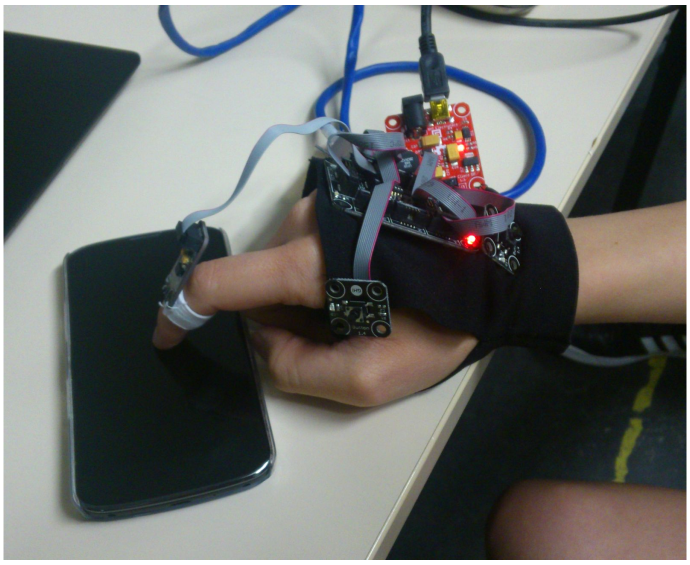
iRing



Contents

Authors:
Praveen Kumar Pendyala - 2919474
Alina Dehler - 2979414
Zahra Fardhosseini - 2806668
Jasmin Henze - 2906982
Ritesh Kumar Bamalwa - 2957766

1. Introduction

In the modern day world each one of us owns a smartphone and at times even more than one. For most of the smartphone users it is not only a device that allows making phone calls or responding to one. It is much more than that: it has eventually become a way to stay in touch with friends, sending messages, photos, videos; a part and parcel of their professional activities such as making notes, managing todo-lists and synchronizing their calendar and have reminders for important meetings; A way to supervise the health status, physical activities or even sleeping habits and several other things.

There is a lot of personal data on a smartphone which should not be accessible for other people. However, the present day smartphones aren't intelligent enough to recognise their current user - in the sense that, anyone has full access to the phone and it's contents if they get hold of it unlocked. The access control offered by most of the present day smartphones is discrete - all or none, and the security is entirely dependent on users properly locking their device when they are not in use by them.

Currently smartphones are locked after a period of inactivity or they should be forced locked by their user. While this may work well in theory, in practice, however, there are regular issues of unauthorised people - friends, co-workers or family, getting hold of users' smartphone before it was locking and thus giving unrestricted access to its contents.

In light of the above problem, we began to look for solutions that may address this problem. Through this project, we aim to prevent unauthorised usage of an unlocked smartphone by anyone other than it's owner.

We begin with our approach to this problem and then introduce you to our proof-of-concept prototype - platform and tools used, realization and then implementation details for the same. The source code of our implementation is available for download.

2. Abstract

To improve the security and privacy we create a system that automatically locks the smartphone whenever it recognizes that a person who is not the smartphone's owner uses it. The owner of the device gets a ring to wear on their finger and it is paired with their smartphone. The ring recognizes which movements are made and sends this information to an app on the smartphone. Simultaneously, the app recognizes the touch movements made on the screen and compares it to the data sent by the ring. As long as they are matching, the user is the authorized person to use the phone and nothing happens. When the movements differ, it is detected as an indicator of foul play and the smartphone will be locked immediately. For

example, when the smartphone's owner has put the phone aside to write a letter and another person takes it and wants to looks up the user's mails.

An added advantage is that our system can also be used to control the smartphone without having to touch it. When switching to remote mode, the movement of the finger with the ring will be used by the app to execute the equivalent touch events. This can be useful for example, when scrolling through a picture gallery without covering up parts of the pictures where the user's finger would touch the display.

3. Platform and Tools

In this section, we discuss about the platform and libraries used in the realization of our solution.

3.1 Platform

1. Smartphone side - Android platform
2. Hardware - Microsoft .NET Gadgeteer[1]
   Microsoft .NET Gadgeteer is an open-source toolkit for building small electronic devices using the .NET Micro Framework.

3.2 Tools and Libraries

1. On Android
   a. Android studio[2] - official IDE for Android development
   b. Appcompat[3] - Android support library
   c. NanoHTTPD[4] - For running HTTP server on Android
   d. Pagerslidingtabstrip[5] - Horizontally swipeable pages

2. On Gadgeteer
   a. FEZ Spider Mainboard[6] - The coordinator of all hardware aspects
   b. 2 Button[7] modules - 1 for mode change and 1 for calibration
   c. Accelerometer[8] module - To detect finger movements
   d. Ethernet[9] module - For cross-platform communication
   e. Router[10] - Bridges Android over WiFi and Gadgeteer in LAN
   f. Microsoft Visual Studio 2013[11]
   g. .NET Micro Framework 4.3 (QFE2)[12]

4. Project setup

Here we provide the details to setup the project from scratch.

Android

1. Import project into Android Studio and run to get an APK.
2. Connect to a router over WiFi and note the IP address received.

Gadgeteer

1. Import Gadgeteer project into Visual Studio - 12 or higher.
2. Open *Modules/EthernetHandler.cs* and update the ServerAddr with the IP address received by the android device.
3. Depending on your Router config, you need to update gadgeteer's configuration.
   Note : The DHCP mode is not working well on Gadgeteer so, static IP has be set
4. Connect the gadgeteer to the same router, using ethernet cable, that Android device was connected to.
5. Run the Gadgeteer project in Visual Studio.

5. Implementation

In this section, we present to you the implementation details of our solution. We start with an overview and then delve into the specific aspects of our implementation codes.

5.1 Overview

Android device continuously tracks user's touch interactions with the device and records them. In hardware, Gadgeteer tracks user's finger movements and sends them to the Android device. In lock mode, Android device looks for matching events and issues a lock signal after certain mismatch threshold. In Remote mode, the application simulates the corresponding touch interaction.

5.2 Specific details

We present an overview our code structure and also discuss some important aspects of each independent module.

Android

We divided our android implementation into 6 independent packages and an AdminReceiver. We present to you below the member classes of each package and their public API.

1. **access**
   a. *AccessController* - Contains everything related to device administration.
      i. checkDeviceAdminRights - Called on application startup. Checks for admin rights and requests them if needed.
      ii. lockDevice - Locks the device.
   b. *EventBox* - Caches all the events and maintains a history of hits and fails. Issues a lock signal when hit rate goes below a predefined threshold.
      i. sendTouchEvent - Caches a new touch event. Updates history.
      ii. sendTouchEvent - Caches a new gadget event. Updates history.
      iii. hitrateChangeListener - To subscribe for changes in hitrate. Used by MainActivity to update title.

2. **gadgeteer**
   a. *ServerHandle* - Handles a HTTP server in background and issues callbacks on new requests, along with the corresponding action.
      i. startServer - Starts a server in background.
      ii. stopServer - Stops the HTTP server and releases the port.
   b. *OnGadgetActionListener* - An interface implemented by the MainActivity and passed to ServerHandle to receive callbacks on actions occurred in the Gadgeteer device.

3. **model**
   a. *Event* - Represents an event - touch or hardware, and the associated data.

4. **touch**
   a. *TouchHandler* - Once applied on a view, by the MainActivity, this class checks for all touch interaction over that view and sends asynchronous callbacks using OnTouchActionListener.
   b. *OnTouchActionListener* - Implemented by the MainActivity to subscribe for touch actions.

5. **ui**
   a. *MainActivity* - The glueing component of all classes. Implements listeners for touch, gadget events and hitrate updates.
   b. *SampleFragment* - A sample page with vertically scrollable grid of images.
   c. *SampleImageAdapter* - Manages the grid of randomly generated images.

6. **util**
    a. *Globals* - Constants across Android and Gadgeteer hardware. Public codes for actions.
    b. *LogUtils* - Builds log tags for any class, manages application log verbosity level at build or run time.
        i. makeLogTag - Makes a log tag, not longer than 23, for a class.
        ii. LOGD - Debug log
        iii. LOGE - Error log
        iv. LOGW - Warning log
        v. LOGI - Information log
        vi. LOGV - Verbose log
    c. *OnHitrateChangeListener* - An interface, implemented in MainActivity, to asynchronously receive updates whenever hit rate changes.

Gadgeteer

The gadgeteer project is divided into 2 packages and 2 main files - Program and Controller.

1. *Program.cs* - All handlers for the gadgeteer modules and a Controller are instantiated.

2. *Controller.cs* - Here all callbacks for accelerometer and button events are set and handled. The events are then sent to the android application using the ethernet handler.

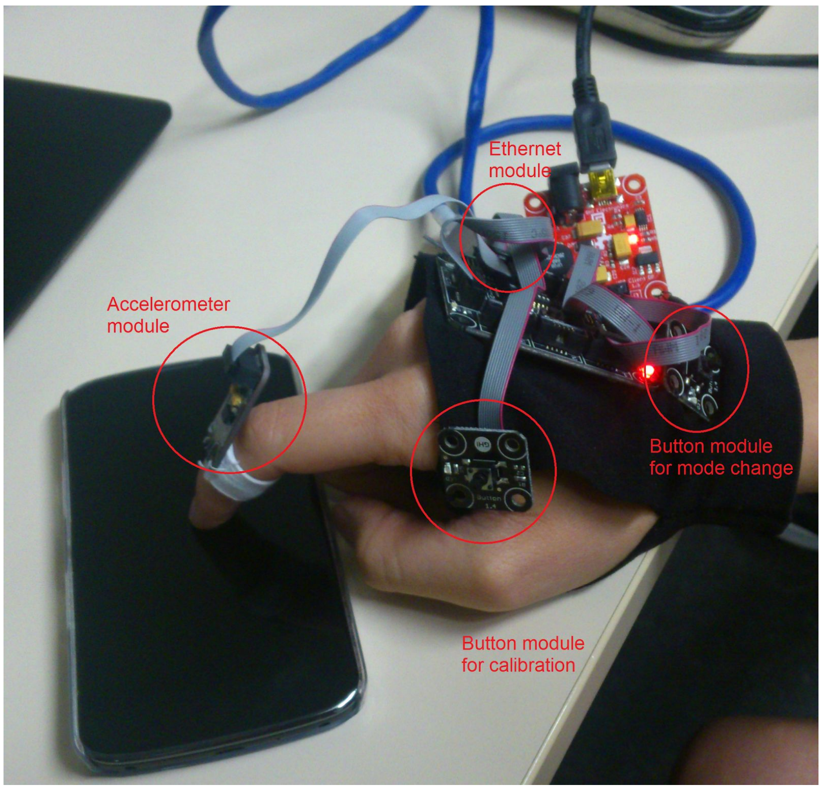
3. **Modules** - Contains the external support modules and their handling.
    a. *AccelHandler* - Handles acceleration events and sends registered movement events to the set callback.
    b. *ButtonHandler* - Handles button events and sends the event to the set callback when a button has been released.
    c. *EthernetHandler* - Used for establishing the ethernet connection between Gadgeteer and Android. Provides asynchronous method to send data to the Android side.
        i. ServerAddr - Constant containing the address of the remote server.
        ii. SendData - Sends data to remote server. Returns immediately and performs data sending in background.

4. **Utils** - Utilities for logging and pattern matching.
    a. *ByteExtensions* - Used for logging the ethernet setup parameters in hex form for better readability.
    b. *PatternMatcher* - Contains a queue with the latest acceleration readings that functions as a window for automatic calibration.

i. *addReading* - Calls checkPattern and adds the current reading to the queue. When no movement event has been registered yet, the reading is calibrated beforehand, otherwise it waits until the event is done.

ii. *checkPattern* - By looking at the acceleration along the x-, y- and z-axis movement events are distinguished and the registered event is returned.

6. Demo



User using their Android device wearing iRing prototype

A video demo of our project in action.

- Lock mode - https://www.youtube.com/watch?v=O4HO1dd9zOc
  The device locks by automatically detecting unauthorized access

- Remote mode - https://www.youtube.com/watch?v=VbVCkYJ6fdI
  Allows remote control over the android device without direct physical interaction

7. References

1. Microsoft .NET Gadgeteer - http://www.netmf.com/gadgeteer/
2. Android studio - http://developer.android.com/tools/studio/index.html
3. Appcompact - https://developer.android.com/tools/support-library/features.html#v7
4. NanoHTTPD - https://github.com/NanoHttpd/nanohttpd
5. Pagerslidingtabstrip - https://github.com/astuetz/PagerSlidingTabStrip
6. FEZ Spider Mainboard - https://www.ghielectronics.com/catalog/product/269
7. Button module - https://www.ghielectronics.com/catalog/product/274
8. Accelerometer module - http://www.seeedstudio.com/depot/Accelerometer-Module-NET-Gadgeteer-Compatible-p-968.html
9. Ethernet module - https://www.ghielectronics.com/catalog/product/284
10. Router - http://www.linksys.com/gb/p/P-E900/
11. Microsoft Visual Studio - https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx
12. .NET Micro Framework - http://netmf.github.io/