Stolen Object Tracker

Cycle 1 Report

By

Charles Baker

Denney Burkholder

Nathan Plotts

Leo Reyes

Steven Whaley

Submitted in partial fulfillment of the requirements for  
COMP 4710 Senior Design  
to the Department of Computer Science and Software Engineering,  
Samuel Ginn College of Engineering, Auburn University

Auburn, Alabama

March 4, 2014

Table of Contents

[2 Executive Summary (System Metaphor) 6](#_Toc381663523)

[3 Project Introduction 7](#_Toc381663524)

[3.1 Previous Development 7](#_Toc381663525)

[3.2 Intent This Cycle 8](#_Toc381663526)

[3.3 Future Work 8](#_Toc381663527)

[4 Requirements & User Stories 10](#_Toc381663528)

[4.1 Customer Requirements 10](#_Toc381663529)

[4.2 User Stories 10](#_Toc381663530)

[4.2.1 User Account Creation 10](#_Toc381663531)

[4.2.2 User Account Login through Web 10](#_Toc381663532)

[4.2.3 User New Device Registration 11](#_Toc381663533)

[4.2.4 Administrator Manages User Accounts 11](#_Toc381663534)

[4.2.5 User Views Device List 12](#_Toc381663535)

[4.2.6 User Views Device Data 12](#_Toc381663536)

[4.2.7 User Views Device Location on Map 12](#_Toc381663537)

[4.2.8 User Reports Stolen Device (Activates Device Tracking) 13](#_Toc381663538)

[4.2.9 User Installs Windows Service 13](#_Toc381663539)

[4.2.10 Laptop Automatically Connects to Open Wi-Fi 13](#_Toc381663540)

[4.2.11 Laptop Logs Keystrokes and Sends Data to Server 14](#_Toc381663541)

[4.2.12 Laptop Checks in with Server 14](#_Toc381663542)

[4.2.13 Geogram Automatically Detects when Stolen 14](#_Toc381663543)

[4.2.14 Geogram Sends Location Updates to Server 15](#_Toc381663544)

[5 Design Documentation 16](#_Toc381663545)

[5.1 Server components 16](#_Toc381663546)

[5.1.1 Central Server 16](#_Toc381663547)

[5.1.2 RequestProtocol 16](#_Toc381663548)

[5.1.3 Web Server 16](#_Toc381663549)

[5.1.4 Web Server Design Diagram 17](#_Toc381663550)

[5.1.5 Device Server 17](#_Toc381663551)

[5.2 SMS Gateway 19](#_Toc381663552)

[5.2.1 Overview 19](#_Toc381663553)

[5.2.2 Structure 19](#_Toc381663554)

[5.2.3 Interfaces 19](#_Toc381663555)

[5.2.4 Activity Diagram 21](#_Toc381663556)

[5.2.5 UI Storyboard 22](#_Toc381663557)

[5.3 Website UI and Storyboard 24](#_Toc381663558)

[5.3.1 Web UI 24](#_Toc381663559)

[5.3.2 Web UI Storyboard 26](#_Toc381663560)

[5.4 Key-logger 27](#_Toc381663561)

[5.4.1 Overview 27](#_Toc381663562)

[5.4.2 Requirements analysis 27](#_Toc381663563)

[5.4.3 Structure 27](#_Toc381663564)

[5.4.4 Interface 27](#_Toc381663565)

[5.5 Windows Service 28](#_Toc381663566)

[5.5.1 Overview 28](#_Toc381663567)

[5.5.2 Interface 28](#_Toc381663568)

[5.5.3 Activity Diagram 30](#_Toc381663569)

[5.5.4 Communication Diagram 31](#_Toc381663570)

[6 Management Plan 32](#_Toc381663571)

[6.1 Task Assignments 32](#_Toc381663572)

[6.2 Schedule 32](#_Toc381663573)

[6.3 Planned Code/Feature Freeze 33](#_Toc381663574)

[7 Risk Mitigation 34](#_Toc381663575)

[8 Test Plan and Test Procedure 35](#_Toc381663576)

[8.1 Test Plan 35](#_Toc381663577)

[8.2 Test Procedure 35](#_Toc381663578)

[8.2.1 SMS Gateway Tests 35](#_Toc381663579)

[8.2.2 SMS Gateway Review 37](#_Toc381663580)

[8.2.3 Windows Service Tests 38](#_Toc381663581)

[8.2.4 Windows Service Review 39](#_Toc381663582)

[8.2.5 Database Unit Testing 40](#_Toc381663583)

[8.2.6 Database Review 40](#_Toc381663584)

[8.2.7 Website Blackbox Testing 41](#_Toc381663585)

[8.2.8 Website Review 48](#_Toc381663586)

[8.2.9 Key-logger Blackbox Testing 49](#_Toc381663587)

[8.2.10 Key-logger Review 49](#_Toc381663588)

[9 Lessons Learned 50](#_Toc381663589)

[10 References 51](#_Toc381663590)

[11 Appendix 52](#_Toc381663591)

# Executive Summary (System Metaphor)

By: Leo Reyes

When an object is stolen, it often never makes it back to its owner. Although there are other tracking devices currently on the market, our product provides more data about the thief than a regular tracker can, and also provides a user-friendly online interface for viewing device related updates and managing user accounts.

There are two different tracking device options. The first is a laptop application that runs in the background, hidden from the user. When the laptop is stolen, the application can gather information from its environment and use this information to locate and retrieve the laptop. This information includes their IP addresses as well as a record of their keystrokes.

Another tracking option available is a GPS tracking device. This device is small enough that it can be placed inside most commonly stolen objects and will activate when notified by the owner.

A user-friendly online system, with a simple account registration and login, is available to device owners. When a device is stolen, the owner can easily activate the tracker by logging into the online system. The owner of the device can view location updates on a map through the online system, along with data related to specific devices including logged keystrokes and IP addresses. Support is also provided for registering and managing multiple devices.

# Project Introduction

By: Denney Burkholder

Over the past four weeks, our team has been working on creating a recovery program to track objects that are stolen without alerting the thief that he or she is being monitored.

We are designing for laptop computers and an object location device.  The goal for our project has been to create a product that will allow our customers to view their locations through our website, along with other information that may prove useful to recovering the device.  The need for a product such as this is undeniable when looking at the recovery rates for expensive devices (such as laptops, mobile phones, and tablets) once they have been lost/stolen.  Anyone who owns a desirable device can fall victim to thievery.  As such, they are our intended clientele.

By logging on to our user-friendly web application, the customer will be able to register their devices and access different information depending on the type of device.  For laptop computers, we have created a key logger that will be remotely activated by our server once the device has been reported missing on our website.  It runs in the background of the laptop, so the thief will not be aware that information is being gathered.  This file will be sent to our server whenever the laptop can get an Internet connection through available Wi-Fi networks.  The files can be viewed and downloaded from our web application. Our object location device is a self-aware tracking device.  It has GPS capabilities and an accelerometer allowing it to transfer its location along with the speed with which it is traversing to our server for viewing on our web application.  It communicates through cell towers, so a Wi-Fi connection is not necessary.

 Implementing our project involves many different components.  We have designed and coded a central server, database, applications for gathering data on laptops and web pages for communicating between the stolen devices and customers, and storing relevant information for recovery.

## Previous Development

By: Charles Baker

Previous development was performed in the architectural spike phase. Development was focused on the server and Windows service. Goals for the architectural spike were to get the most important features to a point where it was evident we could continue development without any major problems that may cause us to miss our final deadline.

Goals for the server were to get most planned functionality working in a basic state. The server was designed in 2 primary parts, the application server and web server. Goals for the web server included: embedding Google Maps into a page, setting markers on the map, and connecting to the application server. Goals for the application server included: communicating with the Windows service, communicating with the database, communicating with the web server, and moving data between different connections on request. The goal to connect the web and application server was not met because of difficulties with GlassFish. Furthermore, we were not able to work on sending requests due to time constraints and because most involve the web server. Otherwise, we made satisfactory progress on the remaining goals. We created the html pages and were able to embed Google maps into it. We were also able to add markers to the map by sending coordinates through a web socket. The application server was able to communicate to the Windows service by a TCP connection. We developed a messaging protocol that both server and client (service) adhere to when communicating. The server was also successfully communicating to the database by the end of the architectural spike. We were able to read and write to a database containing all values we predicted we would need in the future.

On the Windows service, by the end of the architectural spike we wanted to have it start automatically, communicate with the server automatically, identify itself, and log keystrokes. By default, when the service is installed it launches on startup. The service was able to identify itself by using a MAC address of a network adapter. This method mostly worked, but a virtual adapter appearing could cause problems in selecting the same MAC address each time. Connecting to the server was achieved by creating a thread to handle the connection. If connection was ever lost, it would simply try to reconnect. While connected, the service could respond to different commands sent by the server. Commands implemented where: start key logger, stop key logger, return key logs, and send an IP trace. Key logging was achieved, but was in a very basic state at the end of the architectural spike. It was able to record the key being pressed, but did not consider keys that were being held down. This means that it did not record the case of letters or other effects of key combinations being pressed.

## Intent This Cycle

By: Charles Baker

We decided it would be beneficial to port the existing server to the Google Go language. This allowed us to have a single server, instead of both an application server and web server. This is possible because Go allows application code that contains HTTP handlers. It also provided better support for multithreaded applications, which is important for our server that must manage connections to many devices and web connections. We believed that in the long run, the change to Go would speed up the development of the server by requiring a fewer number of components that must interact.

Another one of our primary goals this cycle was to incorporate the Geogram into the system. The first step was to send messages between the server and Geogram. We decided the communication would be done through an Android phone. An Android application would be responsible for connecting to the server via TCP and interfacing between TCP and SMS communication. The server must be able to send the appropriate commands and interpret received messages.

Work on the Windows service was also planned for this cycle. We planned to have the service check in to the server in set intervals. If the server notified the service that the laptop was stolen, it would constantly try to reconnect if the connection is ever lost. We also planned to improve the key logger by capturing when a key is modified by shift or capslock. We also planned to capture key combinations when multiple modifier keys are held down.

## Future Work

By: Charles Baker

A large portion of the remaining work is derived from finishing the website that will allow user to interact with the system. The registering and login system requires more work so that users will be restricted from certain pages if they are not logged in. We must also associate devices with the users that register them and be able to display relevant information on the website. The users must also be able to send commands to the server from the website, such as manually reporting a device stolen.

On the server we need to improve how devices are handled. One thing this includes is setting how automated commands will be sent to devices. Information must be gathered from registered devices in regular intervals to detect the possibility of being stolen. For example, Geogram devices must be sent commands in a way that maximizes battery life but provides up-to-date information once devices are stolen. Devices must be read from the database to memory in an efficient way and removed from memory when no longer needed. One occasion where this is important is when a user logs in and wants to view his/her devices.

# Requirements & User Stories

By: Charles Baker

## Customer Requirements

Our customer described an open ended project that resulted in being able to track stolen objects. He gave us a few requirements that we must adhere to. First the device must be able to locate itself. Secondly, the device must identify itself. Thirdly, the object must be able to communicate this information to the user. He also mentioned, when applicable, we should consider the battery life of any solutions we implement.

We were also given examples of possible implementations. One example was a device that could be hidden in large shipments and tracked if anything were to happen. Another was a device that is disguised as an expensive object that a thief would likely grab during a robbery. Additionally, the device could be an existing electronic device, such as a laptop that sends its IP address as a means of location.

We were to present our ideas to the customer and he would decide which solutions he preferred we implement. One solution he wanted us to implement was an application for laptops that could send its IP address for the purpose of tracking. Additionally, he wanted us to integrate the existing Geogram device into our system and use its GPS location to track it.

## User Stories

By: Charles Baker

### User Account Creation

Summary: The user enters account credentials into the website and a new account is registered and stored in the database.

Description: The user enters his/her first name, last name, email address, phone number, password, and confirm password. The user presses the register button. The server ensures all fields are valid, and if so creates a new user in the database with the entered information. If any fields are invalid, the user is notified which field(s) caused registration to fail.

Hours: Total Planned: 4  
Planned this cycle: 4  
Total Actual: 8  
Actual this cycle: 8

Coder: Denney, Leo, Steven

Tester: Leo

Reviewer: Leo, Charles

Status: Adversarial

### User Account Login through Web

Summary: A user with an existing account logs in.

Description: The user accesses the website and enters his/her email address and password. The server verifies the credentials and logs the user in if they are correct. The logged in user is redirected to his/her main page. If the login fails the user is notified that login failed.

Hours: Total Planned: 2  
Planned this cycle: 2  
Total Actual: 26.5  
Actual this cycle: 26.5

Coder: Denney, Leo, Steven

Tester: Leo

Reviewer: Leo, Charles

Status: Complete

### User New Device Registration

Summary: The user registers a new device on the website.

Description: The user goes to his/her device page and selects to register a new device. The user adds relevant device identifiers. The server checks that the information is valid, and if so associates a new device with the user in the database.

.Hours: Total Planned: 6  
Planned this cycle: 6  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Collaboration

### Administrator Manages User Accounts

Summary: The administrator views user accounts and can modify them.

Description: The administrator views a full list of user accounts. The admin can select an account to view its information. The administrator can modify account information or terminate the account.

Hours: Total Planned: 8  
Planned this cycle: 0  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Unstarted

### User Views Device List

Summary: The user views his/her devices.

Description: The user goes to a page displaying all devices registered by him/her. The devices are displayed in a list.

Hours: Total Planned: 8  
Planned this cycle: 8  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Collaboration

### User Views Device Data

Summary: The user views information about a registered device.

Description: The user selects a device on his/her device page. Information and commands relevant to the device appear. This can include (depending on device) name, ID, device type, option to load other data (key logs, etc.), and the view the last location (IP or map depending on device).

Hours: Total Planned: 8  
Planned this cycle: 8  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Unstarted

### User Views Device Location on Map

Summary: The user selects to view information of a GPS device and a map is displayed on the page with location data for the device.

Description: The user selects to view information for a GPS device. Google Maps is embedded in the page. The maps contain a marker for the most recently recorded location of the device. If the user selects to view previous locations of the device, the location data will appear as markers on the map colored differently than the current location.

Hours: Total Planned: 10  
Planned this cycle: 10  
Total Actual: 15  
Actual this cycle: 15

Coder: Leo

Tester: Group

Reviewer: Group

Status: Complete

### User Reports Stolen Device (Activates Device Tracking)

Summary: On the device list, the user selects a device and reports it stolen.

Description: The user is viewing a certain device. The user chooses to mark the device as stolen. The server attempts to keep gathering up-to-date information about the device.

Hours: Total Planned: 15  
Planned this cycle: 0  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Unstarted

### User Installs Windows Service

Summary: User installs the Windows Service to their laptop.

Description: The user runs the installer for the Windows service. The installer places the executables and resource for the service and keylogger so that they will run on startup.

Hours: Total Planned: 15  
Planned this cycle: 0  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Unstarted

### Laptop Automatically Connects to Open Wi-Fi

Summary: Laptop tries to connect to open or known Wi-Fi if stolen.

Description: When the laptop is marked as stolen it will try to maintain a connection to the server. If the laptop is unable to connect because it has no Internet connection, it will attempt to connect to open and known Wi-Fi so it can communicate with the server.

Hours: Total Planned: 15  
Planned this cycle: 2  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Unstarted

### Laptop Logs Keystrokes and Sends Data to Server

Summary: When stolen the laptop logs keys and sends them to the server.

Description: If the laptop is flagged as stolen, it turns on the key logger. The key logger stores keystrokes in a logs files. Periodically the service sends the contents of the log files to the server for viewing.

Hours: Total Planned: 20  
Planned this cycle: 20  
Total Actual: 20.5  
Actual this cycle: 20.5

Coder: Nathan, Charles

Tester: Nathan

Reviewer: Charles, Nathan

Status: Complete

### Laptop Checks in with Server

Summary: The laptop periodically connects to the server when not flagged as stolen.

Description: The laptop will connect to the server periodically when it is not flagged as stolen on the laptop. If the device is flagged as not stolen on the server, the server will tell the laptop it is not stolen and the laptop will wait again before checking in. If the server has the laptop marked as stolen, it will tell the laptop that it is stolen and it will try to maintain communications.

Hours: Total Planned: 15  
Planned this cycle: 5  
Total Actual: 11  
Actual this cycle: 11

Coder: Charles, Nathan

Tester: Charles

Reviewer: Charles, Nathan, Leo

Status: Complete

### Geogram Automatically Detects when Stolen

Summary: The Geogram detects if it is stolen based on movement and location data.

Description: The accelerometer options are set so that the Geogram sleeps until it detects movement. Once it wakes up, location data will be gathered to determine if it is stolen. If the server determines there is potential for it to be stolen the user will be alerted.

Hours: Total Planned: 15  
Planned this cycle: 0  
Total Actual: 0  
Actual this cycle: 0

Coder: N/A

Tester: N/A

Reviewer: N/A

Status: Collaboration

### Geogram Sends Location Updates to Server

Summary: The server sends a request for the Geogram’s location due to the user marking it as stolen or the Geogram detects movement.

Description: The user marks the Geogram as stolen or it detects movement and sends the server a message. The server periodically requests the location of the Geogram.

Hours: Total Planned: 25  
Planned this cycle: 10  
Total Actual: 31.5  
Actual this cycle: 31.5

Coder: Charles, Nathan

Tester: Charles

Reviewer: Charles, Leo

Status: Complete

# Design Documentation

3-tier architecture – Client (Web), server, and database

## Server components

By: Leo Reyes

### Central Server

The central server is the entry point of the server. It initializes and starts the modules of the server, and initializes the communication channels used by the modules.

The central server connects all of the server modules together. All components send and receive requests through the central server. When a request is received, the central server redirects the request to the proper module.

### RequestProtocol

The modules of the server communicate with a requesting protocol. Using a requesting protocol allows the server to be organized. It also allows for adding and removing server modules with less effort, because it allows each module to have a high cohesion.

The protocol contains op codes for modules to use when creating requests.

A requests contains an ID, the destination of the request, the source of the request, an op code, a payload, and a response channel that the response to the request will be sent over.

### Web Server

Http handler – handles http requests received and sends back an http response to the request. The response will contain all of the files needed for the client view. A typical response would contain files such as images, style sheets, scripts, and the html file.

Web socket handler – handles web socket requests made to the server. When a web socket request is received, the handler creates a new connection to the client and registers the connection in the Web hub.

WebClientConnection – Middleman between web socket and the web hub. Reads in messages from the web socket and passes it on to the hub. Also, receives messages from the hub and passes it to the web socket.

Web hub – Maintains a list of connected clients as WebClientConnections; creates a channel for communication to and from those clients

Client Web Sessions – when a web client makes an http request to the server, a cookie-based session is created. A copy of the session is stored in the server. When a client logs in, the contents of the cookie are changed and represent a session for the client. The sessions are used for access control and to present the clients with relevant data.

RequestProtocol handler – handles incoming and outgoing requests made from the other server components.

### Web Server Design Diagram



### Device Server

By: Nathan Plotts

This portion of the server in its entirety is the part that communicates with all devices that are registered to be tracked. Also, all object structures that are used for defining a laptop or GPS device and its corresponding connection within the code are defined within this part of the overall server. Within this portion of the server we have created several files to help organize the individual device connections and structures.

#### deviceHub

This is the large driver file for the rest of the device package. It drives the smaller, more specialized GPS and laptop hubs by starting their connection handlers and giving them channels to send requests to the server through, though most of the work is done by the smaller hubs, laptopHub and gpsHub. One of the main jobs of the deviceHub is to monitor the request channel coming from the server. As requests are received from the server they must be forwarded to the correct hub to be processed.

#### laptopHub

The laptopHub is one of two smaller driver programs that manages the handling of laptop connections. When the laptopHub is started 2 goRoutines are launched. The first is a thread for hashing open connections using the corresponding device. The second thread begins by creating the socket through which connections will be accepted. After the socket is created, the thread blocks while waiting for incoming connections. Once a connection is opened a new goRoutine is started to read from the connection. This is done because reading from a connection is a form of blocking IO which pauses the thread until there is a message for it to read. The first message that is received by this goRoutine is the deviceID, which for a laptop is a MAC address. Once the deviceID is received it is passed into the goRoutine that is used for hashing connections. The connection is then hashed using the deviceID that has now been read. After hashing, the laptopHub responds to the laptop device by sending it an OP code that lets it know if it has been stolen. Before the laptopHub can respond with this OP code though, it must first send an internal server request to the database to find out whether or not the user has marked the device as stolen. The database responds either true (stolen) or false (not stolen) and then this response is converted to an OP code which is sent to the connected laptop device. If the OP code is not stolen, then the server signals the laptop to close the connection and check back in again in a predetermined amount of time. If the OP code returns as stolen then the laptop starts tracking and the connection remains open for the laptop to send data over.

#### gpsHub

The gpsHub is the other of the two smaller driver programs that manages the handling of the GPS device connections. When the gpsHub is started a socket is created that will be used to communicate with the SMS Gateway. Once the socket is created a connection is opened with the gpsHub by the SMS Gateway. As SMS messages are received by the SMS Gateway they are passed over the open TCP connection. Each message is sent to the gpsHub with a phone number that the message originated from and a set of GPS coordinates from its last recorded location. As each message comes in, the gpsHub parses the GPS coordinates and sends a request to the webClient to update the web view if the owner of the GPS device is currently tracking the device online. At the same time, a request is also sent to the database to store the GPS location. Commands to start tracking can also be sent to the GPS devices through an online webpage that will send a request to the gpsHub to send a message to the GPS device.

#### device

This is a small file that defines the basic structure for any device that connects to the server. This struct is inherited by other child structs like laptopDevice and gpsDevice.

#### laptopDevice

The laptopDevice file is used to define both the structure of a laptopDevice object and also to define the internal server request methods. The laptopDevice struct, as stated above, is a child of the device struct and defines extra information like a device current open TCP connection. The internal server requests defined for laptopDevice are methods that can be called on a laptopDevice object that will return the expected values.

#### gpsDevice

The gpsDevice file is very similar to the laptopDevice file, but used instead to define the gpsDevice struct and the methods for creating internal server requests for GPS devices. Currently though, the gpsDevice struct does not hold any extra information that is already defined in the device struct. Later, we will implement more functionality with the gpsDevice struct, but as it stands it is most useful to more accurately describe the objects we define with it in the code.

## SMS Gateway

By: Charles Baker

### Overview

The SMS Gateway app enables communication between the server and Geogram device. It runs on an Android smart phone and connects to the server via TCP. The server can send messages via TCP and the SMSGateway app will send those messages as SMS messages to the target Geogram device. The app will also read incoming SMS messages and forward them to the server via TCP.

### Structure

The SMS Gateway app is comprised of three classes. The first is SMSActivity. SMSActivity defines what happens when a user interacts with the user interface. The user interface is defined by the activity\_sms.xml file, which places the various buttons, text boxes, etc. on the screen. SMSActivity has a few functions that start and stop the thread that the TCP connection exists on. It also updates the UI when connection statuses change.

The TCPAsyncTask class inherits from the AsyncTask class, which is a class used for multithreading in Android. This class creates a TCP connection to the address entered by the user. While the connection is alive, it checks for new TCP and SMS messages and processes them. It also sends heartbeats to the server at set intervals, and expects a response, to ensure that the connection is still alive. If the connection is lost, it will continually try to reconnect until the user manually tells it to stop (by pressing the disconnect button). When the TCP connection status changes, it also calls methods from SMSActivity that are executed on the GUI thread that update the GUI to reflect the current connection status.

The SMSReceiver class extends the BroadcastReceiver class and listens for incoming SMS messages. When an SMS message is received, it places a copy of it in a linked list that can be accessed by static methods.

### Interfaces

#### SMSActivity

**Public:**

void connectFunction(View view): This method is called when the connect/disconnect button is pressed. It is responsible for starting and stopping the thread that handles TCP connections.

**Protected:**

void connected(): This method is called when there is a TCP connection to update the GUI so it reflects that there is a connection.

void notConnected(): This method is called when there is no TCP connection, but it is trying to connect, so the GUI will be updated to reflect this.

void disconnected(): This method is called after disconnect(), once the TCP connection has actually disconnected and thread ended, so the text fields are re-enabled for the user to type in.

#### TCPAsyncTask

**Public:**

TCPAsyncTask(SMSActivity parentAct): This method creates a new TCPAsyncTask with a reference to the parent activity so it can update the interface.

void endTask(): This method sets the task to disconnect and end the thread.

execute(String… arg): This method runs the task as a new thread. The arguments should be the IP and port of the server in that order as Strings. This method calls the doInBackground method.

**Protected:**

void onProgressUpdate(Integer… progress):This method is called when publishProgress(Integer… progress) is called. It executes on the GUI thread and updates it to reflect the current connection status. The argument should be one of the constants NO\_CONNECTION or CONNECTION.

void onPostExecute(Boolean bool): This method is called automatically when the task ends. It makes the text fields on the GUI editable again.

Boolean doInBackground(String… arg): This method should not be called directly. It maintains a TCP connection with the server. It processes inbound TCP and SMS messages when the TCP connection is alive and uses a heartbeat to test if connection is still alive.

#### SMSReceiver

**Public:**

void onReceive(Context arg0, Intent arg1): This method is called automatically when an SMS message is received. It adds a copy of the SMS message to a static list.

boolean hasMsg(): This method returns whether there is an SMS message waiting in the list.

SmsMessage getNextMsg(): This method removes and returns the next SMS message from the list.

#### TCP messaging protocol

Messages are sent in the following format: [<phone number>]<message>|

Example: [1234567890]Hello World|

Characters ‘[‘, ‘]’, and ‘|’ are reserved and should not be used in the message (they are not used in any Geogram One commands or responses, so this does not cause problems).

### Activity Diagram

This activity diagram shows the actions the SMS Gateway app takes from start to stop. The top area on the diagram has the user enter server information and then the app connects to the server. Once the app is connected to the server it enters a large loop where it processes incoming SMS and TCP messages. There are two paths out of this loop. Either the user has manually disconnected by pressing the disconnect button or the connection to the server has been lost.



### UI Storyboard

The UI storyboard shows the UI changes under typical usage of the app. This involves the user entering server information, connecting to the server, and then disconnecting from the server. The main changes to notice are the button’s text changing based on connection status, the text that displays the current status of the connection to the server, and the text fields becoming disabled for editing purposes when a connection is established or attempting to be established.

User enters server information



User presses disconnect

Application connects to server

User presses connect

## Website UI and Storyboard

By: Denney Burkholder

### Web UI

User Webpages:



Admin Webpages:



### Web UI Storyboard

User Webpages Storyboard:



## Key-logger

By: Nathan Plotts

### Overview

When the key-logger is started, it will simply sit idle until it receives a start command from the WindowsServiceTracker. Once the key-logger receives the start command it sets a hook into the beginning of the Windows keyboard input hook chain. Now that the key-logger is hooked into the beginning of the keyboard input hook chain, any keyboard inputs will intercepted by our key-logger before it is able to pass to any other application's in the keyboard input hook chain. When the key-logger receives the keyboard input it will decode the key code and print it to a text file. The text file will then be accessible to the WindowsServiceTracker, which will be able to transfer the data to the server. The WindowsServiceTracker can disable key-logging by a stop command, and the key-logger will remove its hook from the chain.

### Requirements analysis

We determined for the key-logger to be useful and meet requirements it would need to receive commands from the Windows service which communicates to the server. To do this it should start when the user session begins so it is always ready to log. It must also be hidden from the user, so it does not alert them.

### Structure

The key-logger is a single file program. It runs as an application in the system tray, but does not place an icon there that would alert a user of its presence. It is intended to run on startup and be called to be turned on and off as needed. The callback function (HookCallback) is the most notable function. When the hook is placed in the keyboard input hook chain, it is called with every keystroke. It is responsible for parsing the key code passed in as an argument and storing it in the log file.

### Interface

#### SystemTrayKeylogger

StartKeyLogger(): Sets a hook in the keyboard input hook chain and begins logging keystrokes

StopKeyLogger(): Removes its hook from the keyboard input hook chain and stops recording keystrokes

CheckIfRunning(): Returns true. Used to check that the key logger application is running.

GetKeylogFilePath(): Returns the path of the key log file as a string.

## Windows Service

### Overview

On startup, the Windows service attempts to connect to the key logger by a named pipe. This allows the service to make function calls on the key logger that are in the interface defining the pipe. If the laptop does not have a LAN connection, the service attempts to connect to an open or known Wi-Fi connection. The service then creates a thread that tries to connect to the server via TCP/IP connection. Once it connects to the server, it sends its identifier and waits for commands. It performs commands until the service shuts down or it loses its connection to the server. When it loses its connection to server it checks and begins trying to connect again in the same fashion. Alternatively, the server can tell the service that it is not stolen and it will wait for a duration before checking in again.

### Interface

#### Tracker

Tracker(): Constructor for the service

bool StartKeylogger():If connected to the key-logger, turns key logging on.

bool StopKeyLogger():If connected to the key-logger, turns the key logging off.

string GetKeylogFilePath(): If connected to the key-logger, returns the string representation of the path of the key log file.

bool CheckIfRunning(): Returns true if the key-logger is running and the service is connected to it via the named pipe

**Protected:**

void OnStart(): Automatically called when the service is started. Initializes variables and spawns a new thread to handle the TCP connection.

void OnStop(): Automatically called when the service is stopping. Closes connections and joins the thread that handles the TCP connection.

#### IP

**Public:**

getTraceRoute(string hostNameOrAddress): Performs a trace route to the address given as an argument and returns the results as a list.

#### Messaging Protocol

The Windows service communicates with the server by TCP/IP. The format of messages is <op code><message><newline>. The op code is a single byte that request a certain operation to be performed. The message may be blank if an op code is sufficient.

Op codes:

* 0: start key logging
* 1: stop key logging
* 2: IP trace route
* 3: request key-log files
* 4: report not stolen
* 5: report stolen

### Activity Diagram

This activity diagram shows the general flow of the Windows service. It first tries to connect to the key-logger. Then it will connect to the server upon startup, unless it does not have an internet connection. It will search for open Wi-Fi if it has no internet connection, but this is currently not implemented. Once connected it will listen for commands from the server. It will continue to process messages until one of three things occurs. First, it could get unintentionally disconnected from the server, and it will continually try to reconnect until it is successful. It will gradually wait longer periods as connections continue to fail in a series. The service could also receive the command that tells it that it is not stolen, and it will disconnect and sleep for a time before connecting again. The last way out of the loop is on system shutdown, where the service will be terminated.



### Communication Diagram

This communication shows a simplified view of the method and data flows between a Windows tracker service, the key-logger that works with the service, and the server that the program relays data and commands to. It first starts with the Service opening a connection with the server. As soon as the connection is open it then sends its MAC Address to the server to identify itself. Once it has identified itself it then waits to receive an OP code from the server. The server will send different OP codes depending on whether or not the device is marked as stolen. The possible actions taken by the Service can branch from there, as shown by the multiple paths in step 4.

# Management Plan

By: Steven Whaley and Denney Burkholder

Each group member has been significantly involved in each major area of development at least in terms of input on design issues and more generally participation in frequent group discussions. The decision to refactor the server from java to GoLang was based on brainstorming sessions involving research and active discussion between all six members. Tasks related to documentation (such as the creation of UML diagrams, code comments, etc.) have been shared between all members as well.

## Task Assignments

Beyond design decisions, a few tasks were explicitly defined to specific people. To save time while refactoring the server, the group members roughly stayed within the area they worked on for the architectural spike.

Server Development:

Leo Reyes: Combining the different aspects of the server, http functionality and creating client sessions.

Steven Whaley: The database communication portion of the server.

Denney Burkholder: The http communication portion of the server.

Nathan Plotts: The laptop device and GPS device portion of the server.

Charlie Baker: The android app to allow communication between the server and the Geogram.

## Schedule

At the end of the architectural spike, we were left with the task of refactoring our server to a different language. For the first week, we discussed the different options we had language wise. We researched and met to discuss ideas frequently during this period. After a few discussions, it was decided that GoLang would be the best option. GoLang automated many of the features we were planning to implement in java. After coming to this conclusion, we sat down and created the design for our server. Because GoLang offers many resources for handling multithreading, most of the design discussion was trying to figure out how to best implement these features without underdoing or overdoing it, for the needs of our system. After much discussion, we had our server redesigned. We started to divide up tasks at this point. Immediately following the architectural spike, the Geogram also arrived. We put off incorporating the Geogram initially as we figured out the server. Afterwards, we began trying to send and receive texts to the Geogram. This was done through an android application we developed. Around the end of the second week, we had the Geogram sending locations, which were translated by the server into locations that are put on the google map as markers. We also have the server communicating with the different components of our system. The key logger has also seen some improvements this week. It can now more accurately identify and capture when a key is modified by shift or capslock. At this point we chose our code and freeze date to be March 1, 2014. From this point on, until the end of Cycle One we are focusing on finishing documentation and preparing for our presentation. Our next goal is to being working on incorporating the functionality of our system to the website to allow users to access and modify all of the information our system keeps track off. We also need to optimize communication between all of the devices and the server.



## Planned Code/Feature Freeze

March 1st was our Cycle One code/feature freeze

# Risk Mitigation

In this cycle we more strongly enforced regular commits to GitHub. Last cycle some members would infrequently commit, and then have difficulties merging files. We felt checking in nightly is the longest time period that is acceptable and reduces the likelihood of conflicting changes. This change improved our workflow by ensuring we were all on the same build.

# Test Plan and Test Procedure

## Test Plan

## Test Procedure

### SMS Gateway Tests

By: Charles Baker

These tests used the SimpleSmsServer in the Testing folder. This is a simple TCP server that prints out received messages to console and return the same message over the TCP connection. The server will also disconnect if the client has not sent a message in a sufficient amount of time and wait for a new connection. The server can be stopped to force an unexpected disconnect on the client side.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Action | Expected Result | Actual Result | Notes |
| 1 | User launches app | The app opens with IP and port fields empty and button labeled “Connect” | Correct on first opening, but retains field information if closed on apps menu |  |
| 2 | User presses connect with leaving either IP or port blank | The app alerts the user that fields must be filled in with the server info | Attempts to connect |  |
| 3 | User presses connect with incorrect (but not blank) IP address | The app changes to and stays in connecting state, the “Connect” button changes text to “Disconnect”, and text fields cannot be edited | as expected |  |
| 4 | User presses connect with incorrect (but not blank) port | The app changes to and stays in connecting state, the “Connect” button changes text to “Disconnect”, and text fields cannot be edited | as expected |  |
| 5 | User presses disconnect while the app is connecting | The app goes into a disconnecting state then back to its initial state. The button changes from “Disconnect” to “Connect” and the text fields become editable. | as expected |  |
| 6 | User presses connect with valid server information entered | The app goes into the connecting state then to the connected state. The “Connect” button changes to “Disconnect” and the text fields cannot be edited. | as expected |  |
| 7 | User presses Disconnect while the app is connected to a server | The app goes into a disconnecting state then back to its initial state. The button changes from “Disconnect” to “Connect” and the text fields become editable. | as expected |  |
| 8 | The app is connected to a server and receives an SMS message | The app sends a TCP message to the server in the format [<phone number>]<message>| | as expected |  |
| 9 | The app is connected to a server and receives a TCP message in the format [<phone number>]<message>| | The app sends an SMS message to <phone number> containing the message <message> | as expected |  |
| 10 | The app is connected to a server and receives a TCP message in an incorrect format (format other than [<phone number>]<message>|) ending in “|” | The app does not send an SMS message | as expected | A message is considered all the characters received until a “|” is received. |
| 11 | The user changes the orientation of the phone while the app is open | The app remains in portrait orientation | as expected |  |
| 12 | User does not interact with the app | The app prevents screen from locking | as expected |  |
| 13 | The app loses focus while not connected and then regains focus | The app retains any information in text fields | as expected |  |
| 14 | The app loses focus while either connected, connecting, or disconnecting | The app continues operation as it would when it has focus | as expected |  |
| 15 | The app regains focus after having changed connection status while it is disconnected | The GUI reflects the correct connection status when it regains focus | as expected |  |
| 16 | The app sends an SMS message to the server via TCP | A received counter on the screen is incremented |  | unimplemented |
| 17 | The app sends a TCP message from the server as an SMS message | A sent counter on the screen is incremented |  | unimplemented |

### SMS Gateway Review

It is currently in a state that can be used by the server. A few more features could be implemented to relay more data to the user. We believe that because of time constraints, the tradeoff for implementing these features would be a net loss. We have decided to mark it as complete, and if time permits revisit it in the future to polish it.

### Windows Service Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Action | Expected Result | Actual Result | Notes |
| 1 | Service connects to server | Server sends MAC address as identifier | as expected | Example shown below |
| 2 | Service is sent a start key-logging command | Service turns on key-logging | as expected | Example shown below. Proof by inclusion of the typed characters after key-logging was enabled being in the key-log file requested. |
| 3 | Service is sent a stop key-logging command | Service turns off Key-logging | as expected | Example shown below. Proof by the exclusion of characters in the requested key-log file typed after turning key-logging off. |
| 4 | Service is sent a request key-log file command | Service returns key-log, possible in multiple segments, in the form <op code><log><newline> | as expected | Example shown below |
| 5 | Service is sent an IP trace route command | Service response with trace route in format <op code><list of IPs delimited by ‘~’><newline> | as expected | Example shown below |
| 6 | Service unintentionally disconnects when flagged as stolen | Reconnects quickly | as expected | Example shown below |
| 7 | Service is sent a not stolen command | Disconnects and waits before checking in again | as expected | Example shown below.  Service waited approx. 1 minute before reconnect. Time will be increased on final version |
| 8 | Service unintentionally disconnects while flagged as not stolen | Reconnects quickly | as expected | Example shown below |
| 9 | Service has no internet connection | Connects to open or known Wi-Fi to attempt to gain internet access |  | unimplemented |
| 10 | Windows startup | Service runs automatically | as expected |  |

By: Charles Baker

The Windows service was tested with the SimpleServer project in the Testing folder. It performs a series of tests where the service connects and interactsin a scripted manor. It is assumed that the key-logger works for purposes of turning it on, off, and requesting its files. Below the table is one instance of the tests’ console output.

===============================================

Testing connection and MAC address sending...

Connected

485B3973C455

Disconnected

===============================================

Testing turning keylogger on/off and receiving key log files...

Connected

Type here (off): 123 AbC

Turning keylogger on

Type here (on): 456 dEf

Turning keylogger off

Type here (off): 789 GhI

Requesting keylog...

op code: 3

Keylog: 456 dEf

Disconnected

===============================================

IP trace route...

Connected

Requesting trace route...

op code: 2

IP trace route: 127.0.0.1

Disconnected

===============================================

Testing reconnect times...

Connected

Not flagged as stolen

Disconnected

Connected

Reconnect time (ms): 500

Reporting stolen

Disconnected

Connected

Reconnect time (ms): 500

Reporting not stolen

Disconnected

Connected

Reconnect time (ms): 60005

Disconnected

### Windows Service Review

The Windows service is not complete because it does not automatically connect to Wi-Fi or have an installer yet. Otherwise, progress made on it has put it in a good state. For the next cycle, improvements will be a low priority to working on the server.

### Database Unit Testing

By: Leo Reyes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Test Case | Expected Results | Actual Results | Notes |
| 1 | parsePayload([]byte{“Param1”, 0x1B, “Param2”, 0x1B, “Param3”, 0x1B}) | Array of strings: {“Param1”, “Param2”, “Param3”} | As expected |  |
| 2 | IsDeviceStolen(string) | Returns true if stolen flag is set in database |  | Not fully implemented |
| 3 | VerifyAccountInfo(“Test@Test.com”, hashedPassword1) | True, true | As expected |  |
| 4 | GetUserDevices(“Test@Test.com”) | []string{“Stolen Laptop Test”, “Stolen GPS Device Test”} | As expected | Not fully implemented |
| 5 | GetAccountInfo(string) |  |  | Not tested |
| 6 | GetDeviceInfo(string) |  |  | Not tested |
| 7 | GetCustomerInfo(string) |  |  | Not tested |
| 8 | GetIpList(string) |  |  | Not tested |
| 9 | GetKeylogs(string) |  |  | Not tested |
| 10 | GetCoordinateInfo(string) |  |  | Not tested |
| 11 | SignUp(string, string, string, string, string) |  |  | Not tested |
| 12 | GetAllCustomers() |  |  | Not yet implemented |
| 13 | GetAllAccounts() |  |  | Not yet implemented |
| 14 | GetAllDevices() |  |  | Not yet implemented |
| 15 | UpdateAccountInfo(string, string, string, string, string) |  |  | Not yet implemented |
| 16 | UpdateDeviceInfo() |  |  | Not yet implemented |
| 17 | UpdateCustomerInfo() |  |  | Not yet implemented |
| 18 | NewIpList([]string) |  |  | Not yet implemented |
| 19 | UpdateCoordinates() |  |  | Not yet implemented |
| 20 | SetDeviceToStolen(string) |  |  | Not yet implemented |
| 21 | Connect() |  |  |  |
| 22 | Disconnect() |  |  |  |

1 The password is created using SHA1. The string password is concatenated to the username (email), and then hashed.

### Database Review

The database has not been reviewed because it is not sufficiently complete.

### Website Blackbox Testing

By: Denney Burkholder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Action | Expected Result | Actual Result | Notes |
| 1 | User enters URL | Home page | As expected | URL: localhost:8080/home |
| 2 | User fills Sign Up form and submit | New account and customer created on database | As expected |  |
| 3 | User access control: before log-in | Error page: not logged in | As expected |  |
| 4 | User fills login form and submit | Rerouted to user map page | As expected |  |
| 5 | User map receives updates | Markers on map updating | As expected |  |
| 6 | User logs out | User redirected to home page and logged out of session | As expected |  |
| 7 | User incompletely fills Sign Up form and submit | Login Page with error message to fill in expected values that were empty. | As expected |  |
| 8 | User incompletely fills Login form and submit | Routed to login page with error message to fill in expected values that were empty. | As expected |  |
| 9 | User gives username that does not exist | Routed to login page with error message that an account with that user name does not exist | As expected |  |
| 10 | User gives a password that doesn’t accompany the given username | Routed to login page with error message that incorrect password for that user name was given | As expected |  |
| 11 | User fills out signup form and gives and invalid email address | Routed to login page with error message that the email address is invalid | As expected |  |
| 12 | User fills out sign up form and gives an email address that is already linked to an account | Routed to login page with error message that the email address already exists in the system |  | Not yet implemented. |
| 13 | User fills out sign up form and gives a phone number that is already linked to an account | Routed to login page with error message that the phone number already exists in the system |  | Not yet implemented. |
| 14 | User fills out sign up form and gives a phone number that is too long or too short | Routed to login page with error message that the phone number is invalid |  | Not yet implemented. |
| 15 | User fills out sign up form and password and confirm password don’t match. | Routed to login page with error message that password and confirm password don’t match |  | Not yet implemented. |
| 16 | User is rerouted to Home Page | Home page shows up displaying the location of all the users registered devices on the map |  | Not fully implemented |
| 17 | User selects a specific device | User rerouted to the map page displaying the location of the device, time last updated, and additional view options |  | Not fully implemented |
| 18 | User changes laptop device status to stolen and hits submit | Device status updated in database. Service beings tracking device, and logging keys in database. |  | Not fully implemented |
| 19 | User changes geogram device status to stolen and hits submit | Device status updated in database. Service beings tracking device. |  | Not fully implemented |
| 20 | User changes laptop device status to not-tracking | Device status updated in the database. Service discontinues tracking the devices, and stops logging the keys. |  | Not fully implemented |
| 21 | User changes geogram device status to not-tracking | Device status updated in the database. Service discontinues tracking the devices. |  | Not fully implemented |
| 22 | User selects keylogger. | Keylogger is pulled from the database and displayed over the specific device page. |  | Not fully implemented |
| 23 | User selects clear keylogger | User prompted to confirm or cancel delete keylogger |  | Not fully implemented |
| 24 | User selects confirm clear keylogger | Keylogger information cleared in the database. User rerouted to specific device page |  | Not fully implemented |
| 25 | User selects keylogger close | Keylogger closes and the display returns to just the specific device page. | As expected |  |
| 26 | User selects rename | User is prompted for a new name. | As expected |  |
| 27 | User enters new name and hits submit. | Device name is changed in the database and user is returned to specific device page |  | Not yet implemented. |
| 28 | User selects delete | User is prompted whether they would like to delete the specific device | As expected |  |
| 29 | User selects confirms device delete | Device information is removed from the database. Rerouted back to home. |  | Not fully implemented |
| 30 | User selects toggle previous location | Previous device location information is displayed on the map. |  | Not fully implemented |
| 31 | User deselects toggle previous location | Previous device location information is removed from the map | As expected |  |
| 32 | User selects update location | Device is relocated, new information is entered into the database, and updated location is displayed on the map | As expected |  |
| 33 | User selects all devices | Rerouted to the home page displaying all the locations of every device associated with the account |  | Not yet implemented. |
| 34 | User selects add new device | User Rerouted to add new device form. |  | Not yet implemented. |
| 35 | User fills out new geogram device form validly and completely | New device added to database. User rerouted to specific device page. |  | Not yet implemented. |
| 36 | User fills out new device form and uses a name already used for one of their devices. | User routed to new device form with message indicating device name already exists with this account |  | Not yet implemented. |
| 37 | User fills out new device form and selects geogram device. Enters phone number that already exists as a geogram device. | User routed to new device form with message indicating device phone number already exists for a different geogram device |  | Not yet implemented. |
| 38 | User fills out new device form and selects laptop device. Enters IP address that already exists for a different laptop device. | User routed to new device form with message indicating device IP address already exists for a different laptop device |  | Not yet implemented. |
| 39 | User fills out new laptop device form validly and completely | New device added to database. User rerouted to specific device page. |  | Not yet implemented. |
| 40 | User selects account | User rerouted to account information page. |  | Not yet implemented. |
| 41 | User selects edit account info. | User rerouted to edit account information form |  | Not yet implemented. |
| 42 | User makes valid changes to their user info. Selects save | User information updated in database. User rerouted to account information page. |  | Not yet implemented. |
| 43 | User makes invalid changes to their user info. Selects save. | Appropriate error message displayed on edit account information form page. User information not updated. |  | Not yet implemented. |

#### Testing Screen Captures

By: Leo Reyes



Password: hello

Figure 1: Test #2 form filled before submit



Figure 1.1: Test #2 customer table after sign up submitted

Relation

Figure 1.2: Test #2 account table after sign up submitted

Hashed password



Figure 2: Test #3 Error page



Figure 3.1: Test #4 login form before submit



Figure 3.2: Test #4 login after submit, redirected to map page

### Website Review

The website cannot be fully tested with the incomplete status of the server. It will be worked on as the server’s functionality allows it. The next cycle will focus heavily on improvements to the Website and server.

### Key-logger Blackbox Testing

By: Nathan Plotts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Action | Expected Result | Actual Result | Notes |
| 1 | User types “aabbccxxyyzzAABBCCXXYYZZ” (using caps lock for capital letters) | Keylog text file has output of “aabbccxxyyzzAABBCCXXYYZZ” | Passed |  |
| 2 | User types “aabbccxxyyzzAABBCCXXYYZZ” (using shift for capital letters) | Keylog text file has output of “aabbccxxyyzz[CAPSLOCK]AABBCCXXYYZZ” | Passed |  |
| 3 | User types “abcABC123!@#” (using shift for capital letters and symbols) | Keylog text file has output of “abcABC123!@#” | Passed |  |
| 4 | User types “-=[];’,./\_+{}:”<>?” (using shift for alternate symbols) | Keylog text file has output of “-=[];’,./\_+{}:”<>?” | Passed |  |
| 5 | User presses left arrow, right arrow, up arrow, down arrow | Keylog text file has output of “[ARROWLT][ARROWRT][ARROWUP][ARROWDN]” | Passed |  |
| 6 | User presses left control, left alt, enter, backspace | Keylog text file has output of “[CTRL][ALT][ENTER][BKSPC] | Failed | Returned “[CTRL][ENTER][BKSPC]” |
| 7 | User presses “12345” on the numpad with num lock on | Keylog text file has output of “12345” | Passed |  |
| 8 | User presses “12345” on the numpad with num lock off | Keylog text file has output of “[END][ARROWDN][PGDN][ARROWLT]” | Passed |  |
| 9 | User presses control, shift, and g keys at the same time | Keylog text file has output of “[CTRL + SHIFT + g]” | Failed | Returned “[CTRL]•” |

### Key-logger Review

The key-logger is sufficiently implemented. The gains from spending more time on the key-logger would be little. We agree o consider it complete unless we have additional time at the end to do minor fine-tuning.

# Lessons Learned

Initially we had decided to code our server in java. Because all of our group members are skilled with coding in java and all of the java libraries that would be at our disposal, it sounded like a good idea. We then designed our java server and began to code it. After getting into the code, our java was very difficult to implement. While it would have been possible to continue coding our server in java, it was becoming an arduous task. Java didn’t have features that would make coding the server much easier (such as HTTP handling and multithreading capabilities) built into the language like other languages available do. We decided that the costs of learning a new language and refactoring the server was worth the time and effort. Based on our needs, we decided to go with Google’s GoLang.

When designing our server we spent a lot of time discussing many intricate details. Often these discussions would eventually lead to the efficiency of these details, and many times we found ourselves spending way too much time arguing these minute details leading to stand-stills in design development. We realized that spending so much time discussing these details would probably have little to no effect on our final product and that the more obvious and major design decisions were the discussions that were worth arguing over.

# References

[1] "Developing." *Android Developers*. N.p., n.d. Web. 03 Mar. 2014.

[2] "SMS Messaging in Android." *MobiForge*. N.p., n.d. Web. 03 Mar. 2014.

[3] "The Go Programming Language." *The Go Programming Language*. Google, n.d. Web. 03 Mar. 2014.

[4] "Gorilla Web Toolkit." *Gorilla, the Golang Web Toolkit*. N.p., n.d. Web. 03 Mar. 2014.

[5] "Google Maps JavaScript API V3." *Markers*. Google, n.d. Web. 03 Mar. 2014.

[6] "Hook Functions." *Msdn*. Microsoft, n.d. Web. 3 Mar. 2014. <http://msdn.microsoft.com/en-us/library/windows/desktop/ff468842%28v=vs.85%29.aspx>.

[7] "Keyboard Input Functions." *Msdn*. Microsoft, n.d. Web. 3 Mar. 2014. <http://msdn.microsoft.com/en-us/library/windows/desktop/ff468859%28v=vs.85%29.aspx>.

[8] "Keylogger Code in C#.net." *Keylogger Code in C#.net*. N.p., n.d. Web. 03 Mar. 2014. <http://social.msdn.microsoft.com/Forums/vstudio/en-US/ed63b033-663a-4a20-80a5-a732d31e9486/keylogger-code-in-cnet?forum=csharpgeneral>.

[9] Geogram One User Manual

# Appendix