FailSafe

McMaster Engineering Competition 2015

Mikhail, Chamu and Jacoby

Version 04.2, November 14th 2015

Contents

Overview

The FailSafe is a state of the art safe that includes the highest quality components to ensure that it will never be breached. The safe is connected 24/7 to the internet via LTE. It is locked by both a physical combination lock and electronic lock. The electronic lock is disengaged through the safes android companion app loaded onto the owner's phone. It also has several sensors that monitor the safe all day and relay data back to the owner's smart phone. The owner can also request specific data and the system will return up to date data from anywhere in the world. If an anomaly occurs the safe will send a notification to the server and the owner can choose what the safes next action is. A simulator has been designed to test the safe virtually in real time by altering the sensor values and sending data from the safe to the owner's smartphone and vice versa.

Features

The FailSafe contains several sensors including:

- Internal and External Thermometers
- Internal Humidity
- Internal Air Pressure
- Accelerometer
- GPS
- Combination Dial RPM Sensor
- Electronic Scale

It also has an internal battery that can last approximately one week in case the safe is unplugged and maintain system functionality.

The FailSafe uses all of these sensors to detect if someone is attempting to breach the safe, and allows the owner to retrieve the safe in case it is stolen. The owner has a few options when they receive a notification from the safe. They have the ability to:

- Incinerate Contents, as long as safe is closed
- Initiate Lockdown mode, where electronic lock can't be disengaged
- Check state of other sensors
- Contact local authorities to investigate

These options give the owner the ability to ensure that the items they have will be secure and gives the owner the opportunity to act on the potential threat.

Error Control

The FailSafe has plenty of fail safes in place. The system is able to adapt to many problems that tend to occur when operating a system with many sensors and internet connectivity. The FailSafe is able to deal with the owner sending incorrect authentication codes to the server. It sends an alert to the server if a sensor becomes disabled allowing the owner to make a decision based on the remaining information they can get from the functioning sensors.

Simulator

The FailSafe simulator allows the operator to change sensor readings in order to simulate actions in which the safe reacts to. There are also a few preprogrammed simulations that recreate common safe cracking techniques. These include:

- Fire
    - This simulation increases the exterior temperature of the safe over time.
- Drill
    - This simulates a small hole being put into the safe.
- Flood
    - This simulates the safe being dunked or surrounded in water.
- Elevator
    - This simulates the safe being pushed or put in an elevator
- Neil Caffrey
    - This simulates someone attempting to crack the physical lock with old safe cracking techniques. Someone slowly turning the dial and listening to the tumblers.
- Throw
    - This simulates someone attempting to throw the safe or hit it with heavy tools such as a sledge hammer.
- Relocate
    - This simulates the safe being taken moved to another location.

Using these build in simulations and generating new ones allows the user to test to see how the safe reacts to specific security threats to ensure that the safe is properly functioning even when some sensors are disabled.

Conclusion

The FailSafe is an extremely robust and complicated system that gives the owner of the safe full control over what to do in situations that are impossible to plan for. It allows the owner to quickly act when the system is attempting to be breached and is able to function without all of its systems being online.

Appendix A: Operation Class Methods

- Safe ()
    - Safe ()
        - Initializes all variables and sensors
    - void monitor ()
        - Checks for anomalies in sensor data and sends notifications to user smart phone via SafeServer
    - void electricalUnlock ()
        - disengages electrical lock
    - void initiateLockdown ()
        - activates lockdown
    - void endLockdown ()
        - deactivates lockdown
    - void incinerateContents ()
        - increases interior tempeture to 666 degrees C
    - void physicalUnlock ()
        - disengages physical lock
    - void lostConnection ()
        - sets connected to false
    - void resetNotifications ()
        - sets all of the notification status to unsent
    - void unplug ()
        - pluggedIn is false
    - void openSafe ()
        - attempts to open safe
        - if safe is opened send notification if on fully or secure
    - void doorClose ()
        - closes safe door and initiates physical lock
    - void changeSecSettings (int)
        - changes the notification settings of the user
    - int getStatus ()
        - returns an integer that represents the notification settings.
- Sensor()
    - Sensor ()
        - sets active to true
    - void disable ()
        - sets active to false
    - void activate
        - sets active to true
    - *All Sensors Have* String getInitial ()
        - returns initial value of the sensor
- Thermometer extends Sensor

- Thermometer ()
  - current internal and external temps set to default
- Thermometer(float internal, float external)
  - sets current and external to above values
- float getIntTemp ()
  - returns internal temp
- float getExtTemp ()
  - returns external temp
- void setIntTemp (float temp)
  - sets internal temp to temp
- void setExtTemp (float temp
  - sets external temp to temp

- AirPressure extends Sensor
  - AirPressure ()
    - sets default air pressure
  - AirPressure (float ap)
    - sets air pressure to ap
  - void chngAirPressure (float pressure)
    - changes air pressure to pressure
  - float getAirPressure ()
    - gives current air pressure
- Humidity extends Sensor
  - Humidity()
    - sets default hum to 70
  - Humidity (float hum)
    - sets default hum to given hum
  - float getHum ()
    - returns hum
  - void setHum (float humidity)
    - sets hum to humidity
- Accelerometer extends Sensor
  - Accelerometer (float lim)
    - sets current accel to 0
    - sets accel lim to lim
  - float getAccel ()
    - returns accel
  - float getAccelLim ()
    - returns accel limit
  - void moving (float chng)
    - accel = chgn
  - void notMoving ()
    - sets accel to 0
- ComboSensor extends Sensor
  - ComboSensor (float lim)
    - RPM = 0
    - RPM min is set to lim
  - float getRPM ()
    - returns current RPM

- float getLim ()
  - returns rpm Lim
- void changeMin (float lim)
  - changes the min speed to lim
- void opening (float rpm)
  - RPM = rpm

- Scale extends Sensor
  - Scale (float weight)
    - sets current weight to weight
  - float getWeight ()
    - returns weight
  - void setWeight (float weight)
    - sets weight to input

- SafeServer ()
  - SafeServer()
    - Initializes socket and establishes streams
  - private void initServer()
    - Initializes ServerSocket and Socket objects
  - private void establishStreams()
    - Initializes Input and Output Stream objects
  - private void authenticate()
    - Sets authenticate to false or true
  - public String receiveFromDevice()
    - Sets authenticate to false or true

Appendix B: Data Transfer Flowchart

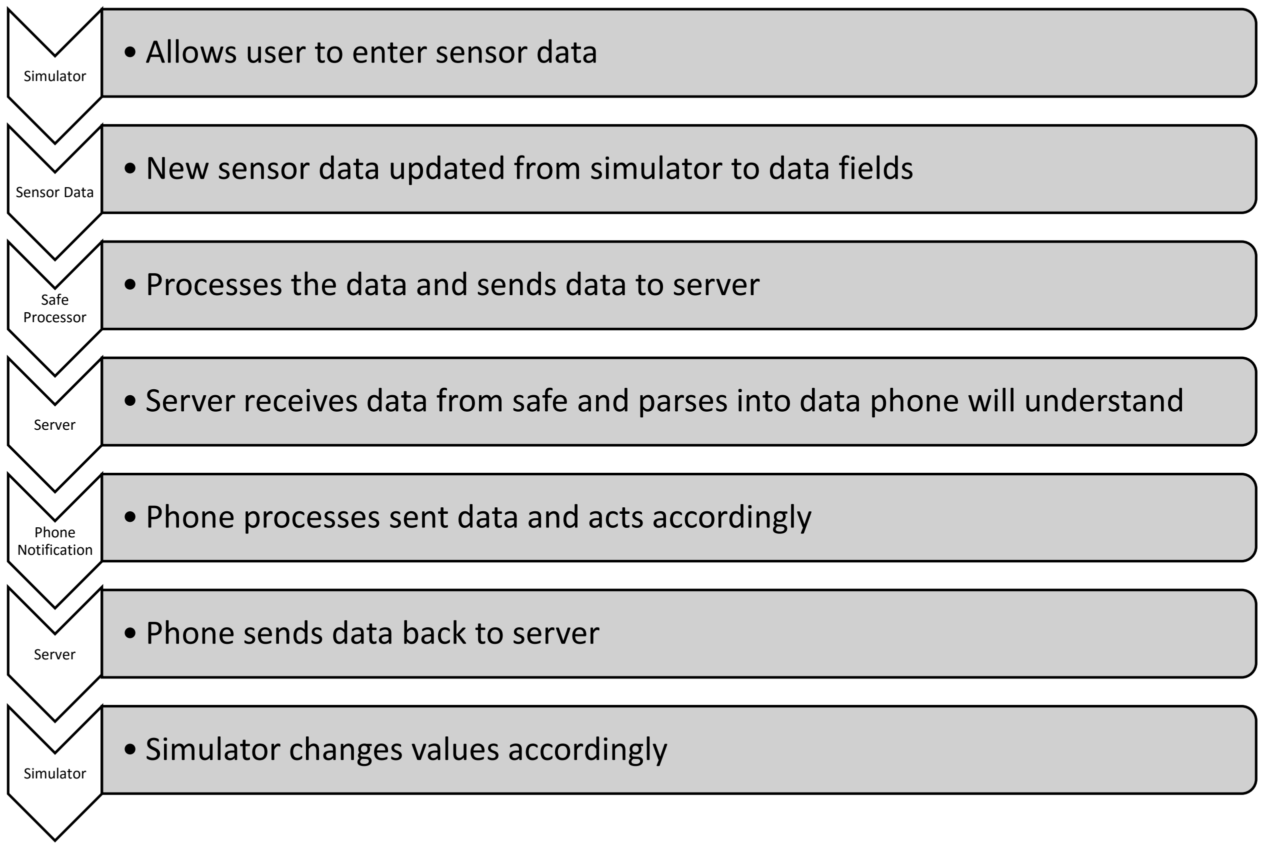
Simulator
• Allows user to enter sensor data
Sensor Data
• New sensor data updated from simulator to data fields
Safe Processor
• Processes the data and sends data to server
Server
• Server receives data from safe and parses into data phone will understand
Phone Notification
• Phone processes sent data and acts accordingly
Server
• Phone sends data back to server
Simulator
• Simulator changes values accordingly

Figure 1: Data Transfer Chart

Appendix C: Additional Resources

Google Docs Link:
https://docs.google.com/document/d/1U3ZBbd8Wen8qlrnxdDJajwJCUP5GwHd9uUzWfQ1H5_I/edit?usp=sharing

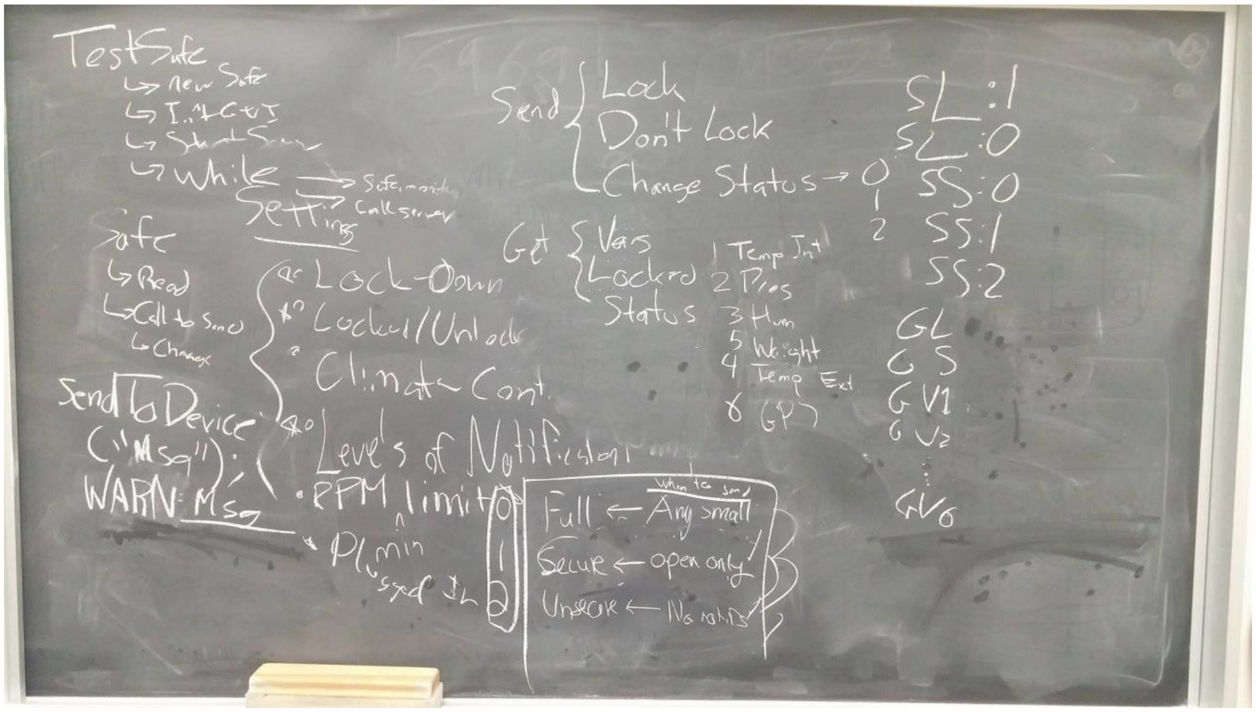

Figure 2: File Transfer Outline