features” are extracted and converted into a numeric format suitable to be analyzed by the anomaly detection. This format is an n-dimensional vector, where n is the number of features extracted for a given dataset. This process is complicated and involved, and is important so these useful vectors of data can be passed on to the next step.

2.3.3 Anomaly Detection

Anomaly Detection is, in many ways, the heart of the entire system. This is where the program makes decisions about useful data sent to it by feature extraction. This is also where a user’s learned profile is compared with the incoming data vectors; it is in this comparison where the determination of security risks is implemented. A user’s profile is their baseline, and while it may be impossible to have a perfect baseline for a user, the redundant systems of the program, such as the scoring and notifications, help mitigate gaps in the ability of the system to find security lapses.

Analysis of the data coming from the client side is in real time. Following the feature extraction, the anomaly detection uses the data sent to it combined with the user's learned profile on the database to determine when behavior is out of the ordinary. This is being done using the local outlier factor technique. The local outlier technique compares the local density of points from our feature extracted vectors and learned user data. Using these densities the program can find points that have lower densities than others. These are the outliers and the anomalies in the user behavior data. When these anomalies are detected, information about them is passed back to the runner & receiver which passes it along to the user interface web app.

2.3.4 Database Interaction

This module is responsible for handling all communication with the system’s database. The database stores user account information, necessary for a user to log in to the web application. It also stores learned user behavior data for comparison during anomaly detection, as well as a list of IP addresses required for lookup of location data about network data packets. Since the database is frequently accessed, this module is used by each part of the server-side system.

2.4 User Interface Web Application Modules

The other main part of the program that directly involves the user is the User Interface Web Application. This system engages the user with feedback about their activities through a performance score, likened to a credit score, which is available to see in the web app for the program. This program interface is accessible through a user's web browser so they can view their score and other information at their convenience.

The data presented to the users in the web application is at the other end of the pipeline as it were, as far as the scope of the entire program is concerned. The score and other information comes from data collection done on the user's computer, and is computed by the server and sent to the client side after analysis and calculations have been made. By being able to see their user score, it is possible to educate the user on what actions they may be taking that could be considered security risks, and tell them what steps they may take to prevent such risks. With this system, the goal is to teach users to observe diligent and secure use of their workplace computer system.

2.4.1 User Scoring

The user's security score, which is the numeric representation of their security risk determined by the system, is calculated in this module. The score is formulated from data output from the anomaly detection and is updated regularly as new data arrives into the system and anomalies are confirmed as security issues or dismissed. Similar to a credit score, this number encourages a user to engage in secure computer practices and mitigate risk on their own part. As an ever changing aspect of a user's presence on the system, this is passed on to the user notification module so it may be shown to the user or to their supervisors. False positive anomalies can be stricken from the user's score and not affect the end result if so decided by an administrator. This aspect of the system can potentially require administrative access to allow tweaking of the user scoring system to the organization's preference.

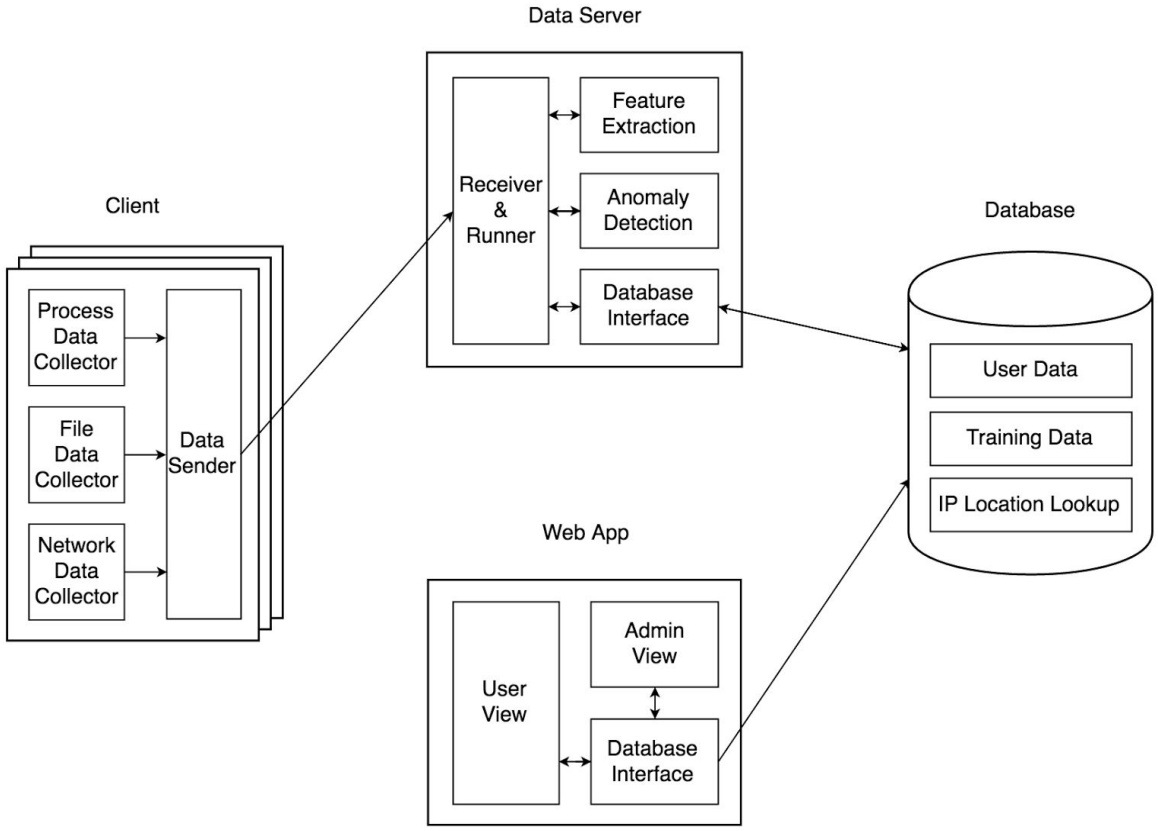
2.4.2 User Notification

The user notification module provides information about potential threats, abnormal behavior, and weeds out false positives through the use of notification by email. This module is important,

as it is one that interacts directly with the users. With the notification module, it will be possible to inform not only the client side user, but also others in that organization about abnormal behavior or security breaches. Another essential duty of the notification system is dealing with false positives that may arise during analysis of user data. If the user is flagged for abnormal behavior or a security breach, and the system has potentially misinterpreted data, the user has the chance through notifications to tell the system that this aberration is not a threat. This has the potential to be abused (ie, we don't want actual intrusion events to be able to confirm their own legitimacy), so the removal of anomalies considered false positives is handled by the system administrator.

Keeping the client aware of their current security status within the system is important, but perhaps just as important is alerting others to potential issues. Since this is a matter of discretion, and because many more scenarios for the use of this program will likely become apparent over time, the notification system may be controlled on the server side of the program. As an administrator of the program, one is be able to decide the level of notification by email a client would receive, or even the level of notification to the supervisors of the firm. Because many users have a wide range of skill sets on modern computers, a company may not find it acceptable to pester an advanced computer user with notifications. On the other hand, a company may find a user with less experience requires these notifications in order to thwart easily preventable incidents. The goal of the notifications system is to provide timely feedback about security, and to not be an annoyance or hindrance to the process.

3 Implementation Details



3.1 Client Implementation

Our client system is split into two main modules data collection and data sending, the data collection module can be further broken down into small modules for each type of data as discussed previously in this document.

3.1.1 Data Collector Implementations

The data collectors of our system are all implemented in a similar way, but each has its own unique features in order to make it fit its type of data.

Our system has a data collector class for each different type of data it would like to collect. This gives us three different data collector classes, the “FileDataCollector”, the “NetworkDataCollector”, and the “ProcessDataCollector”. Each of these classes implement a “DataCollector” interface which has a single method within it called “Initialize”. The Initialize

method has a single responsibility, which is to set up all of the event listeners that listen for changes on the system related to the given data type.

3.1.1.1 File Data Collector Implementation

The file data collector's initialize method sets up an event listener for each directory and subdirectory on the user's system. This event listener listens for changes to any files within the directory. When an event is received the system will analyze the event and obtain the type of change that was made to the file, the path to the file that was changed, and the time the file was changed.

3.1.1.2 Network Data Collector Implementation

The network data collector's initialize method behaves similarly to the file data collector's. First it sets up listeners for different network devices on the user's system. These listeners will listen for any packets sent or received by the network devices and upon receiving an event, obtain the size of the packet, the destination or origin ip address of the packet, and the time the packet was received or sent.

3.1.1.3 Process Data Collector Implementation

The process data collector's initialize method creates event listeners that listen for process create or delete events on the user's system. Upon receiving one of these events the listeners will obtain the name of the process, the amount of RAM used by the process and the time of the event occurrence.

3.1.2 Data Controller Implementation

The Data Controller acts as a middle-man between the data collectors and the data sender. It is responsible for storing the data and determining when to send the data. When any of the collectors is finished collecting data for one event, it sends that data to the Data Controller. The data controller then puts that data into the correct type of data container. These data containers are classes that we created that are only responsible for holding all of the data points we obtain.

The Data Controller also has its own event listeners, these listeners keep track of how much data has been gathered by the collectors and how much time has passed since the last time data was

sent. When either of these reaches a given threshold, an event is triggered and data is sent using the data sender.

3.1.3 Data Sender Implementation

Our data sender is responsible for sending all of the data that has been collected by our data collectors to the server so that it can undergo analysis. In order to do this it uses an HTTP post request that consists of three parameters; a user name, to determine where the data is coming from and what data the new data should be added to, a type, so that the server knows if the data is file, network, or process data, and last, the data itself. The data is sent to the server as a JSON string. This string is a serialized version of all of the data that was stored inside the data collectors.

3.2 Server Implementation

Our server is written exclusively in python and consists of four main modules. The basic flow of data throughout our server is as follows. First JSON data is received in the receiver which deserializes the data into a list of data points and sends it to the runner. The runner then takes the data and passes it on to feature extraction which looks at the data and converts it into a form that can be used by our anomaly detection module and then returns it to the runner. The runner takes that data and passes it to the anomaly detection module which compares it to the previously learned data and looks for anomalies.

3.2.1 Receiver and Runner Implementation

The Receiver and Runner Module is responsible for receiving data from client machines and the passing this data along to other modules on the server. The first thing that will happen on the server is data is received by the receiver in the form of an HTTP post request. The receiver creates a new thread and parses the necessary data from this request, which consists of the data type that was sent, the user who sent it, and the data itself which is in JSON form. Once it obtains the JSON string, it uses the Python JSON library to parse the json into a list of data points, and passes all of this data to the runner.

When the runner receives the data it checks what type of data it has received and sends it to proper feature extraction module. The feature extraction module will return a vector of features

back to the runner. The runner then checks if the user is in learning mode or not. This is very important because it will determine what the runner does with the vector that the feature extraction module has created. If the user is in learning mode, the runner will use the DIM (database interaction module) to insert the data into the proper table in the database, if the user is not in learning mode, then the runner will pass the data onto the anomaly detection module.

3.2.2 Feature Extraction Implementation

The feature extraction module is responsible for converting all of the data we received from the client into numeric form so it can be used by the anomaly detection. It puts the numeric data into an n-dimensional vector, where "n" is the number of numeric features that we can extract from the data sent by the client. The feature extraction module is split into three submodules. Each submodule corresponds to one of the data types our application is analyzing. The three submodules are required because we are receiving different pieces of data for each data type.

3.2.2.1 File Feature Extraction

The data we collected related to the file changes on the user's system consisted of, the name of the file that was changed, the time the file was changed, and the type of file change. Using this data we created an 87 dimensional vector which consisted of time of file change (in the form of second since the previous Sunday at midnight) and the number of times the characters accepted characters appear in the name of the file*.

* The accepted characters are SPACE, !, #, $, %, &, ', (,), +, COMMA, -, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, =, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [,], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, {, }, ~.

3.2.2.2 Network Feature Extraction

The data we collected related to the network usage on the user's system consisted of packet size, time of packet, and IP address. With this data we created a 4 dimensional vector, consisting of packet size, time of packet arrival or departure (in the form of seconds since the previous Sunday at midnight), latitude of IP, and longitude of IP. We acquired the latitude and longitude corresponding to the IP address using info acquired at http://lite.ip2location.com. This is mentioned below in the Database Implementation section.

3.2.2.3 Process Feature Extraction

The data that was collected related to the processes running on the user's system, consisted of time the process was started or ended and the amount of RAM the process used. Using this data we created a 2 dimensional vector of numeric data consisting of time the process was started or ended (in the form of seconds since the previous Sunday at midnight) and the peak RAM usage by the process.

3.2.3 Anomaly Detection Implementation

The anomaly detection module is implemented using the sklearn package from the Anaconda distribution of Python. From this package, we used the One Class SVM (Support Vector Machines) functionality to detect anomalous data. The class creates a data model based on a set of training data. We provide this training data from our feature-extracted learning mode data. The model is created to represent this data as a single "class" of data. It does this using the Gaussian Radial-Basis Function (RBF) to compare points. (This is in contrast to comparing points linearly using a Euclidean distance function.) The RBF is used for more amorphous data sets, as ours will be; by amorphous, we mean that we expect there to be several local maximums of activity for each of the various data types. For example, users probably will be more active immediately before lunch, and immediately before they leave. We can then pass in vectors of the same size to determine whether or not they fit the class modelled by the training data. If they do not, we count them as anomalous data and use this to update the user's security score.

3.2.4 Database Interface Implementation

The database interface is written using the MySQLdb library for Python. This module is a static class that handles the functionality of adding, retrieving, and deleting information from the database for use in anomaly detection.

The module has the following functionality:

- Insert new learning mode data point into the database
- Retrieve the set of learning mode data points for anomaly detection
- Switching on and off learning mode for a user
- Checking if a user is in learning mode or not
- Getting the latitude and longitude from an IP address value

- Getting the user's security score
- Setting the user's security score

Having this a separate module creates better separation of concerns and makes the system more maintainable and extensible should the database setup need to be changed or new functionality added.

3.3 Database Implementation

The database is implemented using a standard MySQL server. This MySQL server is hosted on the same Red Hat Linux virtual machine on which the anomaly detection server is running. The database includes the following tables with columns as specified.

3.3.1 LearnedFileData

The table for learned training data is an 89 column table that keeps track of the information of the sending user and the counts of each alphanumeric characters use in the filename of the change being tracked.

The table will look as follows with 86 columns of the generic type listed:

Column Name	**Data Type**
User	varchar(16)
Type	tinyint
Time	int
Num_[alphanumeric character]*	tinyint

*The 86 possible alphanumeric characters are as follows: SPACE, !, #, $, %, &, ', (,), +, COMMA, -, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, =, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [,], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, {, }, ~.

3.3.2 LearnedNetworkData

Column Name	Data Type
User	varchar(16)
Type	tinyint
Latitude	float
Longitude	float
Time	int
Size	int

3.3.3 LearnedProcessData

Column Name	Data Type
User	varchar(16)
ProcessName	varchar(100)
Type	tinyint
Time	int
RAM	int

3.3.4 Users

Column Name	Data Type
Username	varchar(16)
Password	varchar(255)

Name	varchar(100)
Score	float
PointsProcessed	int
Admin	tinyint
LearningMode	tinyint

3.3.5 ip2location_db5

Column Name	Data Type
ip_from	int
ip_to	int
country_code	char(2)
country_name	varchar(64)
region_name	varchar(128)
city_name	varchar(128)
latitude	double
longitude	double

This product includes IP2Location LITE data available from http://lite.ip2location.com.

The table is set up and filled as specified by the above website.

3.4 Web Application Implementation

The web application is implemented using an Apache Server. The pages are written with HTML, CSS, and PHP. A database support module is written to fetch the necessary information from the database to display on the pages.

There is a main login page to let the user log in to their profile. From there they are taken to the user view. The user tells the user their score and gives them some information on how the score is calculated.

4 Testing Process and Testing Results

4.1 Unit Testing

During development we split our project into independent modules and assigned each module to a developer. As our team developed these modules each member was responsible for testing their modules by giving it various inputs verifying that the correct outputs were returned. Using Python for the server-side made it easy to test out individual modules that each of us wrote, give it expected or unexpected input and observe the results. This was a major part of our testing as we were often not working in the same place and just uploading things through version control.

4.2 Integration Testing

When modules were completed, we would integrate them into the main project and then use operational scenario testing to verify that the system is behaving as expected. Using our ISU provided server space we combined our work on the server-side and put it on the server. At this stage we began sending testing data to this code that we created in order to verify the output of the server-side program.

On the client-side things were brought together on our computers and run to verify the collection of data and the correctness of the data being sent from the clients.

4.3 Final Testing

Once our project was nearly complete, we began final testing to ensure the entire project was working. To do this, we first procedurally generated large amounts of data considered to be “typical” to use as the training data set. Once that was complete, we generated more points, some generated in the same way as the training set, and others generated in a way that would make them anomalous. We then passed these to the anomaly detection module to verify that the correct

data points were found to be anomalies.

In addition, we also tested the system by running it on group members' own computers and observed the results. With data from up to 6 group members, this would be a moderate stress test for the system, but not the maximum number of intended users for a corporate computing environment.

5 Appendix I: Operation Manual

5.1 System Requirements

The client-side portion of the program is designed to be lightweight and run unnoticed in the background of a user's computer. Therefore it does not have particularly stringent system requirements; since that portion of the program primarily consists of storing data, sending data, and displaying alerts. The system requirements for the client-side are as follows:

- A 32-bit or 64-bit Windows operating system
- An internet connection with a bandwidth of 0.5MB/s

The server side portion of the program does all of the computation for the networked user computers and handles all of the data that comes in, and therefore requires quite a bit more processing power than the client-side. The server-side is also designed to be a Linux based system.

Step 1: Install on Server

a. The system uses the Anaconda version Python, so this will have to be installed on the server to be used.
b. Have the planned system administrator install the Abacus system on a server networked to a group of client computers to be monitored.This administrator will have exclusive access to the server-side system and will be responsible for making false positive changes.
c. A database must also be installed on the same server for the system to access. Ideally in a finished shippable product this would be automatically created when the server is installed, however this is not a feature at this time. The database should be set up as SQL tables, which are defined above in the Database Implementation section. This will allow the client-side and server-side code to communicate with it.
d. For the web application on the server, an Apache web server and PHP must be installed. Then the files for the web application should be loaded into the WWW folder for the Apache server. The database_support.php script must be edited to set the MySQL

username, password, and domain. All other scripts must be edited to set the proper URLs for redirects as necessary. Finally, the Apache Server must be started.

Step 2: Install on Client Computers

a. Have the system administrator set up the client-side system on the user's PCs networked to the main server, according to what level the organization has decided to track data.
b. Individual users will log in to the Abacus Web App to make accounts so they can monitor their information and security score.
c. The system administrator will then engage learning mode on the user's PC for admin determined amount of time. This will collect data that will be used to compare against for anomaly detection.
d. Anomaly detection will be running in the background and the user score will go down if an anomaly is detected.

Step 3: How to Use the Abacus Web App

a. Have the user login, using the credentials that we set up for them during the training portion.
b. View security score and how to improve upon it.

6 Appendix II: Alternative Versions of the Design

6.1 Expanded Design

Originally we had considered a more complex version of the implementation. This had included doing keyboard logging and detecting how the user uses the keyboard, in addition to mouse tracking and how the user uses it per program. In the end we scrapped this since it was too complex, and we couldn't really figure out a useful way to use this data in regards to detecting anomalies. As there is not a good way to keep track of all of this data, we decided it would be best to look at this feature at the end and implement it if we had time.

The other thing that we had originally considered was a continuous learning mode. This was to hopefully reduce false positives as if the system detected something that was a false positive it would be added to the user's data and therefore would not be detected as a false positive again in the future. This was scrapped due to time constraints, and now we just have the initial training data from the user that we detect anomalies against.

An important aspect we considered adding to the design, and in fact had it as part of it for a long time, was the ability for the system administrator to customize the level of monitoring on individual users. This was a reasonable course we thought because different organizations that potentially used our software would have different needs and users with different levels of computer experience. This, along with a continuous learning mode would provide and more accurate way to determine one user from another.

6.2 Altered Design

An aspect of the design that went through major changes was the client-side User Interface. Originally we planned on having the access to the system on a client's PC through an application that they installed and would be able to access on their system. The application would show the current user security score and other data about detected anomalies. Because of the nature of the data and the fact that this UI would need access to information from the server and the database,

we decided to change it so that this would be a web based application that the users could log in to on their web browser of choice. This web site is also hosted on the same server that the server-side program resides and the database, so there is easy access to any data we need to display for a user accessing the site.

It should also be noted that the general organizational plan of the client-side and server-side system went through revisions. We did our best to plan out our system's modules and anticipate how they would communicate with each other but when it came time to code it, the realities of data flow and access sometimes necessitated changes. One of these major changes was also the fact that we had planned to code the server-side system in C. Our line of thought being C would be a natural fit for a Linux system. We went away from that and wound up using Python, which mostly no one in our group knew, but which we think worked out much nicer that using C would have. We stuck with C# for the client-side which was a natural fit for the Windows environment.

7 Appendix III: Other Considerations

One concern we had while writing this project was whether or not the software we were writing was ethical. Since we were autonomously monitoring user behavior, and using it to predict their future behavior and try to influence them in a certain way (in this case, to influence them to have secure usage habits), our project seemed to have a lot of the same characteristics one would expect to see in malware or spyware.

Though those were legitimate concerns, the fact of the matter is that we were able to take certain liberties due to the fact that this was not a project that was not going to be used commercially. Even so, we modified our software to allow the administrator to limit the amount of data that would be collected on a per-user basis, so that users of the software could decide what level of monitoring they would like to keep track of. For example, when we were considering keeping track of the user's mouse movements, we realized that could be construed as a particularly intrusive level of monitoring; therefore, we allow the administrator to disable that form of monitoring. By taking these measures, we feel we allowed the customer to determine the exact level of monitoring they are comfortable using.

Although there may be a couple aspects of our project that could be considered unsuccessful or under-developed, we believe the project was a major learning experience. We learned a lot about group dynamics, how to handle project deadlines and setbacks over the course of the long term development of a system. We also treated this as a research project, a way to expand our knowledge of systems we were already familiar with and many that we had never encountered before. Along the way we occasionally found out what to do and what not to do. There were ups and downs, successes and failures, but at the end of the day our group had to learn to work together and function as a team in order to meet the expectations of our client and our advisers.