FINAL REPORT

# Android:

## Samsung Smartwatch App using CUP:

### 1.1. Overview:

Companion UI Platform (CUP) is a open library which allows you to display information from a host device, for example a smartphone, on companion devices and devices that use CUP. CUP contains many control widgets named winsets, which is useful when you want to create layouts to display information.

The host device control the companion device’s display by sending request to companion devices using Bluetooth. CUP Browser is facilitate to display the CUP winset on wearable devices in many types, and receive interaction events back from the wearable devices.



Figure 1: CUP overview

### 1.2 Architecture:

### 

Figure 2: CUP Architecture

The CUP architecture is simply contained:

* Applications: The applications are built on CUP as main platform.
* CUP API: Components for creating and showing various layouts on the companion device, including the callback interface.
* CUP Service: Service used to connect between the host and companion device.

## CUP library:

### 2.1. Cup Technology:

The purpose of CUP is to provide interaction between a hosting Android device and its wearable CUP Browsers.

* CUP Host consists of the host application built from CUP SDK, and the classes that show more than 12 wingets on a CUP Browser.
* CUP Browser has responsible to receive and perform host request. The CUP winset displays the commands on the CUP Browser and sends back interactive events to the CUP host.

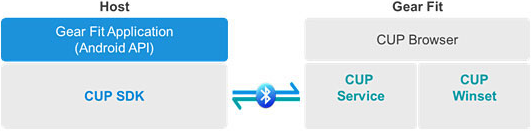
[](https://lazure2.files.wordpress.com/2014/05/image18.png)

Figure 2: CUP Architecture

The CUP process functions as follows:

* The host application sends a command to the CUP browser requiring the Browser to display a certain winset.
* The CUP Browser displays the winset.
* After the user interacts with the winset, the Browser sends the user event back to the host application, which can proceed to the next step.

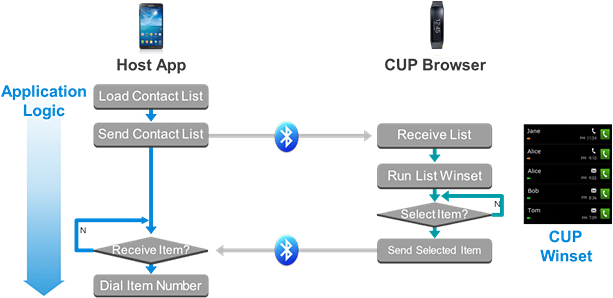
*[](https://lazure2.files.wordpress.com/2014/05/image19.png)*

Figure 3: CUP Interaction

### 2.2. Scup Library Components:

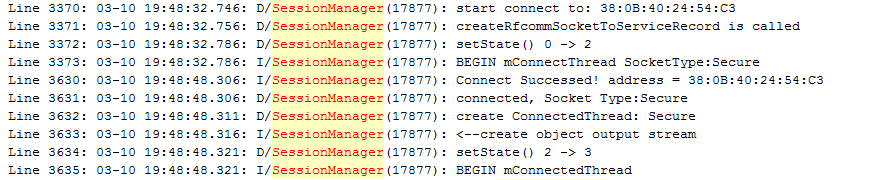
The CUP classes include some important elements:

* Scup: Initializes CUP.
* ScupWidgetBase: Provides basic widget functions in companion devices.
* ScupButton: Provides clickable buttons for the companion UI, with a callback listener for the button clicks.
* ScupDialog: Supports displaying a screen on the companion device. Each dialog can contain multiple widgets.
* ScupLabel: Shows information and images.
* ScupListBox: Shows a list of information.
* ScupSpinner: Shows the same spinner as in Android.

## Connecting App:

## Bluetooth Connecting:

To identify Bluetooth connection process between android smartphone and gearfit. We use android system log which is collected while connecting.

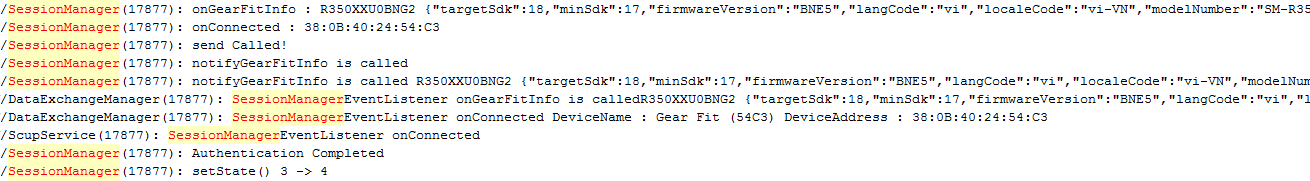


At the beginning, Bluetooth set up a normal Bluetooth connection between 2 devices via RFCOMM channels using UUID equal "9c86c750-870d-11e3-baa7-0800200c9a66". After success creating channel, 2 devices perform authentication state:

* Gearfit send message contain its information:

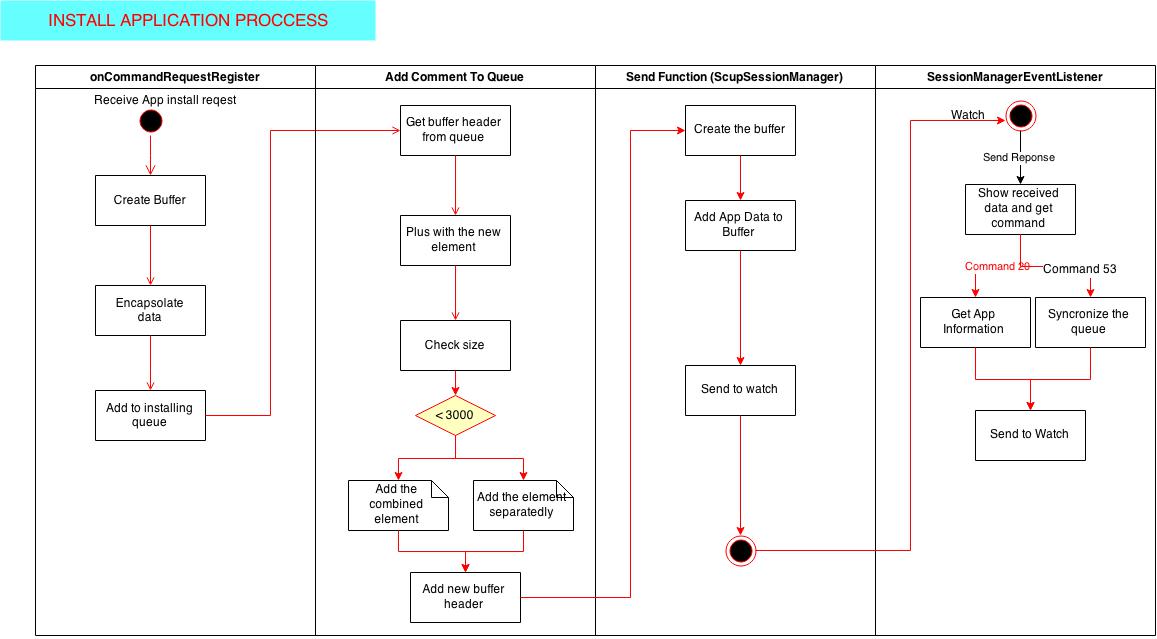
R350XXU0BNG2{"targetSdk":18,"minSdk":17,"firmwareVersion":"BNE5","langCode":"vi","localeCode":"vi-VN","modelNumber":"SMR350","firmwareFilename":"UPDATEMODE\_R350OXA0BNE5\_R350XXU0BNE5.tar"} XEF RFAF40215TF

* Phone received this information and check geafit firmware need update or not. It will send query result to gearfit to finish authentication state.



## Gear fit installing process:

Base on the log file dump from android phone via adb process and reverse APK file. Filter some critical information about the Scup service which is the main service using to control the installing application process. We built the process:

* 

## 

# Reverse APK file:

## APK reverse process:

### Basic approach:

We use dex2jar tool to generate Java Code from unknown APK file. Once JAR is generate, we use JD-GUI which is a standalone graphical utility that displays Java source codes of “.class” files. You can browse the reconstructed source code with the JD-GUI for instant access to methods and fields.

### Virtuous Ten Studio:

Virtuous Ten Studio (VTS) is a free application to be the ultimate solution for the **modification of android applications**. This program allows you to **manage entire Android projects** within an easy to use and familiar environment.With this application we can easily decompile, edit and recompile any apk or jar file.



## Android application Log:

To connect to android device, we use Android Debug Bridge (adb) is a versatile command line tool that is a client-server program that includes three components:

* A client, which runs on your development machine. You can invoke a client from a shell by issuing an adb command. Other Android tools such as the ADT plugin and DDMS also create adb clients.
* A server, which runs as a background process on your development machine. The server manages communication between the client and the adb daemon running on an emulator or device.
* A daemon, which runs as a background process on device instance.

After connect successfully, we use The Android logging system which provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command. You can use logcat from an ADB shell to view the log messages.

# Reverse Firmware:

## Hardware overview:

Then have a look at the internals (teardown):

* 1.84” Curved Super AMOLED touchscreen display (432 x 128 pixels)
* 180 MHz ARM Cortex-M4 CPU
* Accelerometer, gyroscope, and heart rate sensor (an optical one, see right:)
* Battery good for 3-4 days of normal use
* Bluetooth 4.0 LE



Figure : Gear Fit Product Specification

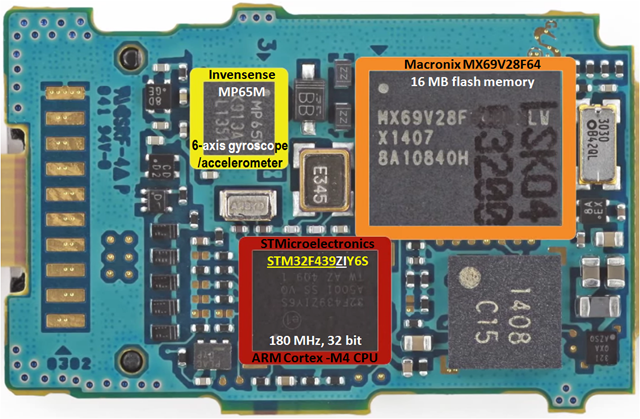


Figure : Gear Fit main board

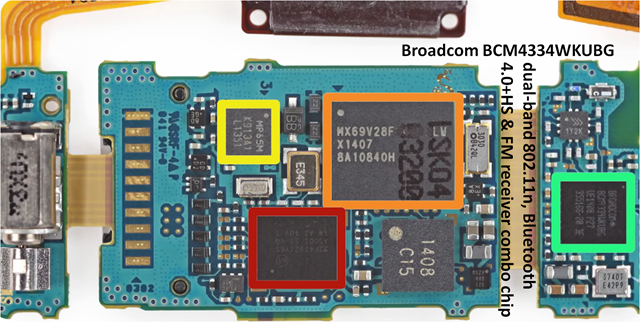


Figure : Gear Fit Bluetooth Chip

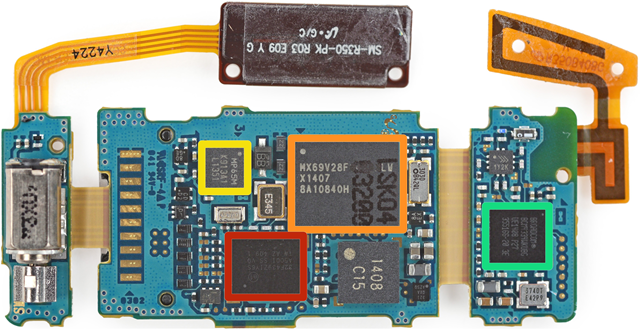


Figure : Hardware Overview



Figure : Hardware main component

Only the following key components are considered here:

* The STM32F439ZIY6S High-performance Microcontroller of the STM32 F4 Series MCUs from STMicroelectronics
* The MPU-6500 Six-Axis (Gyro + Accelerometer) MEMS Motion Tracking™ Devices from InvenSense (marked as MP65M)
* BCM4334WKUBG Single-Chip Dual-Band Combo Device Supporting 802.11n, Bluetooth 4.0+HS & FM Receiver from Broadcom

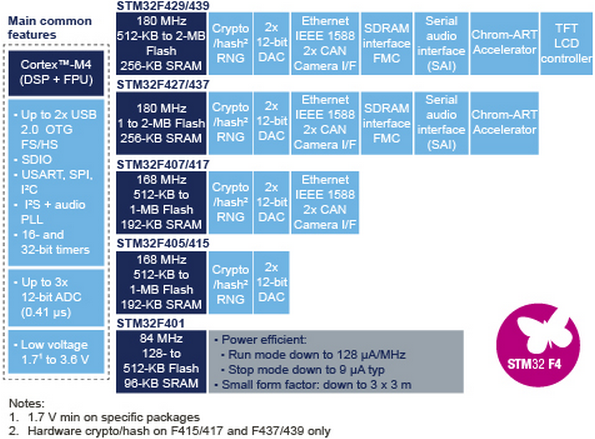


Figure 3: Chip compare version

The high-performance [STM32F429/439](http://goo.gl/9Yu6ok) series now entering production has the industry’s highest performing ARM®Cortex™-M4 core, at 180MHz, able to achieve 225DMIPS (Dhrystone MIPS) and 608 CoreMark scores using industry-standard performance metrics thanks to the ST adaptive real-time accelerator (ART Accelerator) allowing zero-wait execution frSRAM .om Flash. These devices offer up to 2Mbyte of dual-bank Flash allowing safe system upgrades in the field, ST’s unique Chrom-ART Accelerator™ giving customers a competitive edge in graphics processing, and an integrated TFT-LCD controller.

The [STM32F427/437](http://goo.gl/wvHIw0) series [announced [in November 2012](http://www.st.com/web/en/press/en/p3357) with the 168MHz core which entered full production [in February 2013](http://www.st.com/web/en/press/en/p3393)] is also entering full production, upgraded with the 180MHz core, dual-bank memory, and other features of the STM32F429/439 excluding the TFT-LCD controller.

Also entering volume production is the STM32F401 microcontroller announced [in April 2013](http://www.st.com/web/en/news/n3418). The device balances high performance (105 DMIPS and 285 CoreMark, and zero-wait Flash execution with the ART Accelerator), power efficiency and high feature integration in packages as small as 3x3mm.

The Cortex-M4 processor extends the use of Cortex-M cores to applications requiring intensive mathematical computation,” said Semir Haddad, 32-bit MCU Marketing Manager of ST Microcontroller division. “A product line based on the Cortex-M4 processor will complement our line of STM32 microcontrollers, giving our customers the ability to combine the scalability of STM32 with enhanced signal processing capability.

## IDA plugin:

Although our efforts have improved IDA's initial analysis, there is still a good deal of code that has been missed. We have to write some simple IDA scripts which can be used to get more out of the disassembly.First, we want to locate unidentified functions by iterating through the code looking for common function prologues. If one is found, we'll tell IDA to create a function there.

IDAPython is an IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro. These programs have access to IDA Plugin API, IDC and all modules available for Python. The power of IDA Pro and Python provides a platform for easy prototyping of reverse engineering and other research tools.Our IDAPython script will search the code (starting at the cursor position) for byte sequences that correspond to these instructions, and instruct IDA to convert them to functions.

### Find function:

This Script use to find functions which is not recognized by IDA. We use “BADADD” to definite about address that is valid or not. The script simply browse to all address and make the function if this addess is a “BADADD” and do not have function name.

Main function is called from IDC library:

– idc.GetSegmentAttr(address, attr): Use to get the attribute of segment code at specific address.

– idc.GetFunctionName(address): Use to get function name of function which contains specific addess.

– idc.MakeFunction(address): Create function at specific address.

– idc.NextAddr(address): Move to next address.

Result

### Find String:

This Script use resemble method which use “BADADDR” to definite about address that is valid or not. The script simply browse to all String in the list and create new string if it satisfy 3 conditions:

– Address is higher than first address.

– It is not ASCII.

– We can create new string fromcurrent address to the end String (calculated by OS).

It is also cover remaining data into DWORD by using idc.OpOff and idc.MakeDword functions.

Main function is called from IDC library:

– idc.MakeStr(address, end\_address): Use to create string from address to the end address. If end\_address specify to be BADADDR, end\_address is calculated by OS.

– idc.isASCII(address): Check String at address is ASCII or not.

– idautils.Strings(): Return the list of valible String.

– idc.GetFlags(address): Get 32-bit value of internal flags.

– idc.MakeDword(address): Cover current item to Dword.

– idc.OpOff(ea, n, base): Convert operand to an offset

## Firmware Analize:

### Bootloader:

#### Identify the loading address for the executable code:

Loading the boot loader at address 0, Architecture ARM, the first few lines of firmware:

ROM:00000000 dword\_0 DCD 0x20002DE0

ROM:00000004 DCD 0x20016AAD

ROM:00000008 DCD 0x2000E241

ROM:0000000C DCD 0x2000E245

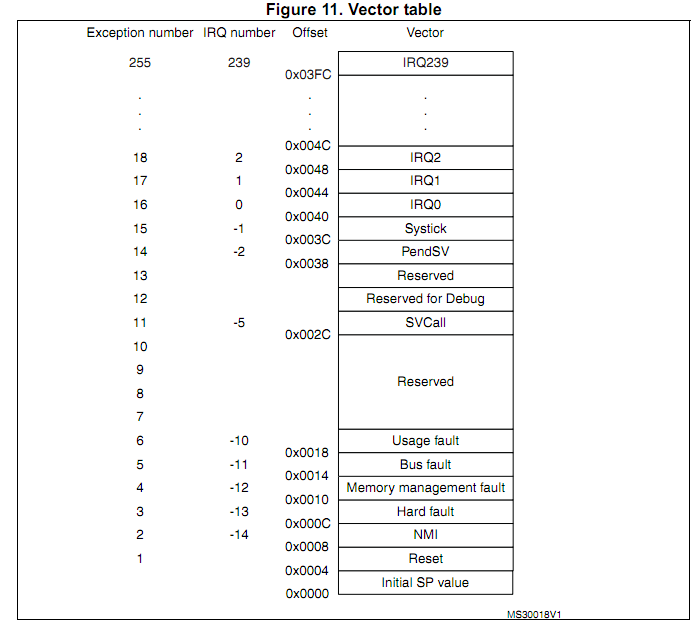
ROM:00000010 DCD 0x2000E2B5

ROM:00000014 DCD 0x2000E32D

ROM:00000018 DCD 0x2000E3A5

ROM:0000001C DCD 0

ROM:00000020 DCD 0



Compare with table vector of Smartwatch chip, we identify that: 0x20002DE0 is the initial stack pointer, and 0x20016AAD is the entry point.

That entry point does not seem to be contained in the boot loader itself (which has 64 K + 256 bytes), but in the on-Chip ROM.

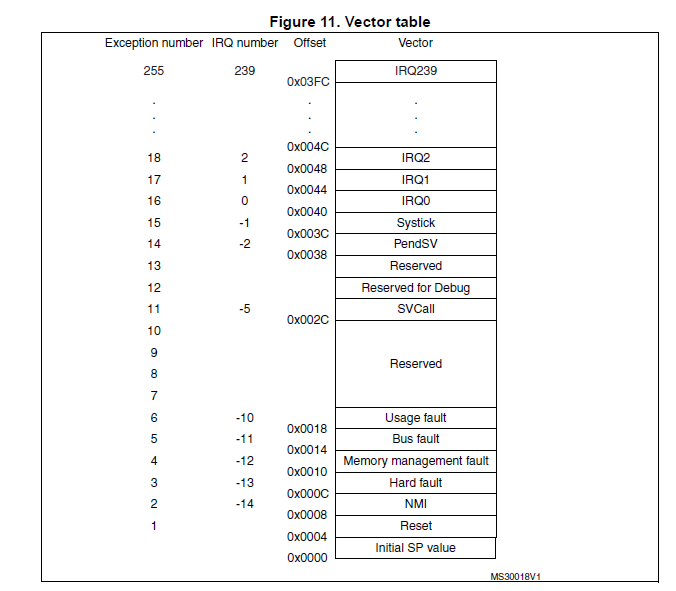
In the modern complex chips such as STM32 F4 Series the first piece of code which is executed is usually not user code but the on-chip Boot ROM. It might check various conditions to determine where to load the user code from and whether to load it at all. In such cases, the user code might have to conform to different start-up conventions.

Unless you can somehow dump the memory at 0x20016AAD, you won't be able to find out what the ROM does, and where within the ROM it jumps. As this explanation, loading bootloader at 0x0 address is the best way in this situation.

## Boot Sequence:

Step 1: The reset

On startup, the processor will jump to fixed location at address 0x0. This address should contain the reset vector and the default vector table. Reset vector is always the first instruction to be executed. The reset vector in this table will contain a branch to an address which will contain the reset code. Normally, at this stage, the rest of the vector table contains just the dummy handler- a branch instruction that causes an infinite loop (this is because this vector table is used very briefly and later on replaced by vector table in RAM after memory remap operation)



Step 2: The reset code

This reset code to which the jump has been executed from the reset vector will do the following tasks:

->Set up system registers  and memory environment

->Set up MMU

->Setup stack pointers : initialize stack pointers for all ARM modes

->Set up bss section : zeroing the ZI data region,  copying initialization values for initialized variables from ROM to RAM

->Set up hw : CPU clock initialization , external bus interface configuration,low level peripheral initialization etc

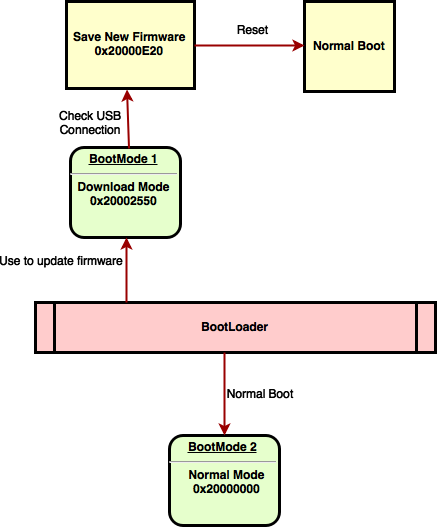
Step 3: Remap Memory

One of the job of the initial reset code will be memory remapping. At the time of power up, the processor jumps at fixed location 0x0.So, this is important to ensure there is some executable code present at this location at the time of power up. And to ensure this, some non volatile memory should be mapped to this address.

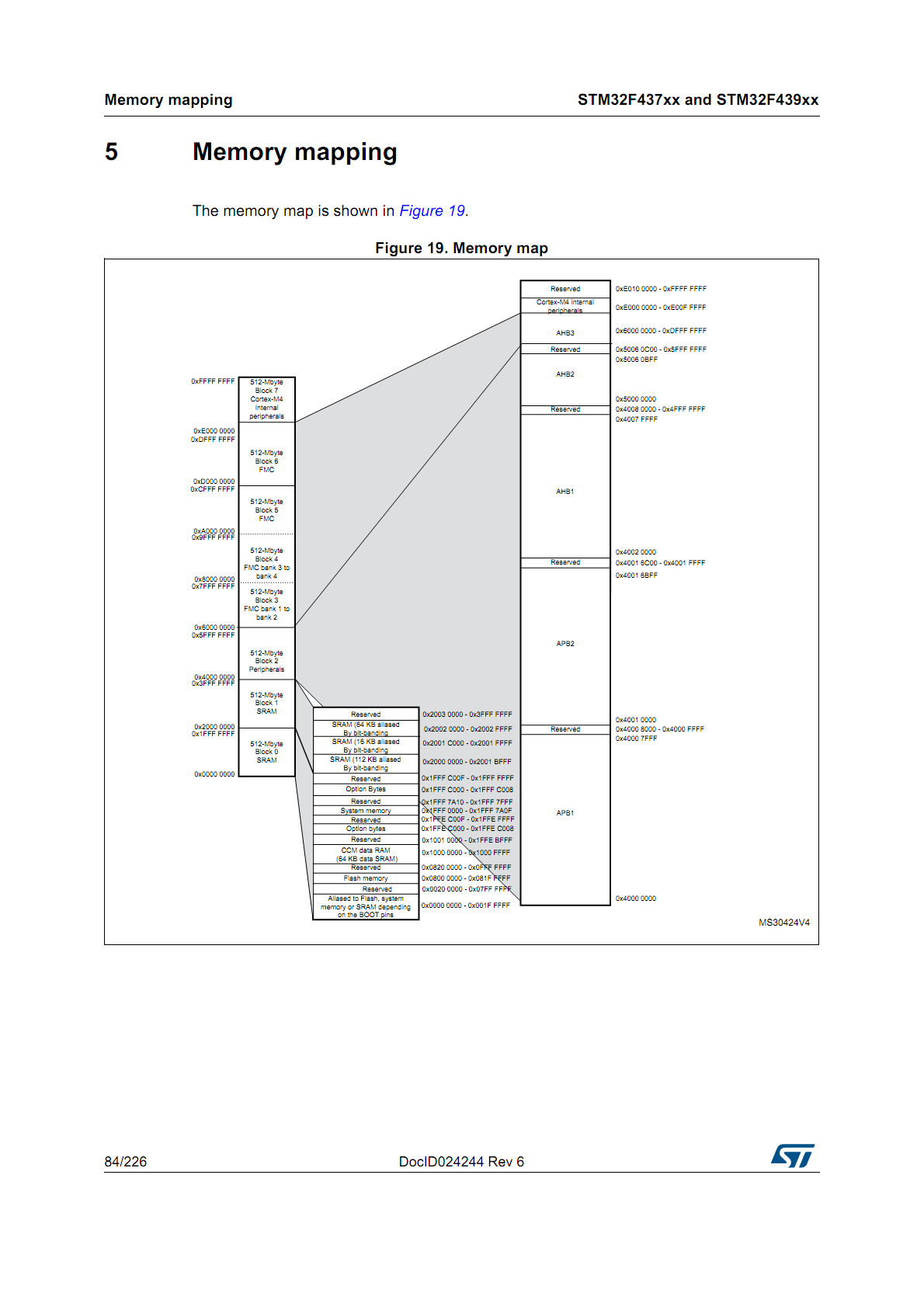
Vector table is located in RAM .However, ROM is located at 0×0 address and then during normal execution RAM is re-mapped to this location. Memory remapping can be achieved through hardware remapping, that is changing the address map of the bus. This can also be done through MMU.

Step 4: Setting up the external memory, loading and executing the OS image

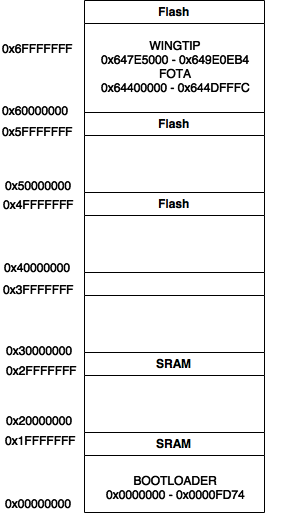
External memory should be setup before loading an image to it (Refresh rate, clock etc),OS image can then be loaded from flash(assuming its NAND flash) to RAM. The OS image may be compressed in which case it needs to be decompressed before PC can be modified to point to the operating system entry point address.



## Memory Map:



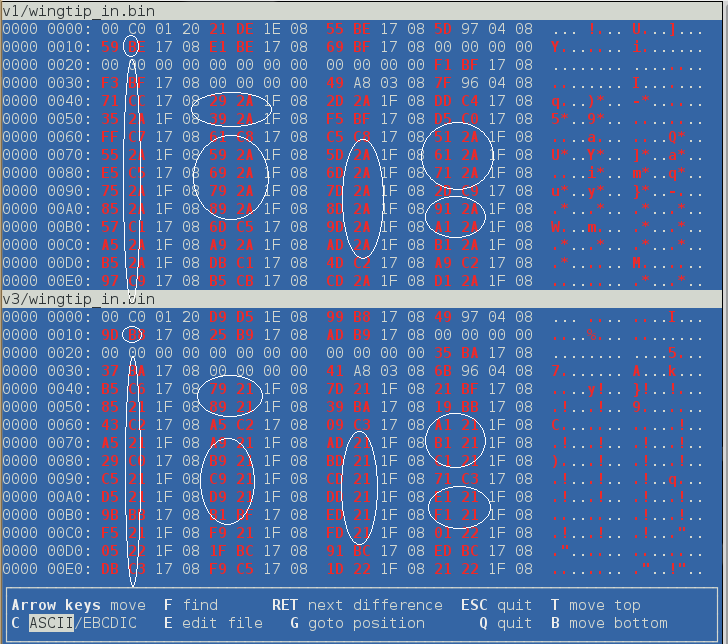
Memory graph after boot:

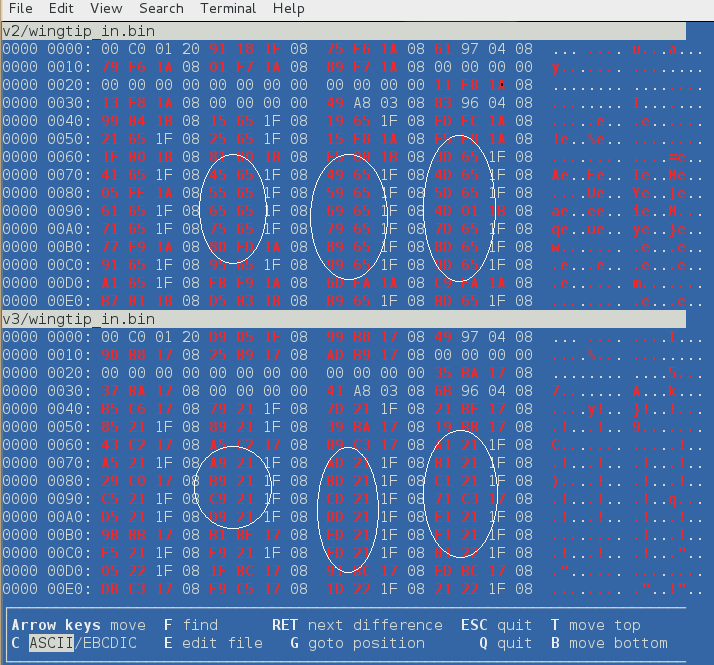


## Reverse Header:

### 6.1. Compare header of the update firmware

* From the internet, we get 3 different versions of the gearfit’s update firmware.
* We used the vfbindiff tool to compare the header of the updates firmware and find some repeat partion.



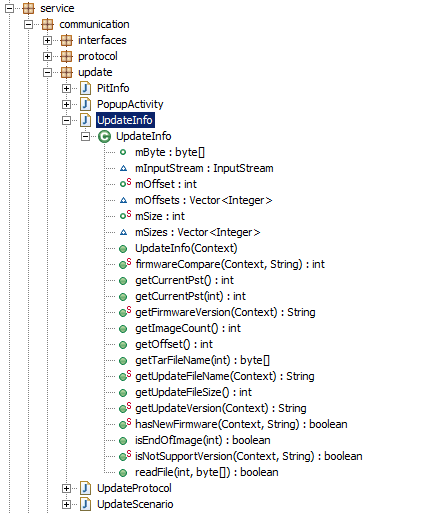


# Vulnerability point:

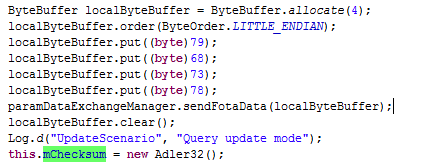
## Modification Attack:

### GearFit Manager Structre:

Using some technique to reverse structure of apk file. We discover a strainge directory named “firmware” which contained a bin file using for update smartwatch firmware directly. Regarding java code, we found some class link to this file.



From this evidence, we conclude that this is file used to update firmware. Subsequently, we continue finding GearFit Manager whether using checksum function or not.



GearFit Manager’s checksum is Adler32 and it implement normal standard support in Android. SamSung create Adler32 checksum value on buffer byte after reading from update firmware file that mean it only ensure integrity while transferring from smartphone to smartwatch. Therefore, we can modify firmware file without concerning about checksum value. With Virtuous Ten Studio tool, we rebuild GearFit Manager apk file with modified firmware and update modified firmware to GearFit Watch.

## Session Hijacking:

## Over view:

The classes in Scup library use to connect to watch apps are private. So the idea is that we will implement a new public class to create a new illegal connection to other watch apps. Scup uses ScupCommunicator class to initiate connection to watch and connection services are only using the package as constructor parameter. We can compromise application by change the target connection.

## Reverse ScupComminication class:

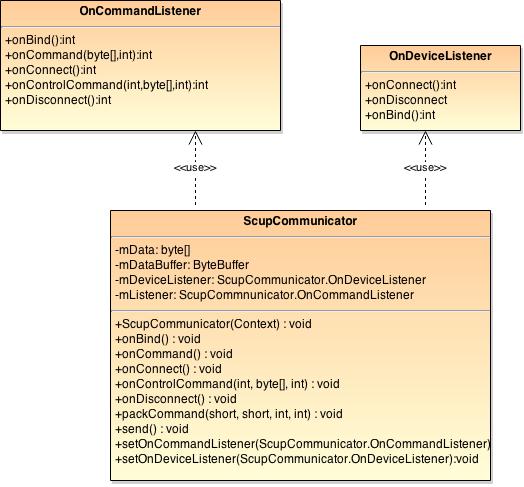


Figure 4: ScupCommunicator Class Diagram

## Our Approach:

The weakness point is that watch does not have any authenticated method when application create service connection to connect with watch application. The parameters are only package name and Samsung service name. After host application connect successful to watch app, we create one more new connection to the other application and send the command. Watch app receive the command and perform because it believes that command come from legal application.

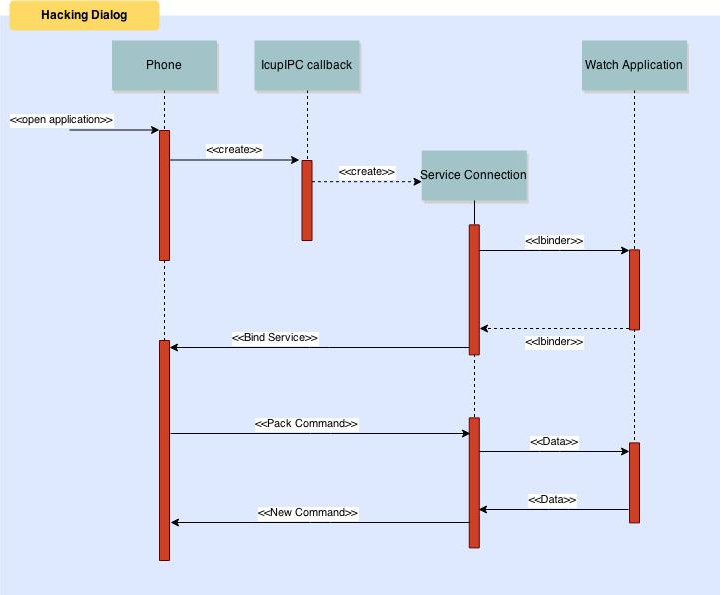


Figure 6: Compromise process