

**CSCI321 – Project (Android Network Sniffer)**

Technical Manual

|  |  |  |  |
| --- | --- | --- | --- |
| Group SS18/2A | Student No | Supervisor | Assessor |
| Soh Yu Xuan | 5498636 | Mr Tian Sion Hui | Mr Tan Kheng Teck |
| Timothy Chin | 5498399 |
| Kenneth Huang | 5498442 |
| Kendrick Tan | 5026556 |

Table of Contents

1. [Proposal 4](#_Toc523493216)

[1. Executive Summary 5](#_Toc523493217)

[2. Project Description 5](#_Toc523493218)

[3. Requirements for Packet Sniffing 10](#_Toc523493219)

[4. Objectives 11](#_Toc523493220)

[5. Development Method – AGILE 11](#_Toc523493221)

[6. Timetable 13](#_Toc523493222)

[7. References 13](#_Toc523493223)

1. [Software Requirements Specification 14](#_Toc523493224)

[1. Introduction 14](#_Toc523493225)

[2. Overall Description 15](#_Toc523493226)

[3. System Features 17](#_Toc523493227)

[4. External Interface Requirements 20](#_Toc523493228)

[5. Other Nonfunctional Requirements 24](#_Toc523493229)

1. [Gantt Chart 26](#_Toc523493230)

[1. Planned Work 26](#_Toc523493231)

[2. Actual Work 26](#_Toc523493232)

1. [Burn Down Chart 27](#_Toc523493233)
2. [Sprint Summary 28](#_Toc523493234)
3. [Technical Information 52](#_Toc523493235)

[1. Document Overview 52](#_Toc523493236)

[2. Introduction 52](#_Toc523493237)

[3. 802.11 Packet 63](#_Toc523493238)

[4. How we implemented libpcap for PCBIN/PCMON 69](#_Toc523493239)

[5. Application Description 70](#_Toc523493240)

[6. Application Program 72](#_Toc523493241)

[8. References 79](#_Toc523493242)

1. [Report 80](#_Toc523493243)

[1. Project Overview 80](#_Toc523493244)

[2. Roles and Responsibilities 81](#_Toc523493245)

[3. Risk Analysis and Counter Measures 82](#_Toc523493246)

[4. UML Diagrams 83](#_Toc523493247)

[5. Test Plan Design 119](#_Toc523493248)

[6. Evidence of VCS (Version Control Software) 126](#_Toc523493249)

# **Proposal**

***Document Control***

Document Name: Proposal.docx

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Owner | Current Version | Last Change on | | Approved by |
| Date | Time |
| Soh Yu Xuan | 0.8 | 17th August | 1138 |  |

***Distribution List***

|  |  |
| --- | --- |
| Name | Title/Role |
| Soh Yu Xuan | Team Leader / Lead Programmer |
| Timothy Chin | Assistant Programmer / Lead UI |
| Kenneth Huang | Assistant Programmer / Lead Tester |
| Kendrick Tan | Project Management |

***Record of Revision***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision Date | Description | Section Affected | Changes Made By | Version after Revision |
| 21th April | Filled Executive Summary & Introduction | Executive Summary & Introduction | Soh Yu Xuan | 0.1 |
| 23rd April | Filled Research | Research | Timothy Chin | 0.2 |
| 24th April | Completed Research & Objectives | Research & Objectives | Soh Yu Xuan | 0.3 |
| 26th April | Completed Scope and Limitations | Scope & Problems of Limitation | Kenneth Huang | 0.4 |
| 28th April | Filled Development Method | Development Method | Kendrick Tan | 0.5 |
| 28th April | Updated Research | Research | Soh Yu Xuan | 0.6 |
| 28th April | Completed Timetable | Timetable | Soh Yu Xuan, Timothy Chin, Kenneth Huang | 0.7 |
| 17th August | Format doc & remove outdated info | All | Timothy Chin | 0.8 |

## 1. Executive Summary

***Summary***

Technology has improved with leaps and bounds, computers used to be large clunky machines which were largely immobile but now, almost everyone if not all has a “computer” with them all the time, a smartphone. One of the jobs of a network administrator requires him/her to monitor the network. This can be done with a computer with either with an external network card or with an internal one, however the computer was something that was needed, adding weight and hassle. What if one could monitor and sniff packets from their device whom they carried with them everywhere? One could simply be in a location, capturing packets in their vicinity for analysis, without having to lug around a laptop right into their mobile device.

***Introduction***

The scope of the project has its limitations. An android device differs from a personal computer, in a great many ways. Applications such as Wireshark or the aircrack suite on a Linux PC are easy to operate and have been around for decades. Such applications, which capture packets, require a Network Interface Card set in Monitor mode. With a large number of drivers easily downloadable and installable on a personal computer, compatibility and configurability between the NIC cards and the OS make it easy and straightforward as compared to on an android device. Android devices themselves differ in a great many ways. The chipsets from different manufacturers are different, resulting in a lack of universal features. Drivers do not work the same as with a personal computer, for an android device, the native drivers are precompiled in the Linux kernel, making compatibility with external WLAN adapters limited at best without a reverse engineer of existing drivers, integration with the kernel and re-flashing to the new OS. These factors, added to the fact that manufacturers do not/are not supporting development with their hardware make this project very limited and experimental. This document will describe in detail the objectives, scope and limitations of the project.

## 2. Project Description

***Background***

The idea of having airodump-ng albeit a limited version of it on a phone thrills us. Even though android devices are largely segregated by their manufacturers, and the result from this project will be fairly limited, there might be a way to implement such an application on a wider scale with enough time. As with airodump-ng, the user can only capture packets in their vicinity within the range of the NIC, and what better way than to use an android phone, something that weighs so little and are most likely on person most times.

***What is packet sniffing?***

“A sniffer (packet sniffer) is a tool that intercepts data flowing in a network. If computers are connected to a local area network that is not filtered or switched, the traffic can be broadcast to all computers contained in the same segment. This doesn’t generally occur, since computers are generally told to ignore all the comings and goings of traffic from other computers. However, in the case of the sniffer, all traffic is shared when the sniffer software commands the **Network Interface Card (NIC)** to stop ignoring the traffic. The NIC is put into promiscuous mode, and it reads communications between computers within a particular segment. This allows the sniffer to seize everything that is flowing in the network, which can lead to the unauthorized access of sensitive data. A packet sniffer can take the form of either a hardware or software solution. [1]

***The Network Interface Controller / Card (NIC)***

To connect to a network, a computer can use a wired connection via Ethernet or wirelessly, which requires a NIC. It can be built into the motherboard or plugged in via USB with an external WLAN adapter. An android device with OTG capabilities (to act as host) with the relevant drivers will be able to detect and use an external WLAN adapter.

***Promiscuous Mode and Monitor Mode***

Monitor mode (RFMON) enables a wireless NIC to capture packets without associating with an access point or ad-hoc network. This is desirable in that you can choose to “monitor” a specific channel, and you never need to transmit any packets. In fact, transmitting is sometimes not possible while in monitor mode (driver dependent). Another aspect of monitor mode it that the NIC does not care whether the CRC values are correct for the packets captures in monitor mode, so some packets that you see may be in fact corrupted. Monitor mode only applies to wireless networks, while promiscuous mode can be used on both wired and wireless networks. Monitor mode is 1 of 6 modes that 802.11 wireless cards can operate in: Master (acting as an AP), Managed (client mode), Ad- hoc, Mesh, Repeater and Monitor mode.

Promiscuous mode allows you to view all wireless packets on a network to which you have to be associated. The need to associate means that you must have some means of authenticating yourself with an access point. In promiscuous mode, you will not see packets until you have associated. Not all wireless drivers support promiscuous mode.

One important difference is Promiscuous mode does tell the card to process all frames (removing 802.11 frame headers) [2]

***IEEE 802.11***

IEEE 802.11 is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication in the 900 MHz and 2.4, 3.6, 5 and 60 GHz frequency bands. They are the most widely used wireless computer networking standards, used in most home and office networks to allow laptops, printers and smartphones to talk to each other and access the Internet without connecting wires. They are created and maintained by the Institute of Electrical and Electronics Engineers (IEEE) LAM/MAN Standards Committee (IEEE 802). The base version of the standard was released in 1997 and has had subsequent amendments. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand. While each amendment is officially revoked when it is incorporated in the latest version of the standard, the corporate world tends to market to the revisions because they concisely denote capabilities of their products. As a result, in the marketplace, each revision tends to become its own standard.

***Layer 2 – Datagrams***

**Management frames**

* Authentication frame
* Association request frame
* Association response frame
* Beacon frame
* Deauthentication frame
* Disassociation frame
* Probe request frame
* Probe response frame
* Reassociation request frame
* Reassociation response frame

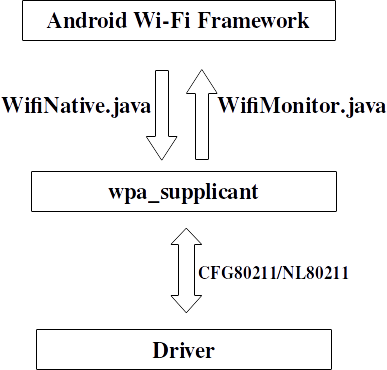
**Control frames**

* Acknowledgement frame (ACK)
* Request to Send frame (RTS)
* Clear to Send frame (CTS)

**Data frames**

Data frames carry packets from web pages, files, etc. within the body. The body begins with an IEEE 802.2 header, with the Destination Service Access Point (DSAP) specifying the protocol, followed by a Subnetwork Access Protocol (SNAP) header if the DSAP is hex AA, with the organizationally unique identifier (OUI) and protocol ID (PID) fields specifying the protocol. If the OUI is all zeroes, the protocol ID field is an EtherType value. Almost all 802.11 data frames use 802.2 and SNAP headers and most use an OUI of 00:00:00 and an EtherType value. [3]

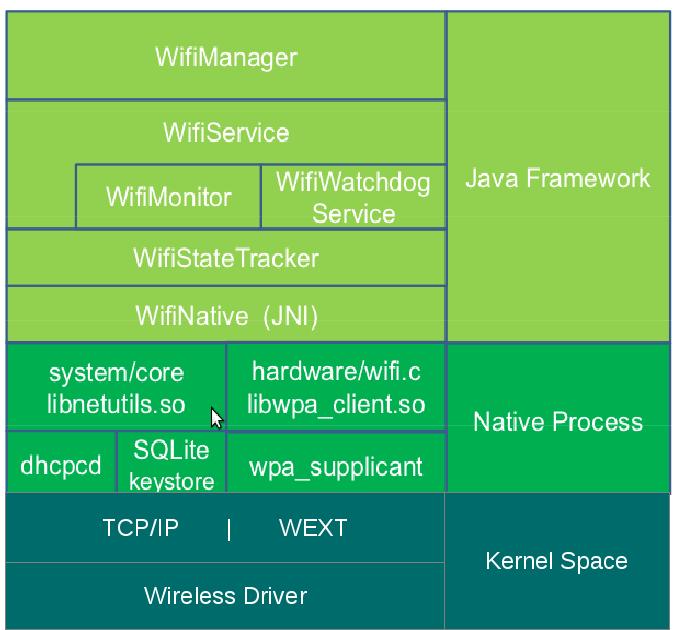
***How Android State Machine Works***



Android uses a customized wpa\_supplicant to perform AP authentication and association, and also communicate with the underlying driver. The **WifiNative** class is used to send various commands to wpa\_supplicant and the **WifiMonitor** class is used to monitor wpa\_supplicant status change and notify the Android framework.

Wpa\_supplicant communicates with the underlying driver using new CFG80211/NL80211 interface [4]

***Android Wifi Architecture***

****

**[5]**

***Root Access on Android***

Root access is not available on a normal android device. Like the sudo command in linux, root access is required to run certain processes in Android. Rooting basically means to give yourself root permissions. However, manufacturers do not support this function at all and by doing so will void the warranty of the device. Rooting of a model of device depends on the community working on it, trying to escalate the default privileges to root. As such, some devices might not have root or exploits available at all.

***PCAP Library – API for packet capturing***

The Packet Capture library provides a high-level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a “savefile” and reading packets from a “savefile”. The pcap API is written in C, with wrapper libraries (Libtins – C++, jNetPcap - Java) available.

***Bcmon/Nexmon – Enabling monitor mode on an android device***

Android devices have their firmware locked down, development of applications which require special privileges are unsupported. Users have to enable root permissions with third party applications which are often experimental, made by enthusiasts who try to reverse-engineer into the device. Bcmon, succeeded by Nexmon is an open project that requires root, it enables devices with select Broadcomm/Cypress WiFi chipsets to enable monitor mode.

## 3. Requirements for Packet Sniffing

***About the Device***

The device we are using: LG NEXUS 5, running Android 6.0.1 Marshmallow. (with a BCM 4339 NIC)

***Requirements***

1. Making sure having a device that has a NIC(Network Interface Card) that supports Monitor/Promiscuous Mode
2. Rooting the device
3. Controlling root privileges(optional, recommended)
4. Modification of Drivers/Kernel to enable Monitor mode

Based on the above research, we have some level of understanding of the context, requirements and limitations of our project. There are two ways to implement packet sniffing with an android device, and both have their limitations. The first way is to connect an external Wifi adapter via an OTG cable (with a device that supports OTG). The limitations to this method is that, depending on the wireless card, specific drivers are required. In android, drivers are precompiled in the Linux kernel, and to update to the required driver, the kernel has to be rewritten with the driver, and the OS has to be re-flashed with the new kernel. This process of reverse engineering of the driver and recompiling of the kernel is extremely difficult and time consuming. Another way is to utilize the in-built NIC of the device. This method is limited by the base device, only a tested device that works will be able to run the sniffer. Both methods require root access to the device, escalating root privileges to the user. Rooting a device is unsupported by the manufacturers and therefore, is highly dependent on open source communities and may differ from model to model. Once a device has a NIC in monitor mode, the application which will be developed, with the help of the PCAP library can then capture packets, save it to a file and possibly provide some sort of an analysis of the data captured.

## 4. Objectives

***We plan to achieve these functionalities***

* GUI
* Capture packets with filtering options
* Manipulate data captured, display data
* Save all captured packets to a file for analysis on PC

***Possible additions of function***

* Cracking WEP, requires packet injection
* Driver reverse engineering and implementation on the application itself instead of re-flashing an updated kernel, allows for usage of external NICs
* Analysis of captured packets & additional functionalities

***Scope and Problems with Limitation***

* Device MUST have a chipset that allows NIC in monitor mode
* Device MUST be rooted
* Bcmon/Nexmon to enable monitor mode

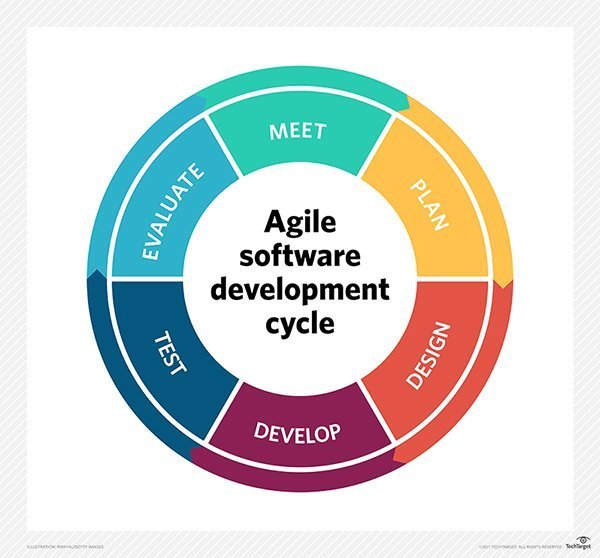
***Choice of language***

* Java / C++ / C

***IDE***

* Android Studio (Java)

## 5. Development Method – AGILE



***AGILE Software Development Method***

* Provides the ability to create and respond to change to succeed in an uncertain and turbulent environment
* We feel that AGILE is the most applicable due to the unpredictability of the project, rooting and enabling monitor mode on specific onboard WLAN cards of chosen devices are experimental and untested.
* There are no true examples on the Google Play Store, true packet sniffing requires root and monitor mode on the onboard NIC, which is unsupported.

***AGILE’s Manifesto***

* Individuals & interactions > Processes & Tools
* Working Software > Comprehensive Documentation
* Customer Collaboration > Contract Negotiation
* Responding to change > Following a plan

***12 Principles of AGILE Software Development Method***

* Customer satisfaction by early and continuous delivery of valuable software
* Welcome changing requirements, even in late development
* Working software delivered frequently
* Close, daily cooperation between clients and developers
* Projects are built around motivated individuals, who should be trusted
* Face-to-face conversation is the best form of communication (co-location)
* Working software is the primary measure of progress
* Sustainable development, able to maintain a constant pace
* Continuous attention to technical excellence and good design
* Simplicity – the art of maximizing the amount of work not done is essential
* Best architectures, requirements and designs emerge from self-organizing teams
* Reflect on how to become more effective and adjust accordingly

## 6. Timetable

|  |  |
| --- | --- |
| * 29th April | * Overall use case |
| * 6th May | * 1st Prototype (Simple GUI) + Documentation (Usecase + Class) (Hello World) |
| * 13th May | * Set up test devices with monitor mode * Understanding PCAP API and relevant libraries |
| * 20th May | * Understanding PCAP API and relevant libraries * Start of 2nd Prototype |
| * 27th May | * Continuation of 2nd Prototype |
| * 3rd June | * End of 2nd Prototype + Documentation (Displaying of raw frames) |
| * 10th June | * Start of 3rd Prototype |
| * 17th June | * Continuation of 3rd Prototype |
| * 24th June | * End of 3rd Prototype (Displaying of packets) + Documentation, Planning of test cases |
| * 31st June | * Debugging, testing for conformity to test cases |
| * 7th July | * Working 3rd Prototype |

## 7. References

1. <https://www.techopedia.com/definition/4113/sniffer>
2. <http://lazysolutions.blogspot.sg/2008/10/difference-promiscuous-vs-monitor-mode.html>
3. https://en.wikipedia.org/wiki/IEEE\_802.11
4. <http://jhshi.me/2014/04/25/how-android-wifi-state-machine-works/index.html#.WuQSE4gRUwo>
5. <https://mitulmodi.wordpress.com/2012/03/21/android-wifi-architecture-wext/>

# **Software Requirements Specification**

***Revision History***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Kendrick Tan | 02/07 | Correction of certain description and explanation | 1.1 |
| Timothy Chin | 17/08 | Format document, updated device and requirement | 1.2 |

## 1. Introduction

***1.1 Purpose***

This document describes the requirement specification for the Android Network Sniffer application, “NetSniffer”, which captures packets through the use of existing packet capturing technologies. It presents a means of viewing packets that are currently in a network and possibly provide some information on the packets. It will explain the Features of the system, Interface, what the system can do, what are the possible constraints and limitation when using the Network Sniffer, possible additional features of the Network Sniffer compared to currently known Android Network Sniffer. The document is intended for developers and users of the system.

***1.2 Document Conventions***

This document follows the basic SRS convention methodology and also based on templates that are online in which to draft this document

***1.3 Intended Audience and Reading Suggestions***

This document is intended for developers, project managers, users, testers and documentation writers. The SRS below contains information regarding the project, scope of the project, references used in writing the SRS, testers who if possible solve any issues that the current developers may have faced, and also for users who wish to understand what the project created was about. It is suggested when reading to have knowledge with regards to Networking, Android, Android Programming, Java, C Programming, as they will provide an easier understanding.

***1.4 Project Scope***

The android network sniffer application where the application should be able to capture packets and view them. However, there may be certain limitations and constraints that may be imposed on due to the requirements which will be further explained in the later sections

Objectives are to have the ability to perform real time capturing and viewing of packet data and save the data into a file which can be viewed later, additional features such as filtering of traffic, viewing

**1.5 References**

<https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database><https://aakashtechsupportdocs.readthedocs.io/en/latest/prodpersp.html>

## 2. Overall Description

***2.1 Product Perspective***

The application is solitary consisting single mobile application. The mobile application will be used to capture and view packets, also to save packet information in text files on the mobile device. The mobile application will need to be rooted for this application to work. By default network card for mobile phone manufacturers disable the ability to change the mode of the network card. Rooting the phone allows the ability to change the mode of the network card that allows the capturing of packets. This is a network/data centric application that requires some storage space. It also utilizes text files and storage locations on the mobile device. Storage locations such as "/sdcard/Download/".

The mobile application also has some restrictions such that it has to be connected to a wifi access point to capture packets.

***2.2 Product Features***

The main features of our product includes:

* Network Sniffing: To be able to start & stop network sniffing
* To be able to provide real time viewing of sniffed data
* To be able to save the sniffed data into a file
* To be able to read from the saved file
* To be able to filter packet type captured

***2.3 User Classes and Characteristics***

The type of User classes for our product would be mostly IT users, developers.

IT Users: users that work in the field of IT such as System Administrators, who constantly have work involving Network such as Monitoring of Network Traffic for anomaly, or even regular data collection of network traffic to improve the system such as avoiding overhead. IT users would be the type of users that most frequently use Android Network Sniffers

Developers: Users that wish to develop a similar type of Application may want to user our product as reference or possibly create a more improved version of ours as the technology improves

***2.4 Operating Environment***

The device of our product choice must support Android Studio, programming language will mostly be done in Java, with the exceptions of binaries done in C using the Pcap library.

Hardware of the device must have a chipset that allows NIC promiscuous mode. Device also needs to be rooted.

***2.5 Design and Implementation Constraints***

Because of the platform in which our product is on which is Android there will be limitation and constraints compared to the Personal Computer type of Network Sniffer.

When designing our product, the constraints are as follows:

* Network Cards/Wifi Chips for mobile devices are manufactured in such a way that they are not allowed to change their mode.(Why root is needed)
* Network Interface Cards not being able to change their mode means that the alteration of the underlying firmware for the cards is needed. (Firmware allows to change card mode)
* Not all wifi chips support promiscuous/monitor mode, resulting in some phones unable to capture packets. (Only certain wifi chips support packet capture)
* The device must be rooted which is to give Super User Access to the phone. However, it is to be noted that rooting a phone would be similar to what would be known as Jail Breaking an iPhone.
* Phone being rooted and requiring the network card means only 1 person test app at a time. (Inconvenience, cannot use emulator to fully test app)
* Device being rooted results in super user privileges given to applications that should not have the permissions. This requires something to manage the SU privileges.
* The inability to give the app SU results in the need to spawn processes that have SU permission.
* The inability to give the sniff app SU also means that code cannot be used in to sniff. (must use process -> binary)
* Limited resources on mobile devices.
* Have to be coded in Android environment, Java or C.

***2.6 User Documentation***

Other than the product of an Android Network Sniffer there will also be a Technical Documentation that elaborate clearly the entire technical aspect of the product

Other than the Technical Document there will also be a User Documentation to allow ease of use for the users who will be using our application

***2.7 Assumptions and Dependencies***

Assumptions -> users are using devices that are rooted, Android Marshmellow 5.0 and above, wi-fi card must be able to support promiscuous mode.

Dependencies -> C compile Binaries

## 

## 3. System Features

Below list our system features of the product and all possible features available

***3.1 Network Sniffing Feature***

Users should be able to start and stop the network sniffing of the application. Be able to see, capture and save packet data.

3.1.1 Description and Priority

This feature is considered the core, High Priority, of our entire product where by the user is able to start and stop sniffing of network. The user should be able to capture packet data and this data should be able to be saved onto storage in the device.

3.1.2 Stimulus/Response Sequences

User starts the network sniffing and the device will display packet data onto the screen. The network sniffing can be stopped at any time.

Upon starting and subsequently pressing the stop button, stopping the network sniffing, the device will save packet data to a file in the ‘/sdcard/downloads’ folder. Once the packet data is saved, it can be viewed and further analysed on the computer using other 3rd party programs like Wireshark.

3.1.3 Functional Requirements

Start and Stop sniffing of packets. Able to save the sniffing data to a file which can be retrieved later for viewing/analysis.

***3.2 Network Sniffing Filter Feature***

Filter data captured from sniffing a network.

3.2.1 Description and Priority

Be able to filter the data from the sniffing feature, Medium Priority. Can be filtered by anything. Can be used in conjunction with the network map feature of obtaining Mac Address of a device. Filter packet data captured by the Mac Address.

3.2.2 Stimulus/Response Sequences

Filter can be set at any time. User enters the filter. The data displayed on screen is related to the filter.

3.2.3 Functional Requirements

Able to filter data captured from sniffing.

***3.3 Network Sniffing Analysis Graph Feature***

Creates and displays a graph dependent on the data captured/saved based on a .pcap file.

3.3.1 Description and Priority

Reads a .pcap file and creates a graph based on the file, Medium Priority. The file can be from the app itself or from elsewhere. Being able to create a graph showing the number of packets related to a device. The Y-axis being the number of packets and the X-axis being the device. The graph is created and displayed to the screen.

3.3.2 Stimulus/Response Sequences

The user first chooses the .pcap file to read. Based on the .pcap file, the graph will be generated onto the screen. The graph shows the number of packets in relation to a specific device.

3.3.3 Functional Requirements

Create a simple analysis related to the network sniffing data captured.

***3.4 Network Mapping Feature***

Users should be able to start the mapping of the network. Be able to see devices connected to a specific network.

3.4.1 Description and Priority

Being able to map the network, Medium Priority, showing every device that is connected to a specific network/sub-network.

3.4.2 Stimulus/Response Sequences

User enters a specific network address to map to. User then press the start network map button. The output pertains to devices connected to the network. The device data should be displayed to the screen.

3.4.3 Functional Requirements

Be able to see devices connected to a specific network. Also to check data collected against a list of authorized mac addresses.

***3.5 Network Mapping Check Mac Feature***

Allows checking of mac addresses found against a list of authorized mac addresses.

3.5.1 Description and Priority

After mapping the network, the user can check against a list of authorized mac addresses to see if there are any devices missing or unauthorized devices connected, Medium Priority.

3.5.2 Stimulus/Response Sequences

After mapping the network, all the details are displayed to the screen. Scrolling all the way to the bottom of the screen, there is a ‘Nmap Done’. Upon pressing of ‘Done’, the user will be prompted to choose the file to check against mac addresses found. A short report will be generated.

3.5.3 Functional Requirements

To check the validity of mac addresses found

***3.6 Network Mapping OS Feature***

Mapping the OS of a device in the network and display possible vulnerabilities.

3.6.1 Description and Priority

User should be able to enter an address related to a device in the network and find out what OS the device is running, Medium Priority. Also able to see vulnerabilities against that OS.

3.6.2 Stimulus/Response Sequences

User should be able to enter an address related to a device in the network. The type of OS the device is running should be displayed onto the screen. Upon pressing on the output displayed, a web browser with a list of possible vulnerabilities against the OS will be displayed.

3.6.3 Functional Requirements

Find out what OS the device is running. Also be able to see OS vulnerabilities for the device.

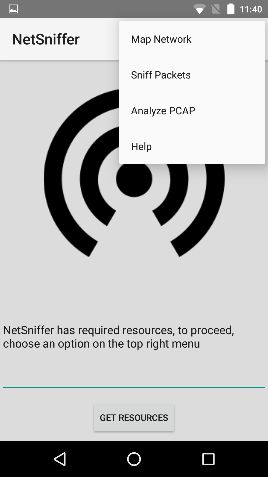
## 4. External Interface Requirements

***4.1 User Interfaces***

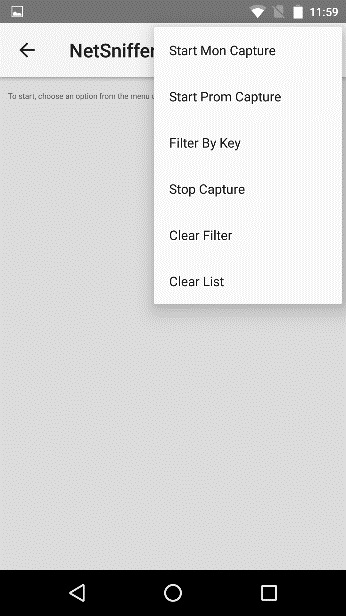
Main Screen of the Application. There are 4 options on the main screen, with the main functionality to “Sniff Packets”. The main requirement of the UI is to be simple and easy to understand while at the same time be able to carry out the main functionality of the application. UI should also be implemented in a way where user can efficiently carry out the task.

Requirements:

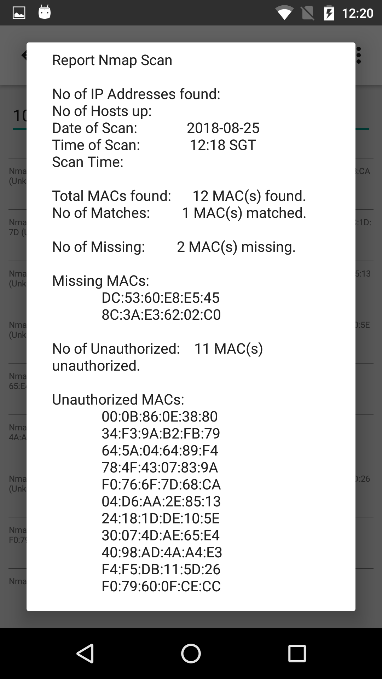
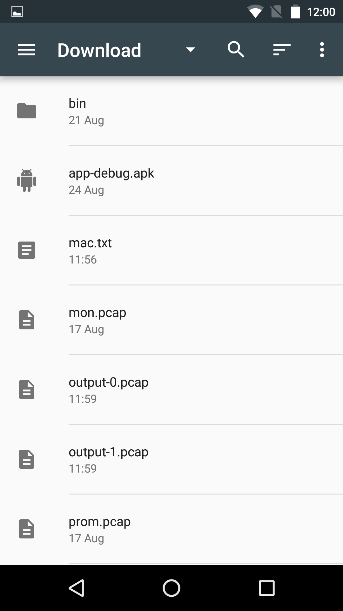
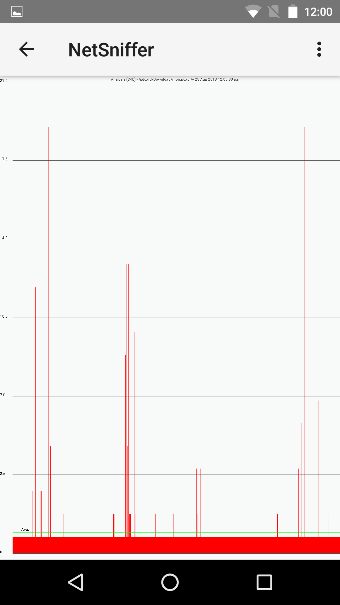
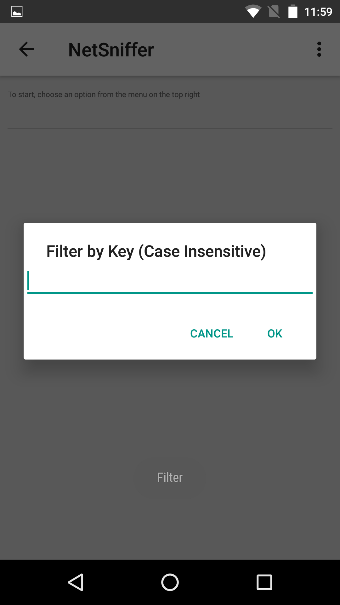
* Simple & easy to understand (Main Menu “clean” look)
* Able to carry out main functionality (Has required UI)
* List view to show each packet individually
* Show options user can choose in simple straightforward manner

* + 1. ***User start capture, display in list view***



***4.1.2 Other User Interfaces added***

***4.2 Hardware Interfaces***

Devices used must have chipset that allows Monitor mode, device also needs to have root access. After Sniffing of the network the file will be saved in a Pcap format that can only be run by applications that can read this particular format.

Example of chipset BCM 4325,4329,4330, 4335,4339.

Nexus 5 device for our project uses BCM 4339.

***4.3 Software Interfaces***

Prototype version -> TCP Binary for ARM devices

* Stored in assets
* Extracted to Internal Storage (allocated for the application)
  + /data/data/com.example.yuxuan.netsniffer/tcpdump
* Main Activity checks if exists
* Sniff Activity
  + Creates processes (threads)
  + Queries Binary and Output to a text file
  + Updates UI
* Nmap Activity
  + Creates processes
  + Binary and Output to text file
  + Update UI

## 5. Other Nonfunctional Requirements

***5.1 Performance Requirements***

Wi-Fi card Range that is device specific as different devices uses different Wi-Fi cards and even if the Wi-Fi cards are compatible they may have different ranges to do the sniffing

***5.2 Safety Requirements***

Rooting the phone is one of the requirements of the device however it is to be noted that when rooting the phone there will be some effects that the user must be aware of

Disadvantages when Rooting phone

* Rooting immediately voids your phone’s warranty
* Rooting have a risk of “bricking” the phone: Device might become dead and unusable
* Poor performance: when additional features that need to be added might cause the device to lose performance speed and features
* Viruses: foreign programs installed might utilize privilege escalation if root access is introduced

***5.3 Security Requirements***

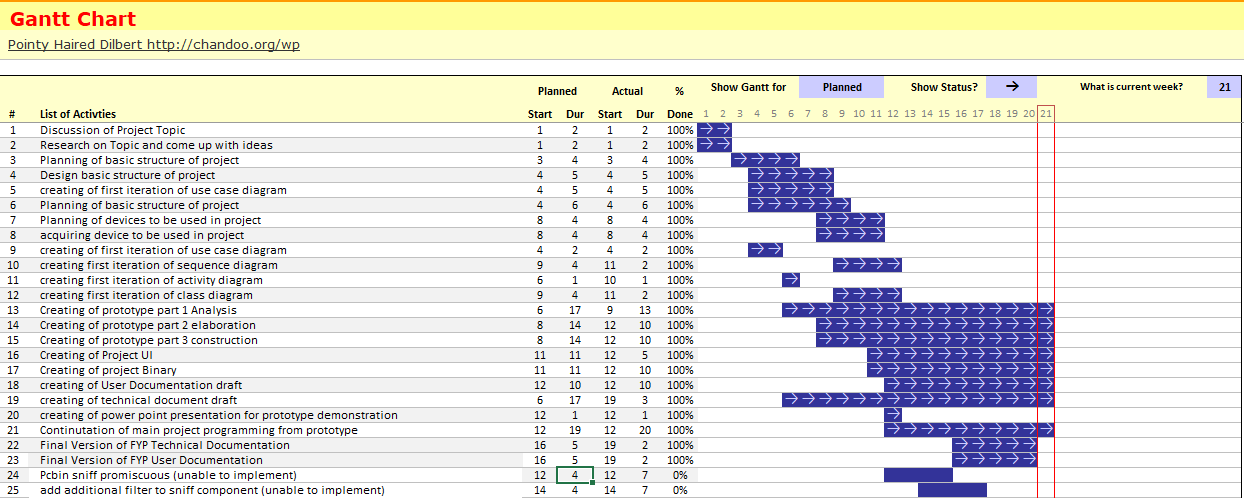
There is a security issue when using the phone in which if the device is rooted. There will be an increase in risk of viruses because rooted device allows for special permissions that allow additional customization of the device. For example, places that were not allowed to be modified can now be accessed. This causes vulnerabilities as things that shouldn’t be changed are now open.

There is now a need to manage the Super User permission on the device. “SuperSu” android application in this case was used to manage permissions.

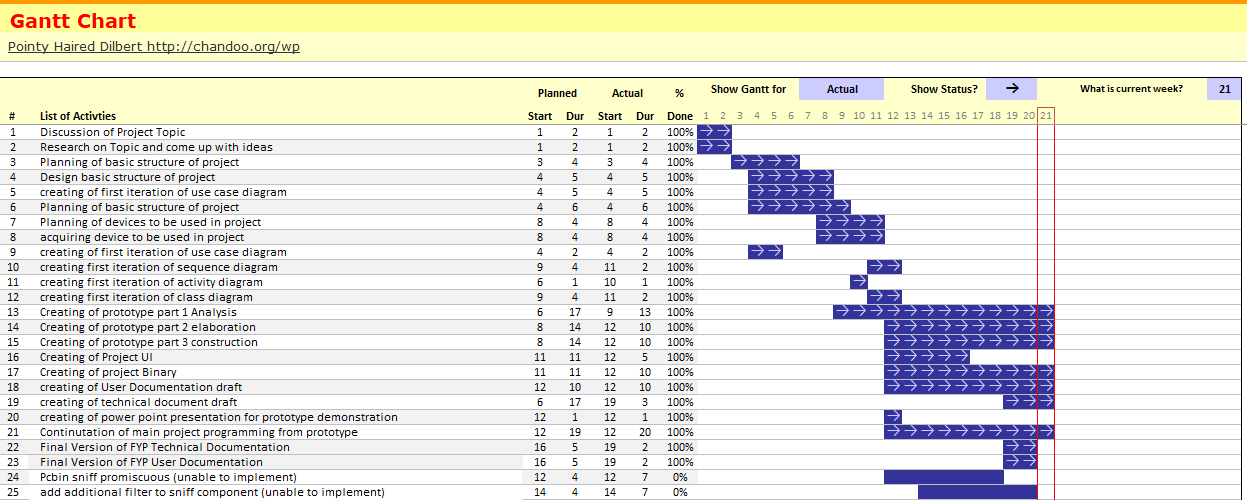
# **Gantt Chart**

Chart showing tasks planned and actually done.

## 1. Planned Work

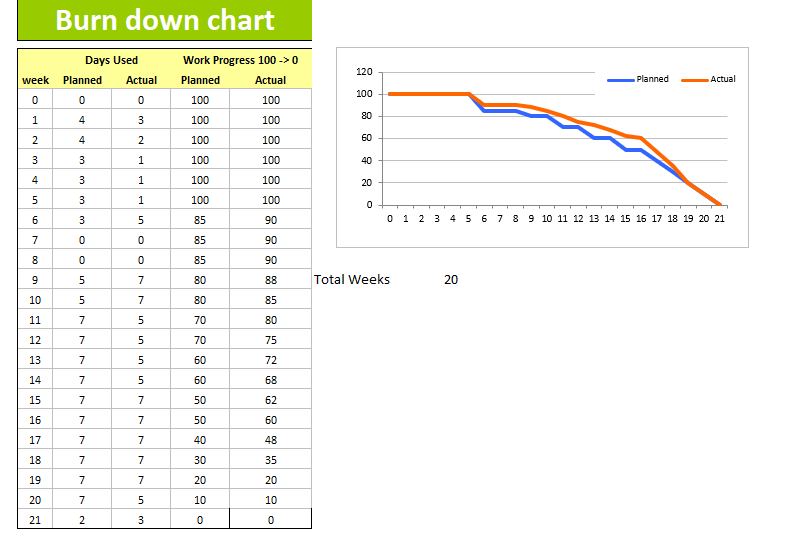


## 2. Actual Work



# **Burn Down Chart**

Actual and Planned burnt down chart according to our progress with the project. Starting from 100% down to 0% (when the project is completed). From the chart, around 6 ~ 9 weeks into the project, our actual work done stagnated due to other modules projects and exams resulting in more work pushed further back into the later weeks.



# **Sprint Summary**

**Ver 1.0, 14 April 2018**

First Day of Sprint: April 14, 2018

Last Day of Sprint: April 15, 2018

Working Days in Sprint: 2

Thoughts and Ideas on the project being discussed with all members of the group

Task: Ideas for the Requirement/Scope of the Project

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Ongoing |

Next Sprint meeting date: 16 April

**Ver 1.1, 16 April 2018**

First Day of Sprint: April 16, 2018

Last Day of Sprint: April 17, 2018

Working Days in Sprint: 2

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Ongoing |

Next Sprint Meeting Date: 18 April 2018

**Ver 1.2, 18 April 2018**

First Day of Sprint: April 18, 2018

Last Day of Sprint: April 22, 2018

Working Days in Sprint: 5

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Ongoing |
| 3 | Research Limitation and Solutions | Ongoing |
| 4 | Research regarding PCap files | Ongoing |
| 5 | Research use of Android Studio for UI | Ongoing |

Next Sprint Meeting Date: 23 April 2018

**Ver 1.3, 23 April 2018**

First Day of Sprint: April 23, 2018

Last Day of Sprint: April 23, 2018

Working Days in Sprint: 1

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Ongoing |
| 3 | Research Limitation and Solutions | Ongoing |
| 4 | Research regarding PCap files | Ongoing |
| 5 | Research use of Android Studio for UI | Ongoing |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process (RUP) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Ongoing |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |

Next Sprint meeting date: 24 April 2018

**Ver 1.4, 24 April 2018**

First Day of Sprint: April 24, 2018

Last Day of Sprint: April 29, 2018

Working Days in Sprint: 6

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Ongoing |
| 3 | Research Limitation and Solutions | Ongoing |
| 4 | Research regarding PCap files | Ongoing |
| 5 | Research use of Android Studio for UI | Ongoing |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process (RUP) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Ongoing |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Ongoing |

Next Sprint Meeting Date: 30April 2018

**Ver 2.0, 30 April 2018**

First Day of Sprint: April 30, 2018

Last Day of Sprint: May 4, 2018

Working Days in Sprint: 5

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |

Next Sprint Meeting Date: 5 May 2018

**Ver 2.1, 5 May 2018**

First Day of Sprint: May 5, 2018

Last Day of Sprint: May 11, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |

Next Sprint Meeting Date: 12 May 2018

**Ver 2.2, 12 May 2018**

First Day of Sprint: May 12, 2018

Last Day of Sprint: May 18, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Creation of Simple UI App | Ongoing |

Next Sprint Meeting Date: 19 May 2018

**Ver 2.3, 19 May 2018**

First Day of Sprint: May 19, 2018

Last Day of Sprint: May 25, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Creation of Simple UI App | Ongoing |
| 11 | Create Our Own Binary (Pcbin) | Ongoing |

Next Sprint Meeting Date: 26 May 2018

**Ver 2.4. 26 May 2018**

First Day of Sprint: May 26, 2018

Last Day of Sprint: Jun 8, 2018

Working Days in Sprint: 14

Team has exams in less than 2 weeks, not expected to do much work.

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Ongoing |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Creation of Simple UI App | Ongoing |
| 11 | Create Our Own Binary (Pcbin) | Ongoing |

Next Sprint Meeting Date: 9 Jun 2018

**Ver 2.6, 9 Jun 2018**

First Day of Sprint: Jun 9, 2018

Last Day of Sprint: Jun 15, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Ongoing |

Next Sprint Meeting Date: 16 Jun 2018

**Ver 2.7, 16 Jun 2018**

First Day of Sprint: Jun 16, 2018

Last Day of Sprint: Jun 22, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Ongoing |
| 15 | Documentation (ongoing process) | Ongoing |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Ongoing |
| 11 | Getting Processes to Work with Binary and SU | Ongoing |

Next Sprint Meeting Date: 23 Jun 2018

**Ver 2.8, 23 Jun 2018**

First Day of Sprint: Jun 23, 2018

Last Day of Sprint: Jun 29, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Ongoing |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Ongoing |

Next Sprint Meeting Date: 30 Jun 2018

**Ver 3.1, 30 Jun 2018**

First Day of Sprint: Jun 30, 2018

Last Day of Sprint: July 6, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |

Next Sprint Meeting Date: 7 July 2018

**Ver 3.2, 7 July 2018**

First Day of Sprint: July 7, 2018

Last Day of Sprint: July 13, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Ongoing |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Ongoing |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Ongoing |

Next Sprint Meeting Date: 14 July 2018

**Ver 3.3, 14 July 2018**

First Day of Sprint: July 14, 2018

Last Day of Sprint: July 20, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Ongoing |
| 22 | Integrate Pcbin(revised) with NetSniffer | Ongoing |

Next Sprint Meeting Date: 21 July 2018

**Ver 3.4, 21 July 2018**

First Day of Sprint: July 21, 2018

Last Day of Sprint: July 27, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | Ongoing |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Ongoing |
| 22 | Integrate Pcbin(revised) with NetSniffer | Ongoing |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | Ongoing |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Inject Resources into ‘System/Bin’ instead of ‘/data/data’ | Ongoing |

Next Sprint Meeting Date: 28 July 2018

**Ver 3.5, 28 July 2018**

First Day of Sprint: July 28, 2018

Last Day of Sprint: Aug 3, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | N |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Ongoing |
| 22 | Integrate Pcbin(revised) with NetSniffer | Ongoing |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | Ongoing |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Inject Resources into ‘System/Bin’ instead of ‘/data/data’ | Ongoing |

Next Sprint Meeting Date: 4 Aug 2018

**Ver 3.6, 4 Aug 2018**

First Day of Sprint: Aug 4, 2018

Last Day of Sprint: Aug 10 , 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | N |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Y |
| 22 | Integrate Pcbin(revised) with NetSniffer | Ongoing |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | Ongoing |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Injected Resources into ‘System/Bin’ instead of ‘/data/data’ | Ongoing |
| 29 | Solve Nmap Crash on “Running” | Ongoing |

Next Sprint Meeting Date: 11 Aug 2018

**Ver 3.7, 11 Aug 2018**

First Day of Sprint: Aug 11, 2018

Last Day of Sprint: Aug 17, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | N |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Y |
| 22 | Integrate Pcbin(revised) with NetSniffer | Y |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | Ongoing |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Injected Resources into ‘System/Bin’ instead of ‘/data/data’ | Y |
| 29 | Solve Nmap Crash on “Running” | Y |
| 30 | Solve Injected Binaries not Executing | Y |

Next Sprint Meeting Date: 18 Aug 2018

**Ver 3.8, 18 Aug 2018**

First Day of Sprint: Aug 18, 2018

Last Day of Sprint: Aug 24, 2018

Working Days in Sprint: 7

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | N |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |
| 20 | Do up Documentation for Final Submission | Ongoing |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | Ongoing |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Y |
| 22 | Integrate Pcbin(revised) with NetSniffer | Y |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | N |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Injected Resources into ‘System/Bin’ instead of ‘/data/data’ | Y |
| 29 | Solve Nmap Crash on “Running” | Y |
| 30 | Solve Injected Binaries not Executing | Y |
| 31 | Do Simple File Browser | Y |

Next Sprint Meeting Date: 25 Aug 2018

**Ver 3.9, 25 Aug 2018 (Last Sprint)**

First Day of Sprint: Aug 25, 2018

Last Day of Sprint: Aug 31, 2018

Working Days in Sprint: 6

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | Come up with ideas on the requirement/Scope of the Project | Y |
| 2 | Discussion on Scope of Project | Y |
| 3 | Research Limitation and Solutions | Y |
| 4 | Research regarding PCap files | Y |
| 5 | Research use of Android Studio for UI | Y |
| 6 | Delegate Roles for Each Team Member | Y |
| 7 | Select Design Tools (Umlet) | Y |
| 8 | Select a Design Process ~~(RUP)~~ (Agile) | Y |
| 9 | Create Draft Proposal by Saturday (28 April 2018) | Y |
| 10 | Program/Document Version Control (GitHub/WordPress) | Y |
| 11 | Device to be Used | Y |
| 12 | Possible Test Cases for Project Discussion | N |
| 13 | Create a Structure of Program Design and Basic Things | Y |
| 14 | Discuss on Diagrams Required | Y |
| 15 | Documentation (ongoing process) | Y |
| 16 | Creation of Simple App UI | Y |
| 17 | Get App Working with Simple UI & Tcpdump | Y |
| 18 | Integrate Pcbin with Simple UI App | Y |
| 19 | Get Documentation Done for July Midpoint | Y |
| 20 | Do up Documentation for Final Submission | Y |
| 21 | Do up 1-min video and demo | Y |

Programming Task List

|  |  |  |
| --- | --- | --- |
| Task | Description of Task | Complete (Y/N) |
| 1 | GUI – Add Menu | Y |
| 2 | GUI – Switching Pages of the UI | Y |
| 3 | GUI – Reading on VCS | Y |
| 4 | GUI – Logging for Administrative use | Y |
| 5 | GUI – Pop ups (Toast) | Y |
| 6 | GUI – TextViews Scrolling | Y |
| 7 | GUI – Prevent Clicking on TextViews | Y |
| 8 | Cross-Compilation of TCPDump Binary for Android | Y |
| 9 | Rooting Nexus 5 with CF-Auto-Root | Y |
| 10 | Create Our Own Binary (Pcbin) | Y |
| 11 | Getting Processes to Work with Binary and SU | Y |
| 12 | Device need SU | Y |
| 13 | Solve Processes Not Killed After Stop | Y |
| 14 | Get Pcbin to Sniff ARP | Y |
| 15 | Pcbin (only sniff itself), try to change NIC setting | N |
| 16 | Compile Pcbin binary for ARM architecture | Y |
| 17 | Solve Toast Output Not Running on UI | Y |
| 18 | Implement Nmap binary to NetSniffer | Y |
| 19 | GUI – Replace TextView with ListView | Y |
| 20 | Add Nmap Services for Nmap binary to run | Y |
| 21 | Add Nmap ListView onClick Feature | Y |
| 22 | Integrate Pcbin(revised) with NetSniffer | Y |
| 23 | Solve Pcbin binary DNS Structure Inconsistency | Y |
| 24 | Solve Pcbin, Read Resource Record from DNS Header | Y |
| 25 | Optimize Info Output for Nmap OS (too much info) | Y |
| 26 | Additional UI Components Sniff Add Filters | N |
| 27 | Solve NetSniffer does not sniff ‘Promiscuous Mode’ | N |
| 28 | Injected Resources into ‘System/Bin’ instead of ‘/data/data’ | Y |
| 29 | Solve Nmap Crash on “Running” | Y |
| 30 | Solve Injected Binaries not Executing | Y |
| 31 | Do Simple File Browser | Y |
| 32 | Implement File Browser with Nmap OS (File of List of Mac Address) | Y |
| 33 | Troubleshoot Sniff Graph & File Browser(Choose .pcap file) | Y |
| 34 | Implement Pcmon into NetSniffer | Y |

# **Technical Information**

## 1. Document Overview

The document describes the design process which is used in creating the application, it also includes UML Diagrams to illustrate the design of the system that can be used to show the functionalities of the application. Some form of implementation may also be described. Other than functionalities there will also be test cases to show that the application have the appropriate quality for use.

## 2. Introduction

Computers used to be large clunky machines which were the innovation of the IT sector but now, almost everyone if not all, has a “small computer” with them all the time, a smartphone. One of the duties of a network administrator is to monitor the network. This can be done with a computer with either with an external network card or with an internal one. However, the computer was something that was needed, adding weight and hassle. What if one could monitor and sniff packets from their device whom they carried with them everywhere? One could simply be in a location, capturing packets in their vicinity for analysis, without having to lug around a laptop right into their mobile device.

***2.1 What is Packet Sniffing?***

“A sniffer (packet sniffer) is a tool that intercepts data flowing in a network. If computers are connected to a local area network that is not filtered or switched, the traffic can be broadcast to all computers contained in the same segment. This doesn’t generally occur, since computers are generally told to ignore all the comings and goings of traffic from other computers. However, in the case of the sniffer, all traffic is captured when the sniffer software commands the Network Interface Card (NIC) to stop ignoring the traffic. The NIC is put either into promiscuous mode or monitor mode, and it reads communications between computers within a particular segment. This allows the sniffer to seize everything that is flowing in the network, which can lead to the unauthorized access of sensitive data. A packet sniffer can take the form of either a hardware or software solution. The difference between promiscuous mode and monitor mode will be elaborated further below.

***2.2 What is TCPDump?***

TCPDump is a free software designed by Van Jacobson, Sally Floyd,Vern Paxson and Steven McCanne in 1988. It uses the libpcap library in C to read packets from a network interface card. TCPDump prints the contents of network packets, it can also read the packets from a network interface card (NIC) or from a saved packet file. It can also write packets to standard output or to a file. For a more specific purpose we can use it to intercept and display communications of another user, with the necessary privileges of a system acting as a router or gateway through which those unencrypted traffic like telnet or HTPP passes we can use TCPDump to view login IDs, passwords, URLs and also the content of which the websites that were being viewed or even any other unencrypted information. When using TCPDump on a network with high volumes of traffic filters can be used like BPF-based to limit the number of packets that will be seen by TCPDump, which makes the output more usable. However, in order for TCPDump to be used, superuser privileges is required as packet capturing mechanisms on those systems requires the privileges. Which is an essential part of our project whereby we need a rooted phone as well as superuser privilegs.

***2.2.1 How to setup TCPDump in Linux Environment***

wget http://www.tcpdump.org/release/libpcap-1.8.1.tar.gz

tar -xf libpcap-1.8.1.tar.gz

sudo apt-get update

sudo apt-get install flex

sudo apt-get install bison

cd libpcap-1.8.1

./configure –prefix=/usr

Make

Sudo make install

Following the above step will compiled the libpcap giving libpcap.a and libpcap.so file.

The .a file are static libraries while .so are dynamic libraries. The difference between the two is, in a static libraries, if you used code stored inside them, it is taken from them and embedded into your own binary while in dynamic libraries, the code is not taken and embedded into your own library. It is referenced so the binary will depend on them and the code from the so file is loaded at runtime.

***2.3 What is libpcap?***

libpcap is the abbreviation of packet capture library which is an application programming interface (API ) that is used to capture network traffic. The libpcap is written in c so therefore in order for android which uses JAVA, we must use a wrapper to translate a library existing interface to the compatible interface. The wrapper enables android which uses JAVA to use libpcap’s C code functions.

Example of some wrapper program readily available would be jPcap,jNetPcap,JPcap and pcap4j. These are all wrapper libraries made by other developers to enable C code to be use in JAVA.

***2.3.1 Creating our own Binary like TCPDump***

We planned to achieve this by using libpcap (API) and write the general code which is similar to TCPDump. After that, we cross compiled it into an ARM compatible structure and use the ARM binary in our applications. Why do we use ARM? As most mobile uses ARM processors instead of intel processors. The difference between linux ARM and linux x86 is that internally their binary code is different thus software done on x86 must be pre compile to match the target architecture.

***2.3.2 Difficulties in creating the Binary***

Although these libraries are readily available for us to use in android we face one problem.

One of the problems we faced was that if we want to sniff packets in promiscuous/monitor mode, we would need root permission. Getting root permission would not be a problem if we were to work in environments such as LINUX or WINDOWS on the computer.

In android any access to protected resources or services is guarded by the application permission framework, all access in native code needs to be analyzed, and the required permissions should be identified. Whatever permissions the native code may need should be published for developers, so that they can include these permissions in their applications Manifest file. Native code should not rely on code that need access as this is not available on standard android. [4]

The above sentence is a research that we did when we are looking for ways to use wrapper class to enable us to use c code that requires root access on android. As stated in the first line o, “in android any access to protected resources or service is guarded by the application permission framework “this implies that our C code would need to get past the application permission framework. From our research we are unable to find or come up with any way to work around it. Therefore we went to look for another alternative which is to use an ARM binary library for ARM architecture. If we were to use a binary in android we can give root to process and therefore bypassing the no rooting to native code problem.

The next problem would be that even though we created our own ARM binary, the binary should have no problem executing the code in the NEXUS 5 device. However, when we tried to execute the binary, it was denied permission even though the phone is rooted and ran in a rooted process. On further research in to the matter, we found out that it was because the compilation method was wrong .The previous ARM binary was compiled using a dynamic linked to the library. Once we changed the binary to using a static link to the library, the binary was able to run with no problems at all.

From there on we started the development of pcmon which is a binary used to enable android device with the supported NIC to sniff in monitor mode. However the binary which was compiled using the same method which allowed pcbin to be used in ARM architecture failed again. However this was not solved and no solution was found as this problem disappeared as soon as the phone was tested on a different network from SIM environment and back to SIM environment.

Another problem faced during the development of pcmon was the lack of knowledge of 802.11 protocol. We didn’t know that for all transmission of wifi packets in a wifi environment, the 802.11 will use a 802.11 radiotap header. Therefore we would need to fully understand 802.11 radiotap before we can dissect the 802.11 packet to get the selected information out. 802.11 will be elaborated further down.

***2.4 IP Packet Header***

**[5]**

IP header usually about 20 bytes. The source address and destination address mean from where the IP packet comes from and where the packet is trying to reach.

The identification field is used to uniquely identify the group of fragments of a single IP datagram.

The Types of service has now been redefined by RFC 2474 for Differentiated service and Explicit Congestion Notification (ECN). Differentiated service (DSCP) is used when technologies require real time data streaming is needed. ECN is defined in RFC3186 and allows end to end notification of network congestion without dropping packets. It is an optional feature which is only effective if there is underlying network.

***2.5 Type of IP Packets***

1. TCP/IP
2. UDP
3. ICMP (INTERNET CONTROL MESSAGE PROTOCOL)
4. IGMP (INTERNET GROUP MANAGEMENT PROTOCOL)
5. IGRP ( INTERIOR GATEWAY ROUTING PROTOCOL)
6. ESP (ENCAPSULATING SECURITY PAYLOAD)
7. AH (AUTHENTICATION HEADER)
8. DNS (DOMAIN NAME SYSTEM)

***2.5.1 Transport Control Protocol (TCP)***

[2]

TCP header usually requires a minimum of 20 bytes anything.

As shown above the source port takes up around 2 bytes, destination port takes up around 2 bytes,

Sequence number takes up 4 bytes. Sequence number is used to mark the ordering of a group of message. Acknowledgment number takes up 4 bytes are used by the sender and receiver to communicate the sequence number of messages.

Data offset field takes up 4 bits and store the total size of a TCP header in multiples of four bytes. A header using the optional TCP field has a data offset of 5, while a header using the maximum-sized optional field has a data offset of 15.

Reserved uses 3 bits and the value is always zero.

Control flags uses 9 bits and TCP use 6 standard and 3 extend control flags to manage data flow in specific situations. Each flag is 1 bit in size. The flags are SYN, ACK, FIN, URG, PSH, RST, ECE, CWR, NS flags.

SYN aka synchronous flag is used as a first step in establishing a 3 way handshake between 2 hosts.

ACK aka acknowledgment is used to acknowledge the successful receive of a packet. It is use to tell the sender it has receive the initial packet.

FIN aka as Finished. Tell the receiver that sender has no more data to send. Therefore this is the last packet sent to receiver.

URG aka urgent. This flag is to the receiver to process the urgent packet first before processing other packet.

PSH stands for push. It is similar to URG flag. It is used to tell the receiver to process the packets with PSH flag instead of buffering them.

RST stands for reset flag. This flag is sent when a packet is sent to a particular host that was no expecting it.

ECE is responsible for indicating if a TCP peer is ECN capable

CWR flag stands for Congestion window reduced is used by sending host to indicate it received a packet with ECE flag set.

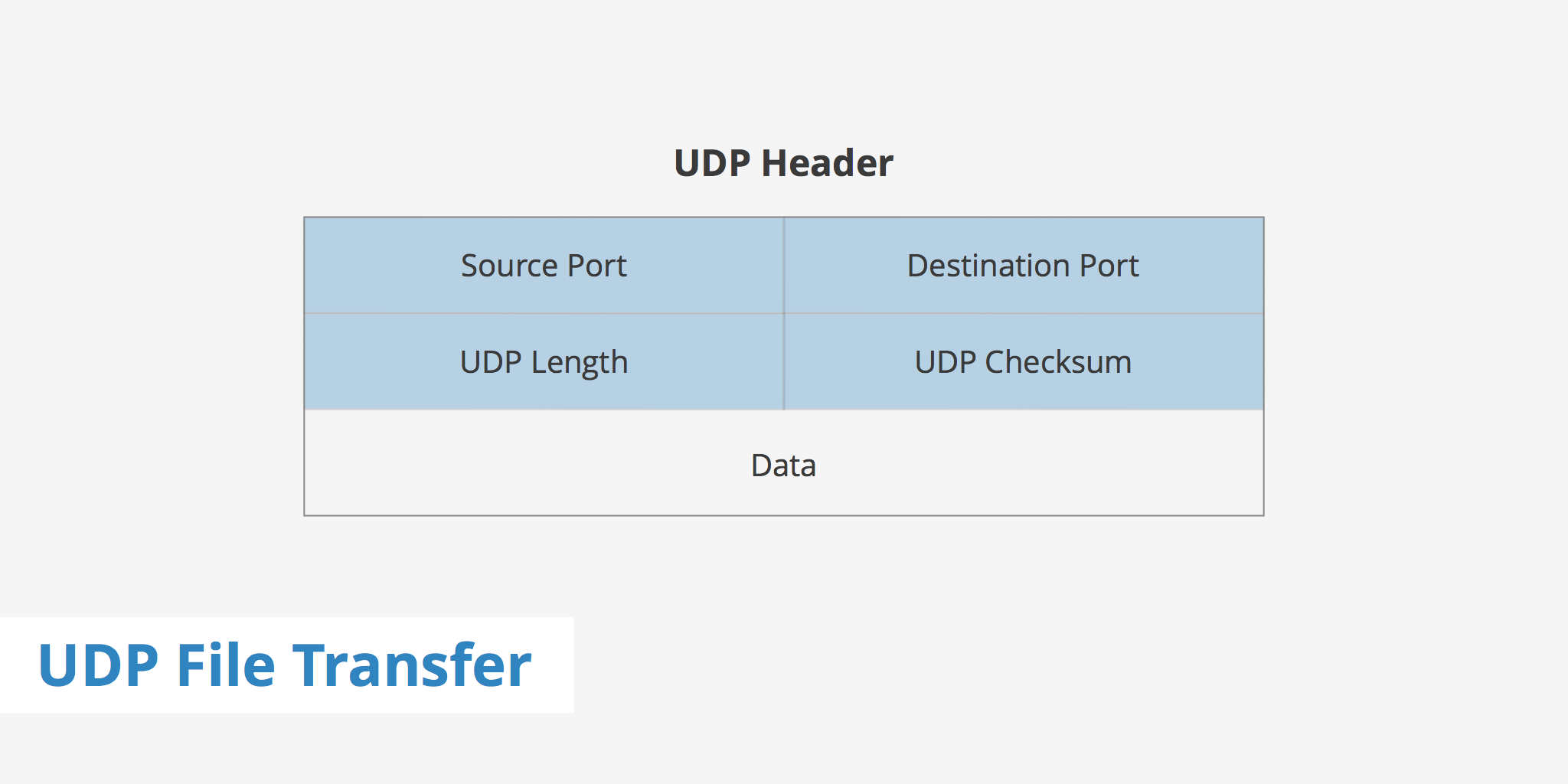
NS stands for Nonce Sum, it is an experimental flag used to help protect accidental malicious concealment of packets from sender.

Window size is a value to regulate how much data is send before an acknowledgment from receiver is required. This value cannot be too small or too big as being too small will slow performance while too big will cause network link to be saturated making it unusable for other application

Checksum is used to help receiver check if the data receive has been corrupted or tempered with.

Urgent pointer field is often set to zero or ignored. [3]

***2.5.2 User Datagram Protocol (UDP)***



[1]

UDP header is much smaller than TCP. It consist of 8 bytes in total.

As shown above it only has four fields. Source port, Destination Port, UDP length and UDP checksum.

UDP length is used to identify the length of the header itself.

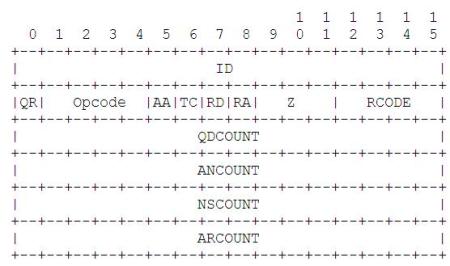
UDP checksum is the same as TCP checksum. It is used to check if a UDP packet data is corrupted or been tampered with.

***2.5.3 Domain Name System (DNS)***

What is DNS? DNS is the name for a web browser when a user wishes to visit that particular website as people are better at remembering domain names than remembering the IP Address of the website. Example of a domain name is [www.google.com](http://www.google.com) . The user internet service provider view the DNS associated with the domain name and translate it into an IP address which the machine will use to direct internet connection to the correct website.

DNS packet are a type of IP packets and they are by default always going through port 53 be it a request or a response packet. DNS server will usually reject DNS packet that does not originate from port 53.

***2.5.3.1 Domain Name System Header***



https://gieseanw.wordpress.com/2010/03/25/building-a-dns-resolver/

ID is a 16 bit unique identifier found in all DNS packet which will help differentiate between all the other dns packet in a large volume of name service traffic

QR is a 1 bit operation field which specify whether the DNS packet is a QUERY or a RESPONSE. The QR is set to 0 for QUERY and 1 for RESPONSE.

Opcode is a 4 bit field that specify whether the DNS packet is a Query,Inverse query , Status , notify or update. 0 is for standard query 1 is for inverse query, 2 is for status,3 is for notify and lastly 4 is for update.

AA is a 1 bit authoritative answer bit to check does that name server is it an authority for the domain.

TC is a 1 bit truncation bit which notifies if the packet has been truncated due to the size restriction that was imposed on the transport layers.

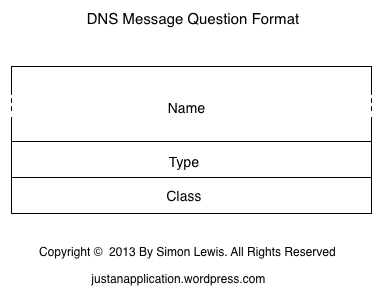
RD is a 1 bit recursion desired bit which tells the name server to pursue the operation even if it cannot immediately return a reply. This mean that it is telling the name server to fetch an answer from another name server if the current name server does not have the answer.

RA is a 1 bit recursion available bit which is set by the server to tell whether the desired recursion is available or not.

Z is a 3 bit reserved bit, currently has no purpose and is set to 000.

RCODE is a response code of 4 bit length. This is set by the server when responding to a DNS query. There are 0-5 possible outcome of RCODE. 0 signifies no error, 1 is format error caused by a flaw in parsing the query, 2 signifies a server failure, 3 is an error that returns when the referenced domain name does not exist, 4 is sent as a response to a query type that is not supported.

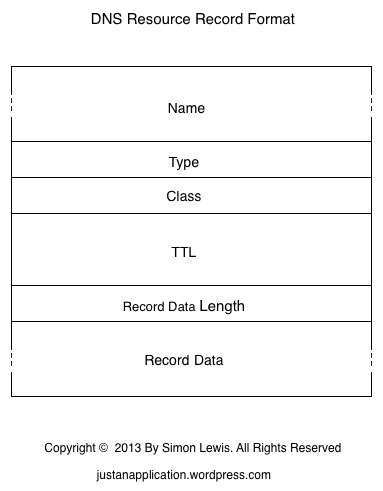
QNCOUNT,ANCOUNT,NSCOUNT,ARCOUNT is a 16 bit /2byte field that contains the number of question/answer/authority/additional record in a DNS packet.



https://justanapplication.wordpress.com/category/dns/dns-messages/dns-message-format/dns-message-header-format/

QUERIES is a field, this is present in both DNS responses and queries .It contains 3 variables , Name which is of size N byte and it contains question that are sent by a client to a name server. Type which is of 16 bits size. It contains the type of the question for eg: IPV4 or IPV6 and lastly the Class variable which is a 16 bit field that contain the class of the requested queries.

This concludes the DNS query section. DNS response come with additional fields behind. The additional fields are as follows.



ANSWER is a field that only exist in the DNS response and it contains answer to the question sent back from name server.

AUTHORITY is a field is the same as ANSWER in the sense that it only exist in DNS response and it contains name servers which are authorities to the data.

ADDITIONAL is a field. It exist only in DNS response. It contains addition record that are returned alongside the answer section.

Each of the replies follow the same DNS resource record format as shown in the picture above. The NAME is of size N and it contains the replies to the queries from the nameserver.

TYPE is a 2 byte field that specify what type of answer the DNS nameserver replied with. There are many different type that a name server could reply with. However we would only be discussing the 3 type that was specified in our pcbin binary which is IPV4, IPV6 and CNAME. IPV4 and IPV6 is pretty much self-explanatory thus we would discuss what CNAME is. CNAME is the canonical name for an alias. For example, usually when u entered google.com it would bring you to the same website as if you entered ww.google.com. This is the functionality of CNAME.

CLASS is a 16 bit field which contain the class of the requested queries.

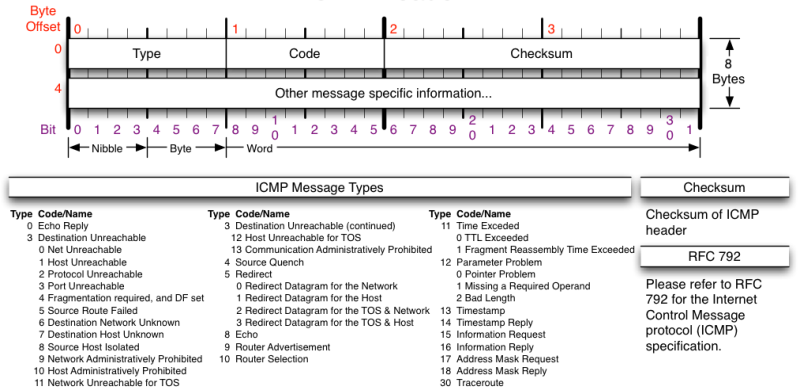
TTL is a 32 bit field which mean Time to live. This denotes how long a resource record will be valid.

RECORD DATA LENGTH is a 16 bit field which contains the length in byte of the data that is present in ANSWER.

RECORD DATA is an N byte field. This usually contains the IP address that the machine can use to direct the user to the request address as machine cannot use DNS purely to reach the requested website across the internet. The IP Address is usually in IPV4 or IPV6

***2.5.4 Other Packet Header***

**ICMP (INTERNET CONTROL MESSAGE PROTOCOL)**

**[5]**

Currently our program is unable to sniff ICMP packets due insufficient time. However, our finished application should be able to ICMP packet would be sniff with our binary, PCBIN.

ICMP is not a transport protocol that sends data between systems. It is used to troubleshoot the internet connections by network admin.

**IGMP (INTERNET GROUP MANAGEMENT PROTOCOL)**

Used by host and adjacent router on IPV4 network to establish multicast group membership.

**IGRP (INTERIOR GATEWAY ROUTING PROTOCOL)**

Distance vector interior gateway protocol. Used by routers to exchange routing data within an autonomous system.

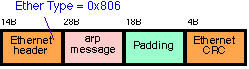
**ESP (ENCAPSULATING SECURITY PAYLOAD)**

Provides data confidentiality and authentication. Authentication mechanism authenticates only IP datagram portions of the IP packet.

**AH (AUTHENTICATION HEADER)**

Provides a mechanism for authentication only. It authenticates IP header and their payload.

***2.6 ARP (Address Resolution Packet)***

**[6]**

ARP is a protocol used by Internet protocol (IP) which broadcast packet to map IP network address to MAC Address. It is used by data link protocol. The ARP request message is sent by using the ethernet broadcast address to all system. Only the person who has the request IP address will reply. The rest will silently discard the packet.

## 3. 802.11 Packet

***3.1 802.11 Standard***

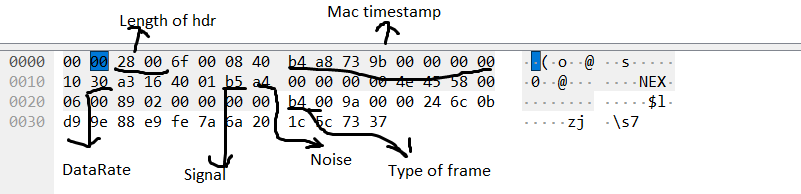
802.11 refers to a family of specification developed by the IEEE for WLAN technology. This specifies over the air interface between the wireless client and a base station .The different wireless networking specification of 802.11 are. 802.11, 802.11a, 802.11b, 802.11e, 802.11g, 802.11n, 802.11ac, 802.11ad, 802.11ah, 802.11r, 802.1X. They all applies to wireless lan but they differ in terms of speed, transmission range and frequency used. All wireless standards can use the same security protocol and either use infrastructure or ad hoc network design.

***3.2 Radiotap Header***

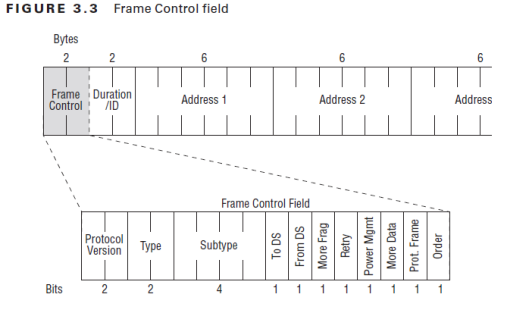
The radiotap header format is a mechanism that is used for all transmission of 802.11 packets. It provides additional information about frames from the driver to user application. It contains the version, the entire length of the radiotap header and a bitmask of the radiotap data field that follow the radiotap header. Based on the length on the raidotap header different information could be found inside. For example, the mac timestamp, channel frequency, data rate, antenna noise and antenna signal. There are still a lot of other information that can be found inside however we would not go through all of them as they are not relevant to the capabilities of pcmon. Our monitor mode sniffing binary.

The length of the radiotap header is different for all drivers as some wireless network interface do no make the additional information available. The length of the radiotap header that nexus 5 network interface provides is 40. On further research, an Intel coporation Centrino ultimate\_N 6300 interface card show that the radiotap header is of length 18 and is missing information like noise and timestamp.

***3.2.1 Radiotap Header Raw Data***



***3.3 Type of Frame in a 802.11 packet***

[7]

Frame control is a 16 bit field which contain information of the type and subtype of a packet. Inside frame control there are 11 fields which are present in all type of frames.

Protocol version. This is a 2 bit field that is used to indicate which version of 802.11 is being used however it is by default set to zero as there Is only one version of 802.11.

Type. This is a 2 bit field that is used to define which frames the 802.11 packet is using. The 3 type of frames are control, management and data frame which will be elaborated further below. 00 means the packet is using management frame, 01 for control frame, 10 for data frame, 11 is reserved.

Subtype is a 4-bit field. In 802.11 there exist a lot of subtype for the 3 main frames and the subtype field differentiate the packet subtypes which will be elaborated further below as well.

To DS and From Ds are both a 1-bit field that work hand in hand that defines whether a packet is going from distribution system to AP / client station or vice versa. There are 4 different possibilities which is 00, 01, 10 and 11. 00 means that it does not go to the Distribution system as it is a station to station communication .It could imply that it is a control or management frame. 01 just means that the frame that was just captured is coming from distribution system to client. 10 mean the opposite where the frame is going to distribution system from client. 11 usually means it’s a data frame which uses the four address format. This usually occur when wireless distribution system is in use where an AP sends a frame to another AP. It exit the DS and destined to the DS at the same time.

More fragment is a 1 bit field and if this bit is set to 1, this means that the frame have another fragment to follow.

Retry is a 1 bit field and if set to 1 in either management or data frame it mean that the frame is being sent is a retransmission as the distribution system did not receive an ACK for a unicast frame.

***3.3.1 Control Frames***

Control frame assist with the delivery of data frames. They are used to clear the channel and provide the unicast frame acknowledgement. The following information are a list of control frame subtype as defined by 802.11 standard.

Request to send is a mechanism that helps prevent collision form occurring. Thus, every station that wishes to send a frame must perform a request to send or clear to send. The duration in the request to send specify the time needed for the next frame to transmit.

Clear to send. This mechanism help to prevent collision and work in pair with request to send. The station sends the RTS and the AP will send a CTS control frame.

Acknowledge frame. Every frame that is received properly is responded with an acknowledgement frame. This allow the station to know that the frame transfer was successful. If no acknowledgement was receive this will trigger the original transmitter to retransmit its frame.

Block acknowledge request frame. This improve efficiency by combining several acknowledgements into one single acknowledgement frame.

Block acknowledge frame. This is used to acknowledge a block of QOS data frame instead of acknowledging each unicast frame one by one.

PS Poll. When the client is in Power save mode, the station will tell the access point that it using power saver by changing value of power management to 1. At this point the station will be in one of two states. Awake or doze. When the station is in awake state, the client can receive frames and transmit frame. When the station is in doze state, client station can receive or transmit any frames.

CF-End is used to indicate the end of a contention free period. A contention free period is used in wireless networks supporting the IEEe802.11 standard and is defined as a period of time during which access to the Wireless medium is free of contention.

CF-End +CF-Ack frame is used to indicate the end of a contention free period and acknowledge receipt of a frame.

***3.3.2 Management Frames***

Management Frames are used by mobile station to establish and maintain communication. They have many subtypes as well. Association request (0000), association response(0001), reassociation request(0010), reassociation response(0011), probe request(0100),probe response(0101),beacon(1000), announcement traffic indication message(1001) , disassociation(1010) ,authentication (1011), deauthentication(1100) .

An association request is a request sent to an access point from the NIC to begin an association process. Once the access point receive the request and acknowledge the reception of the request, the AP will then verify the request frame and check if the variable inside the field of the request frame matches with it own parameters. If there is a mismatch there an association response frame with the status code =1 be sent to the client. If there is not a mismatch it will grant access to the client by sending an association response frame indicating its own parameters.

A reassociation request frame is sent by a STA to an AP when it is already associated to the local area network but it wishes to associate with another AP connecting to the same local area network. It can also be used when the STA left the network for a short while and wishes to reconnect to the network/AP.

A reassociation response is what the AP will send to the STA once it verify the request with its own parameter and if there no mismatch the reassociation response will have a status code =0 which implies a successful reassociation.

Probe request is used to efficiently discover network by going through each channel and sending out probe request on that channel. It is sent to the broadcast address and and timer will be started once a probe request is send. If the timer end and no answer is received it will move on to the next channel. A SSID may be specified in the probe request. Probe response is just a response to the client.

A beacon frame are used by access point to periodically broadcast out to any listening device that the SSID is available contain information about it capabilities.

Disassociation frame is a frame that can be sent by the AP and STA when either side wishes to terminate the association.

Deauthentication is send when all communication are terminated. A disassociated STA can still be authenticated.

***3.3.3 Data Frames***

Most Data frames carry actual data that is passed down from higher level protocols.

There are 15 different subtype of data frames. However we will only be going through 3 types of subtype that pcmon has implemented. The QOS data, null and qos null. As we explained above, the subtype field is used to differentiate the packet between different subtypes of any given frames. For QOS DATA, the subtype 4 bit field would be 1000, NULL would be 0100, QOS NULL would be 1100.

Null data frames are widely used for control purposes such as keeping association alive, channel scanning and management of power.

QOS data is the QOS version of the data frame containing data.

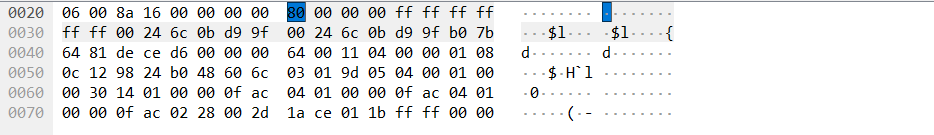
QOS null is the QOS version of null frame.

***3.4 How did we design pcmon to dissect the 802.11 packet***

The way how pcmon and pcbin sniff packet is almost similar except for the fact the pcbin set promiscuous to any none zero value while pcmon doesn’t set it. There is a function in libpcap that set\_rfmon which allow the device to be set in to monitor mode however we didn’t use that function as it android NIC is a hard mac and if we tried to change it using our binary . The application will clash. Luckily, if the device is in monitor mode and we use our pcmon to sniff the packet as per normal, it will capture radiotap header .

struct radiotap\_hdr { uint8\_t it\_version; uint8\_t it\_pad; uint16\_t it\_len; uint32\_t it\_present;

} \_\_attribute\_\_ ((packed));

The packet is cast in to a radiotap header struct. Uint16\_t it\_len is the length of the radio tap header. We needed to know the length of the radiotap header so that we can create a offset variable and use a pointer variable to point to the 802.11 packet mac header.

For nexus 5 the offset for the radiotap header is length 40 . The pointer will jump 40 time to reach the first hexadecimal value of 80. 80 in binary is 1000 0000. 1000 is subtype 8 and 00 is of management frame as stated above. Therefore this 80 means that this packet is a management frame with a subtype of 8 . This packet is the beacon frame.

The next 2 hexadecimal value is describing the flag and duration of a packet. The next 6 value ff:ff:ff:ff:ff:ff is the destination address. As this is a beacon frame its destination is to the broadcast address. Next would be the source address and last 6 would be the BSSID. Most management frame will follow the same format. Control frame and Data frame are different from the management frame in the sense that although it still describe the address at the same location as management frame, but that address might not be the destination address.

## 4. How we implemented libpcap for PCBIN/PCMON

***4.1 Main Functions of libpcap Library***

1. pcap\_lookupdev

* Finds the default device on which to capture the packet.
* Pcap\_lookupdev returns a pointer to a string giving the name of a network device suitable for use with other main function to create a packet sniffer.
* However in our program we would like to use our own interface that we specified therefore this function is not needed.

1. Pcap\_create(device, errbuff)

* Used to create a packet capture handle to look at the packet on the network

1. Pcap\_set\_promisc(handler,int )

* To set promiscuous mode on a non-activated handle. If the int is any non-zero value, promiscuous mode will be set otherwise it will not be set.

1. pcap\_activate(handler)

* Using pcap\_create just creates the handler in order to use the handler , we would need to activate it

1. pcap\_datalink

* Get the link layer header type

1. pcap\_lookupnet ()

* Get the netmask and network number of IPv4 for a device

1. pcap\_compile()

* Used to compile the string str in to filter program.Program is a pointer to a bpf\_program struct .optimize controls whether optimization on the resulting code is performed.
* Return 0 on success and -1 on failure.

1. pcap\_setfilter()

* Used to specify a filter program which is apointer to a bpf\_program struct ,usually the result of a call to pcap\_compile()
* Return 0 on success -1 on failure.

1. pcap\_loop()

* Loop mean that if will keep sniffing packet until it is specified to stop.
* It loops a specified function that is placed on the third argument

1. pcap\_close()

* Closes the handler once it has completed its job.

***4.2 Future Updates***

- combining pcbin and pcmon in to one single binary.

## 5. Application Description

This section will give an overview of the application design, development, and Implementation

***5.1 Introduction***

Our Application is a development of a network sniffer on android platform. Although it might have similarities to that of a Computer version, the user needs to know that although the application might have the same functionalities as that of the Computer Version, there are a lot more limitations that will be faced on the Android platform version.

***5.2 Operational Examples***

The application would be used mostly by network administrators, students who take Computer Science (Security) courses as the knowledge required to use the application might be different from normal users.

How a network administrator can use this application for example is monitor day to day network traffic to check for anomaly and irregularities so as to check for security loopholes or vulnerabilities that might occur as each system will need to be updated constantly as technology is constantly improving the number of loopholes or vulnerabilities will increase.

For students, it mostly lies in their studies where they may or may not need this when pursing their studies as it is an application for those keen in android application development as well as network related studies

***5.3 System Requirements***

* Android 4.0 (Ice Cream Sandwich) – Android 6.0 (Marshmallow)
* Chipset Qualcomm MSM8974 Snapdragon
* Wi-Fi 802.11 a/b/g/n/ac
* Compatibility with Nexmon

***5.4 Functional Requirements***

Under this section, we will be describing the functional requirements of our application.

***5.4.1 Start/Stop Packet Sniffing***

When users open the application, they will be able to select when they wish to start and stop the capturing of packets. The application works by utilizing a binary that will be started in a process with SuperUser permission. The process runs in a thread and stops only when the stop button is pressed. The packets are captured using the binary and are redirected to a text file.

***5.4.2 Display Information***

The application works by reading in the text file that is at the same time being written to by the sniffing thread. As data enters the text file, another thread is used to read in the packet data to the application. Another thread at the same time will get the data read in and display to the gui portion of the application.

Users will be able to view the packets captured by utilizing 3 threads, writing to a text file, reading a text file and updating to the display all being done concurrently.

***5.4.3 Save Packets to File***

After the user is done capturing packets, they will be able to save the data to a file which they can then use another application to view the data at their own time or for system administrators they will be able to keep records of the type of data that has been going through the network.

This packet information is saved in a ‘.pcap’ format contains more information that can be displayed on a mobile device. It would be better analyzed on a computer.

***5.4.4 Filter the Packets***

Our application would be designed to filter the types of packet that the user wishes to see, according to certain devices/websites/IP addresses/packet type.

***5.5 Non-functional Requirements (Quality of life Service)***

This section describes the non-functional requirements of our application that is designed to help users navigate through the application with as little difficulty as possible

## 6. Application Program

***6.1 Application Program Technical Aspect Introduction***

Before looking at the codes there are some technical aspect that needs to be introduced so that the user will be able to understand how our application works, what are the limitations, and possible problems.

***6.1.1 List of Application Program Technical Aspect***

The list below shows what is required for our application for it to be able to run

* Android Platform

This section talks about the device used by our application as well as introduction to the device and possible constrains faced.

* Operating System:

The application is run based on the Operating System (OS) of a mobile device known as Android, it is currently being used in most mobile devices such as Samsung, the only other known OS would be iPhone OS (iOS).

* Android Version:

Users need to know that their Mobile devices have different versions as the device must undergo updates consistently due to maintenance and fixes. Currently the Android Version most up to date is version 9.0 Pie. However, our application runs on Version 4.0 Ice Cream Sandwich to Version 6.0 Marshmallow, so this means newer devices might not be able to support our application which brings us to the first hurdle, acquiring a device for use.

* Chipset:

Mobile Phones have a design like that of a Computer Central Processing Unit (CPU) known as chipset. However, the design of a Chipset is like that of a Motherboard of a Computer, it accepts all the components to sit upon it and connect with each other such as Camera, Bluetooth, Wi-Fi, touch screen with CPU/Flash Storage/Ram. Due to the earlier mentioned Android Version, it affects the chipset as well, as the chipset used by our application is called Qualcomm MSM8974 Snapdragon. It is a type of chipset used in a limited number of devices which is another hurdle as the list of devices with said chipset is limited and thus limits the device that can be used.

Some of the information has been taken from this link [8]

* Network

This section talks about network and how it is related to our application as well as the possible constrains

* IEEE 802.11

a set of Media Access Control (MAC) and physical layer (PHY) specifications implementing wireless local area network (WLAN) it is the world’s most widely used wireless network standards it is used by Mobile Devices to talk to each other and access the internet.

* WNIC

Wireless Network Interface Controller (WNIC) is a network interface controller which connects to a wireless radio-based computer network rather than using wired. It works on the Layer 1 and 2 of the Open Systems Interconnection Model (OSI Model). It is usually applied to IEEE 802.11 adapters.

Modes of Operation are Infrastructure and Ad Hoc

For Infrastructure the WNIC need a wireless Access Point, all data is transferred using the access point as a central hub, all nodes in an infrastructure connect to an access point. All nodes connecting to the access point must have the same Service Set Identifier (SSID) as the access point. And if any kind of wireless security is enabled on the access point (WEP/WPA) they must share the same keys or other authentication parameterss

For Ad Hoc Mode, the WNIC does not require an access point but rather can interface with all other wireless nodes directly. All the nodes in an ad hoc network must have the same channel and SSID

In a 802.11 WNIC, the MAC Sublayer Management Entity (MLME) can be implemented either in the NIC’s hardware or firmware or host-based software that is executed on the main CPU, and a WNIC that implements the MLME function in hardware or firmware is called a FullMAC WNIC or HardMAC NIC, and a NIC that implements it in host software is known as SoftMAC NIC. FullMAC chips are typically used in mobile devices.

* 802.11 Frames

Data Frames which are considered the pack horses, where data is being hauled from station to station, data frames carry higher-level protocol data in the frame body, before the frame body itself it has frame control, duration ID, address 1 (receiver), address 2 (sender), address 3 (filtering), Seq-ctl, and an optional address 4 the type of data frames are Data which is moving the frame body from one station to another and Null which consist of a MAC header followed by the FCS trailer.

Control Frames used in conjunction with data frames to do area clearing operations, channel acquisition, carrier-sensing maintenance functions, and acknowledgement of received data. Control frames assist in the delivery of data frames and all control frames use the same frame control field.

* Management frames which does functions regarding supervisory, such as joining and leaving a wireless network, move associations from access point to access point. An identity of a network station can be broken down into three components, the first is mobile stations in search of connectivity must first locate a compatible wireless network to use for access, second, the network must authenticate mobile stations, third mobile stations must associate with an AP. For all management frames structure the MAC header is the same in all the management frames. And for the frame body most of the data contained in the frame body is fixed fields (fixed-length) or information elements (variable-length). 802.11 management frames have MAC headers with 3 addresses fields in it, for 802.11a/b/g it has 24-byte MAC Header, 802.11n has 28-byte (extra byte is HT control field)

So, what are fields? Fields also have authentication algorithm number with values 0 (open system authentication), 1 (shared key authentication), 2-65535 (reserved). Apart from authentication algorithm fields contain an authentication transaction sequence number used to track progress through the authentication exchange as well as beacon interval used to set the number of time units between beacon transmissions. The beacon transmission contains capability information to advertise the network’s capabilities with each bit being used a s a flag to advertise a particular network function. Other things in field are current AP Address (MAC address of the access point which they are associated), Listen interval (number of beacon intervals that stations wait between listening for beacon frames), association ID (stations associated with an access point are assigned association ID to assist with control and management functions), timestamp (synchronize between stations in a BSS), reason code (disassociation or de-authentication frames, when the sender has not properly joined the network part of the frame is a 16-bit reason code field), status code (success/failure of operation).

For fields in a management frames it may have up to 10, there is a duration field which have 4 rules, contention-free period of duration: 32768, frames transmitted to a broadcast/multicast destination duration: 0, if more fragments bit is 0 duration: SIFS + ACK, if more fragments bit is 1 duration = fragment+3x SIFS+2x ACK.

Management frames generic information element contains an ID Number, length and variable-length component. Within the information elements there is also Service Set Identity (SSID) which allows network managers to assign an identifier to the service set, whenever stations attempting to join a network may scan an area for available networks and join the network with specific SSID. SSID is the same for all the basic service areas composing an extended service area. Apart from SSID there are also others such as supported rates which consist of a string of bytes that uses seven low-order bits for the data rate, most significant bit indicates whether the data rate is mandatory and up to eight rates may be encoded in the information element. Management frames also has their types like Beacon (important part of many network maintenance task), Probe Request (mobile stations use it to scan an area for existing 802.11 network), Probe response (it is sent if the request encounters a network that has compatible parameters), IBSS announcement traffic indication map (ATIM) is when the station has a buffered frames for a receiver in low-power mode, This is sent during the delivery period to notify recipient it has buffered data, disassociation and deauthentication, association/re-association request/response, authentication.

* HardMAC(FullMAC) & SoftMAC

HardMAC describes a WNIC which implements the MAC Layer in hardware & SoftMAC refers to a WNIC which does not implement the MAC layer in hardware, rather it expects the drivers to implement the MAC Layer.

Advantages of SoftMAC is the potentially lower hardware costs, upgrade to newer standards by updating the driver only, possibility of correcting faults in the MAC implementation by updating the driver only, one other advantage that applies only in the Linux Kernel at least is that many different drivers for different types of WNIC can all share the same MAC Implementation. However, despite all the advantages, not all WNIC uses SoftMAC.

Mac80211 is the framework within the Linux Kernel for implementing SoftMAC Drivers. It implements the cfg80211 callbacks which would otherwise have to be implemented by the driver itself and implements MAC Layer function which means that it goes between cfg80211 and the SoftMAC Drivers

The Main Advantage of HardMAC is that since MAC functions are implemented through the hardware, they contribute less CPU load.

For HardMAC, the drivers have to implement the cfg80211 interfaces fully themselves.

* Monitoring and Promiscuous Mode

This section explains the 2 modes used mainly in our application of Sniffing Data Packet as well as possible limitations faced when using these 2 modes for our application.

* Promiscuous Mode

It is a mode for a network interface controller (NIC) or wireless network interfaces controller (WNIC) that causes the controller to pass all traffic it receives to the central processing unit for both wired and wireless network, rather than passing only the frames that the controller is specifically programmed to receive. This mode is used for network sniffing. However, one issue is that not all wireless driver supports promiscuous mode in addition to this, if the network has any form of encryption such as WEP, WPA/WPA2, packets might be dropped because even if it can capture all traffic on the local network it might not be able to decrypt hence the reason for the packets to be dropped.

However, for promiscuous mode there is a chance in which it might not work. One reason is due to the Adapter and Driver as it is dependent on the implementation of promiscuous mode. If a network is protected with WEP the adapter might be able to show third-party packets, but for WPA/WPA2 it is required that a type of decryption which is like that of Wireshark does. This results in a need to generate “fake Ethernet” header which is based on 802.2 header, and SNAP header if present in data packets which then decryption is required. in order for it to be shown which is a major issue for our application as it runs of android instead of on a Computer and currently Wireshark is unavailable on Android and even if there are alternatives it might not be possible that it runs the same as how Wireshark does it. And most wireless access points are protected using WPA/WPA2 which makes it more difficult for our application when we do the network sniffing. Another issue is that they might not have bothered implementing promiscuous mode even for open networks. Promiscuous mode will not always allow you to see traffic while client isolation is in play.

* Monitor Mode

It is a mode which allows for a WNIC to monitor all traffic received from the wireless network, unlike promiscuous mode, monitor mode allows packets to be captured without having to associate with an access point or ad hoc network first. Monitor mode applies only to wireless network. Limitation is that usually the wireless adapter is unable to transmit in monitor mode and is restricted to a single wireless channel which may be dependent on the driver, firmware and features of chipset. Also, it does not check to see if the Cyclic Redundancy Check (CRC) values are correct for packets captured which means that a chance for the packets to be corrupted is present.

* Nmap (Network Mapper)

Apart from knowing the types of modes that the NIC needs be in to Sniff packets there are also other features in which is used in our application, one such feature is called Nmap. It is originally written by Gordon Lyon, where it is used to discover hosts and services on a computer network, which is thus built into a map of the network.

Nmap works by sending specially crafted packets to the target hosts and then analyses the responses from the target. Nmap features other than probing of the network it also includes host discovery and service as well as Operating System Detection. Which is a feature in our application in that every user that uses our application will want to know other than the type of packets being sent, are the types of Operating System that is being used to send these packets. Nmap can adapt to network conditions such as latency and congestion during scan. The features of Nmap also include Port Scanning, such as checking for open ports, Version detection.

Features of Nmap is that it is flexible that supports dozens of advanced techniques for mapping out network filled with IP Filters, firewall, routers and other obstacles, including port scanning mechanisms that work for both TCP & UDP. Nmap is powerful as it has been used to scan up to hundreds of thousands of machines which is considered a huge network, it is also portable as mentioned above that it is flexible as it supports most of the operating systems including Linux, Microsoft Windows, FreeBSD, OpenBSD, Solaris, Mac OS X, Sun OS and more. It is also supported by traditional command line as well as Graphical User Interface (GUI) and Nmap have Binaries available which we used in our application.

Some explanation of concepts have been taken from this link[9] [10]

* NexMon
* Nexmon is what we based our project on as it uses a Nexus 5 Phone with Broadcom BCM 4339 WiFi Chipset to do monitoring as well as packet-injecting as NexMon is Open. NexMon gives full control over a Wi-Fi Chip for research purposes is limited by firmware, which means it is hard to evolve communication protocols and test schemes in practical environments. Monitor mode as mentioned above allows eavesdropping on all frames on a wireless communication channel. This includes uses such as network packet analyses, security research and testing of new medium access control layer protocols, it is offered by SoftMAC drivers that implement the media access control sublayer management entity (MLME) in the driver rather than in the Wi-Fi Chip. On smartphones, mostly FullMAC (HardMAC) chips are used to reduce power consumption as the tasks do not need to wake up the main processor. Even though it is possible in FullMAC situations, it is generally not implemented in today’s Wi-Fi firmware’s used in smart phones which is one of the limitations that our project is facing, that is why in such situations we are bringing monitor mode to Nexus 5 Smartphones to enhance the interoperability between applications that require monitor mode and the BCM4339 Wi-Fi Chips.

Some explanation of concepts have been taken from this link [11]

* Nexutil & ioctls

In this section we talk about nexutil which is being used by our applications and include some details on what it can do.

* Nexutil is one of the utilities/libraries required for controlling the firmware as well as imitate a monitor interface to unmodified analysis and penetration testing (TCPDump)
* Talking to the firmware for many applications it is helpful to configure a firmware during runtime or extract information for debugging purposes one such means is to use ioctls to control the firmware and send events from the firmware to the host. To initiate transfers from the firmware, a user-space program known as nexutil can initiate a synchronous data exchange with the firmware by calling ioctls in the firmware. Each iotcl contain a command number, pointer to buffer to exchange data and length of the buffer, ioctls can either only set data or get data back from the firmware. For the two directions set and get nexutil offers two parameters -s<command\_number> and -g<command\_number> and may pass either integers, strings, raw data from the Standard input or based64 encoded raw data to the firmware. While ioctls are always initiated by the host, Wi-Fi firmware can also create an event and send it as a message to the host, where it is handled in the driver.
* FullMAC Wi-Fi Chips consist of two processers, one is ARM Processor that performs not time critical tasks (implemented in the driver for SoftMAC). The second processor implements a programmable state machine (PSM) which runs in the D11 core which is responsible to quickly process MAC-Layer Events. The D11 core then decides which frames should be dropped or which should be answered by acknowledgements. In Broadcom Wi-Fi Chips the D11 core does real-time MAC Processing as well
* Nexutil (handle receptions)

When monitor mode is active (calling nexutil -ml), this unfunction will call a wlc\_monitor function that will extract the received statistics and write them into a structure which will then pass both the statistics and frame to the wlmonitor function, which is a function that is hooked to implement monitor mode with radiotap headers

[12]

## 8. References

[1] <https://www.keycdn.com/support/udp-file-transfer/>

[2] <https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970>

[3] <https://www.keycdn.com/support/tcp-flags/>

[4] <https://opensourceforu.com/2013/09/what-a-native-developer-should-know-about-android-security/>

[5] <https://nmap.org/book/tcpip-ref.html>

[6] <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html>

[7] <https://mrncciew.com/2014/09/27/cwap-mac-header-frame-control/>

[8] https://en.wikipedia.org/wiki/Android\_(operating\_system)

[9] <https://en.wikipedia.org/wiki/Nmap>,

[10] <http://insecure.org/fyodor/>

[11] https://hackaday.com/author/hexagon5un/

[12] https://github.com/yuxuannn/Documentation/blob/master/Research/dissertation\_2018\_matthias\_thomas\_schulz.pdf

# **Report**

## 1. Project Overview

Designing of an Android Network Sniffer which in our team’s understanding is to achieve real-time scanning of network traffic and recording it to a file for later use. The application would be useful for people with Networking and Security background as the people who use this application would be System Administrators, Similar Application Developers and possibly general users. As such an application is already existing on computers our implementation of it onto a device like android has its limitation but it also thrills us to try and develop an actual working version of it.

***A) Vision***

* Our key feature of the application would be real time capturing of packets and being able to see them displayed on the application.
* To be able to save packet data into a file for later view.
* After being able to achieve the first objective the second one would be to filter traffic based on certain criteria’s such as type of device or maybe specific type of traffic
* If possible be able to scan the access point for devices connected to it.

***B) Initial Use-Case Model***



***C) Project Plan***

* Due to the limitations of the Chipset our available device has been restricted and therefore plan to use Nexus 5 due to concerns such as device cost
* Due to the need to have root access, which may cause bricking to a phone we have opted to use older no longer in use devices
* The device needs to be in monitor/promiscuous mode our choice of device has been restricted.
* The rooting methods may or may not be suitable for our device so therefore we need to have alternate ways to get network traffic such as using VPN
* Using libraries from Pcap to assist with the project as well as tcp dump

## 2. Roles and Responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| Team Number : < **SS18/2A** > | | | |
|  | **Student Name** | **Role** | **Artefacts** |
| 1 | Soh Yu Xuan | Team Leader, Lead Programmer | NetSniffer development, Nmap, Nexutil, Tcpdump binaries, Demo Video, User Manual, Testing, Technical Manual, Website, Research |
| 2 | Chin Tian Zhi Timothy | Sub Programmer | NetSniffer development, Technical Manual, Pcmon binary, Testing, Promo Video, UML Diagrams, Research |
| 3 | Huang Shun Yang Kenneth | Sub Programmer | Pcbin, Pcmon binary, Testing, Technical Document, UML Diagrams, Research |
| 4 | Tan Siyuan Kendrick | Team Manager, Lead Documenter | Technical Document, UML Diagrams, Research, SRS, Report, Website, |

## 3. Risk Analysis and Counter Measures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Impact Type** | **Risk Seriousness (%)** | **Likelihood of Occurrence (%)** | **Risk Description** |
| 1 | Successful Task completion | 90% | 50% | Being able to complete rooting as it is the main criteria failing to do so will result in needing alternative methods which cause re-scope of project |
| 2 | Deadline | 90% | 10% | If deadline is not met all team members might fail the project |
| 3 | Unable to display on phone’s application | 50% | 20% | If application does not display correctly the bulk of the project might not work needing more time to debug |
| 4 | Unable to obtain device for project | 90% | 20% | If we are unable to acquire a device that meets our criteria |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Risk Description** | **Proposed Mgmt Plan** | **(Possible) Reduction in Risk Seriousness (%)** |
| 1 | Unable to root phone | Plan 1 : Search for alternative methods like VPN | -50% |
| Plan 2 : use alternate rooting method till successful | -25% |
| 2 | Unable to obtain device for project | Plan 1: Buy from people who are selling old phones | 75% |

## 4. UML Diagrams

***Design Artefacts - Use Cases (Iteration 1)***

Main use case diagram:



***Design Artefacts - Use Cases (Iteration 2)***

Main use case diagram

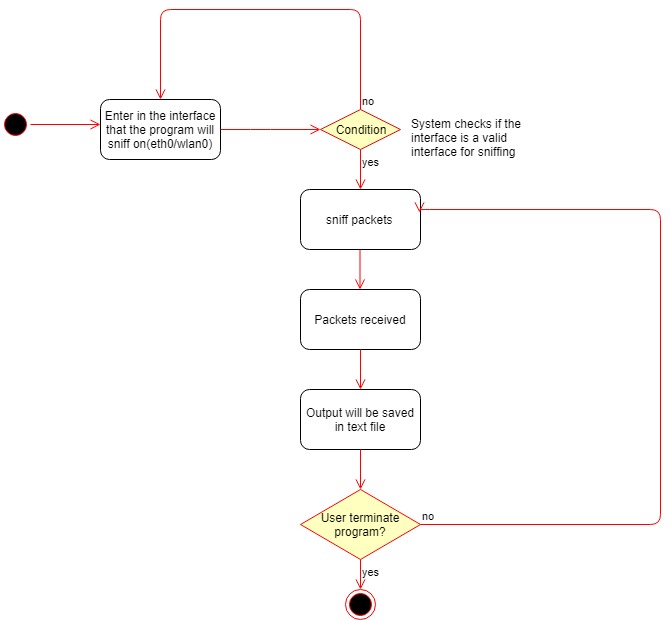
******

***Design Artefacts – Activity Workflows (Iteration 1)***

Android app activity diagram:

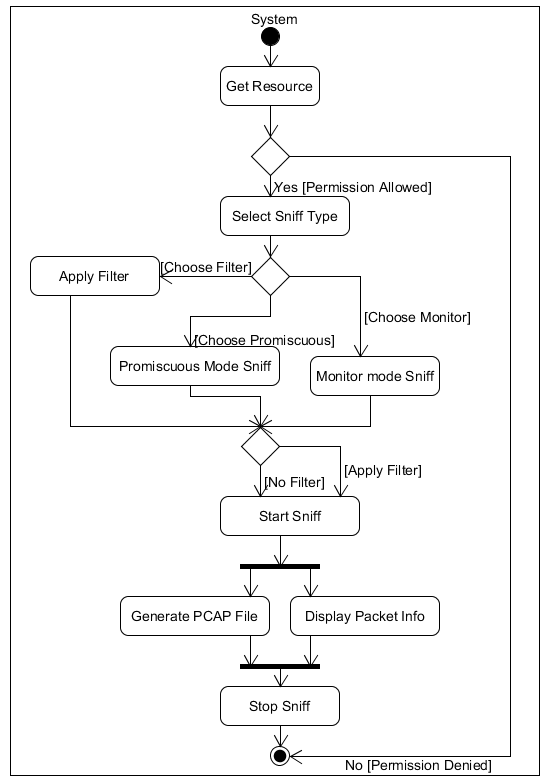


Pcbin activity diagram:

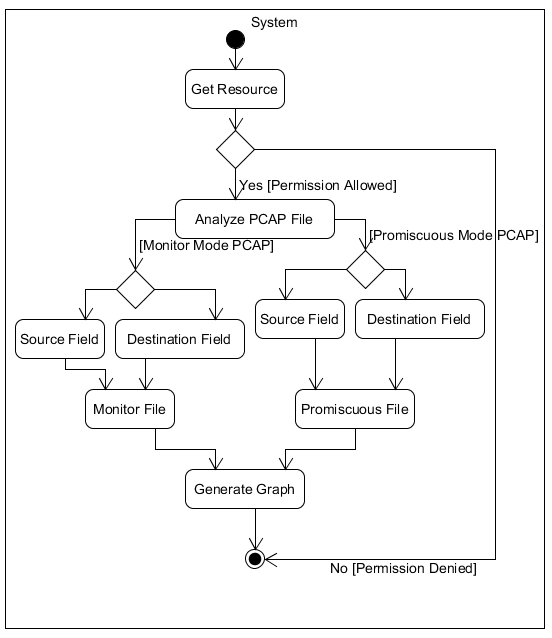


***Design Artefacts – Activity Workflows (Iteration 2)***

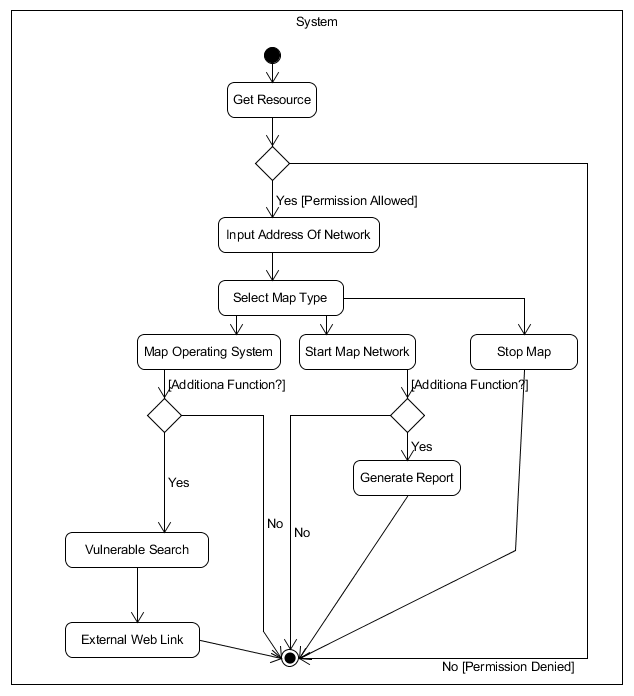
[UC-1] Sniff Packets



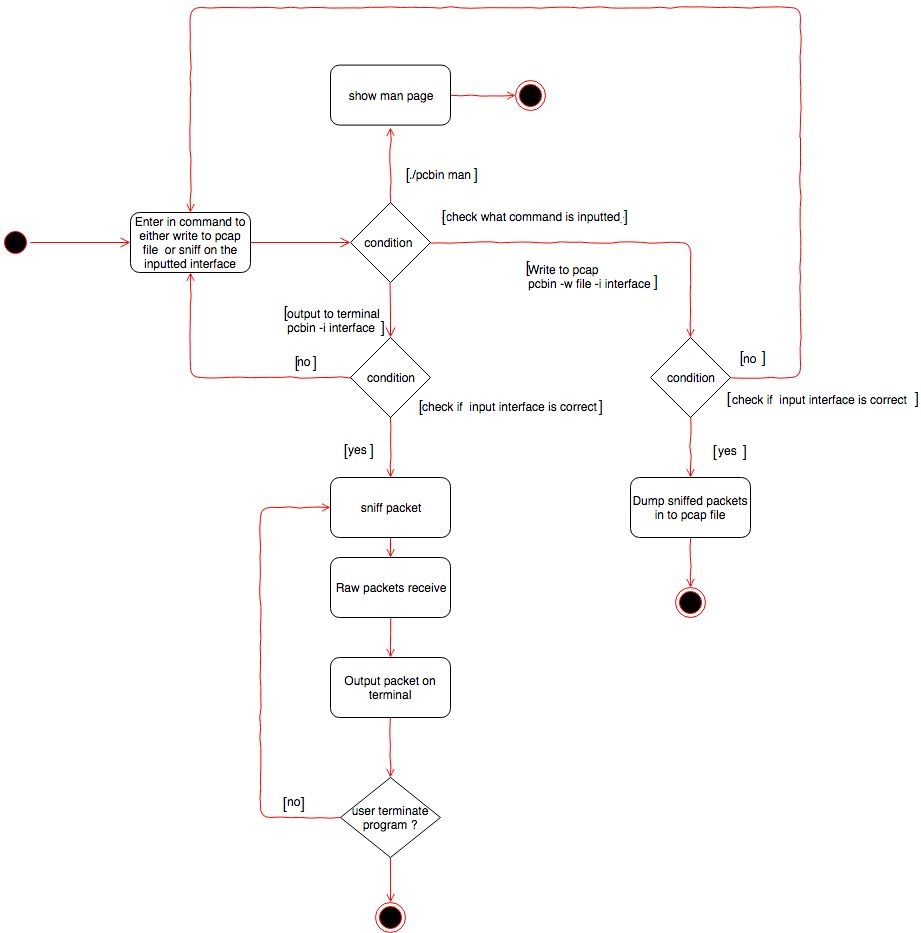
[UC-4] Analyze PCAP



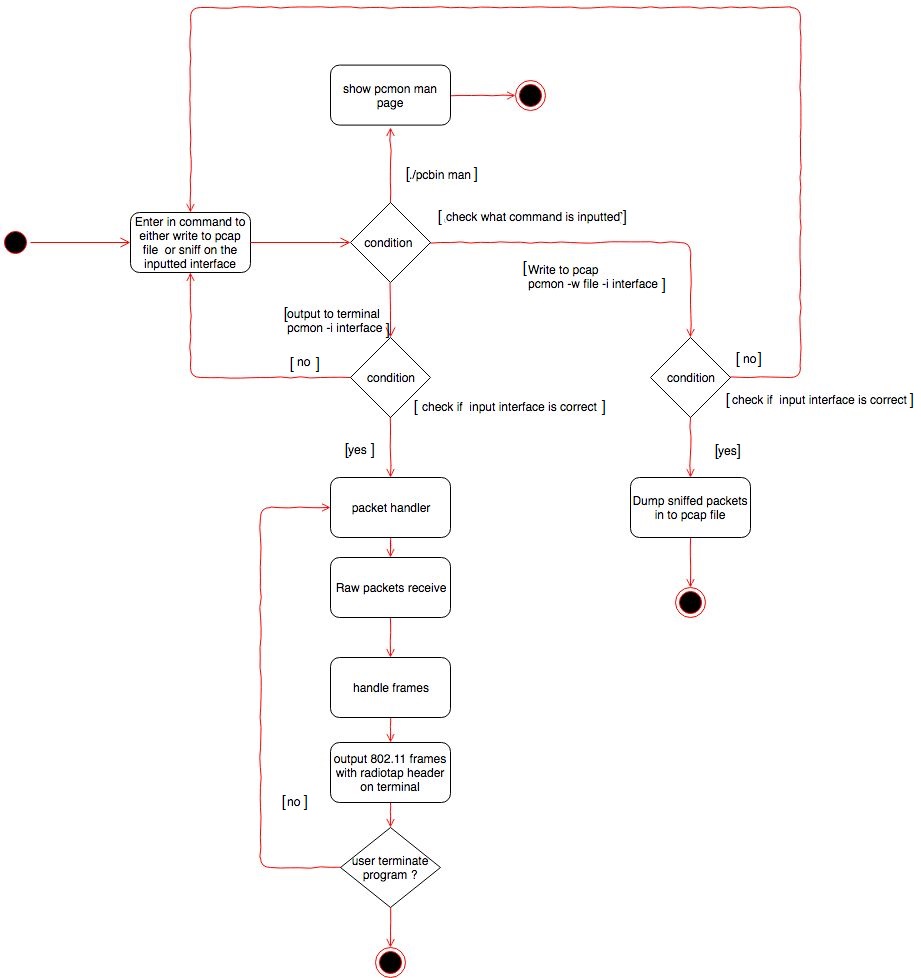
[UC-3] Map Network



Pcbin activity diagram

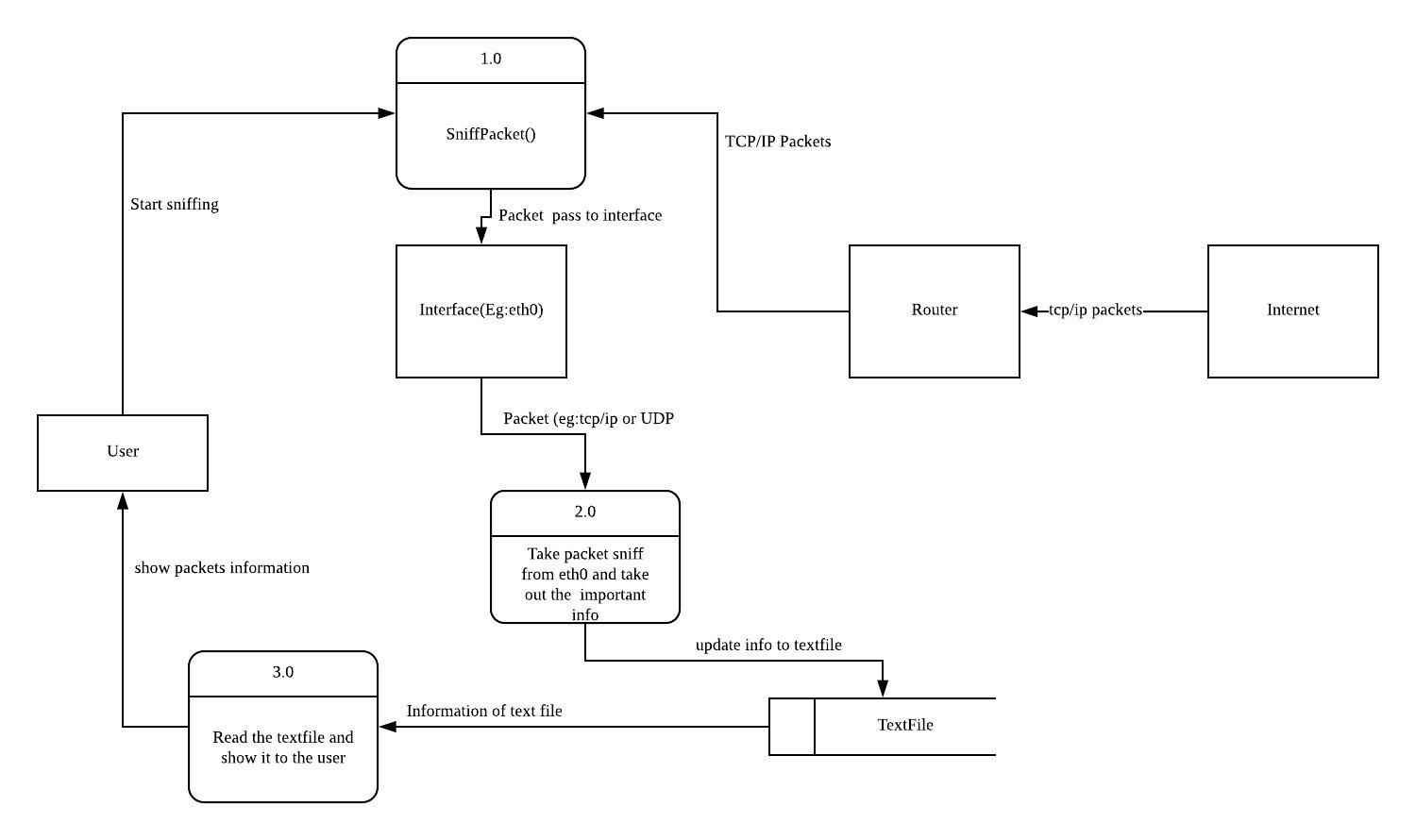
****

Pcmon activity diagram:



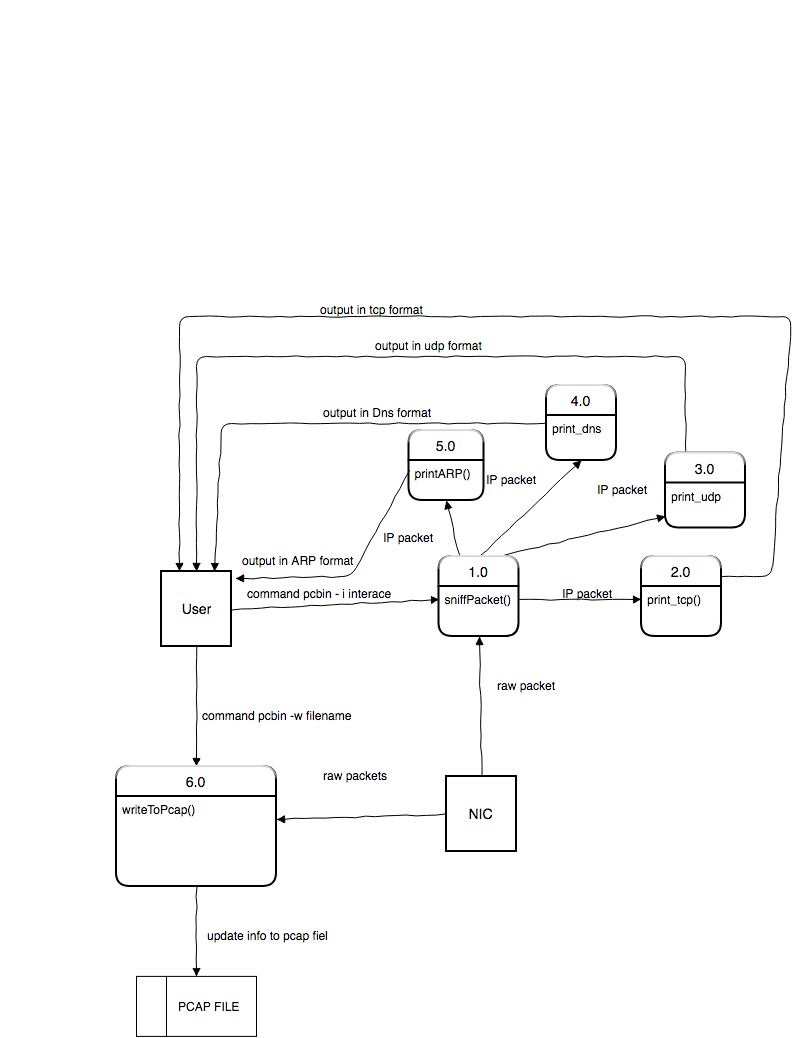
***Design Artefacts – Data Flow Diagram (Iteration 1)***

Pcbin data flow diagram:

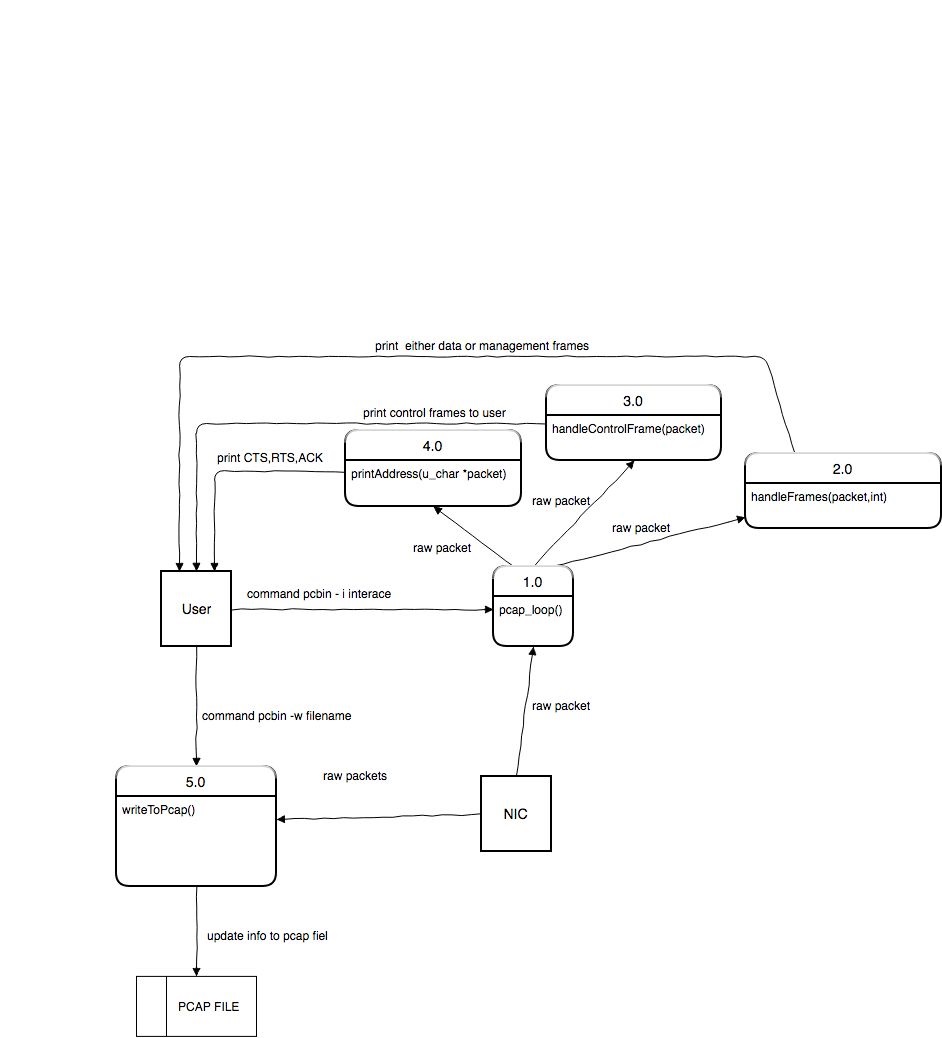


***Design Artefacts – Data Flow Diagram (Iteration 2)***

Pcbin data flow diagram:

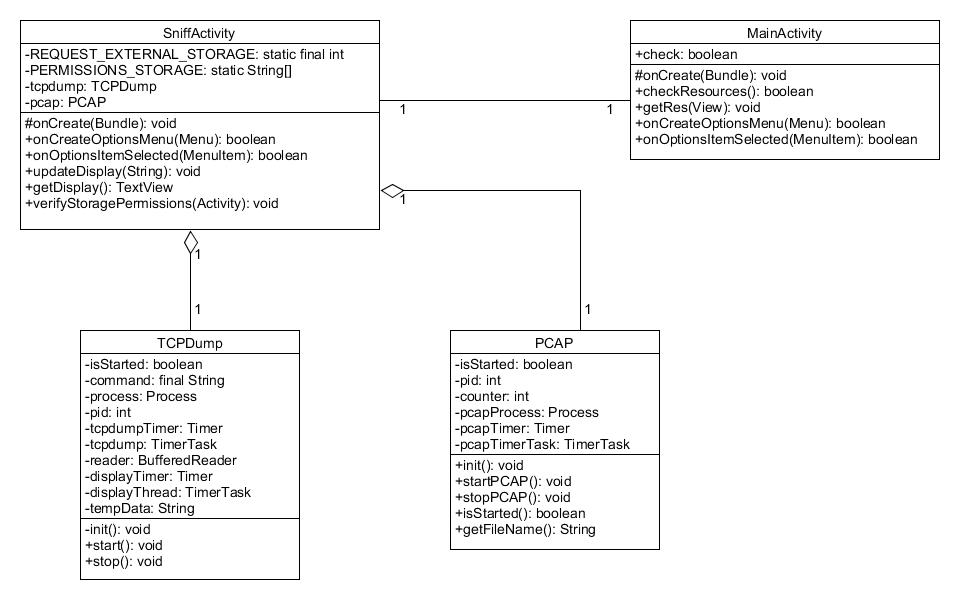


Pcmon data flow diagram:



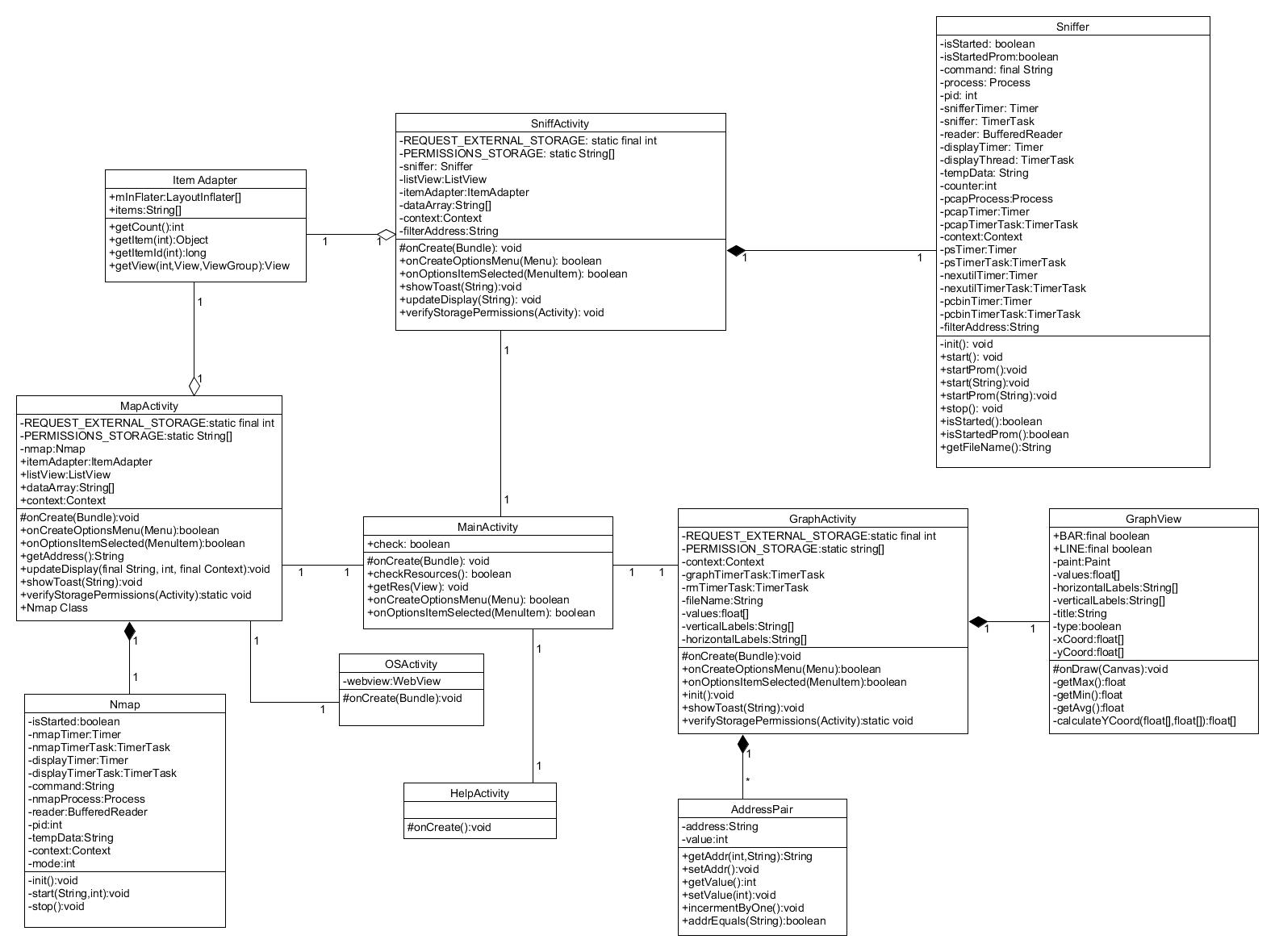
***Design Artefacts - Class Diagrams (Iteration 1)***

Main class diagram:



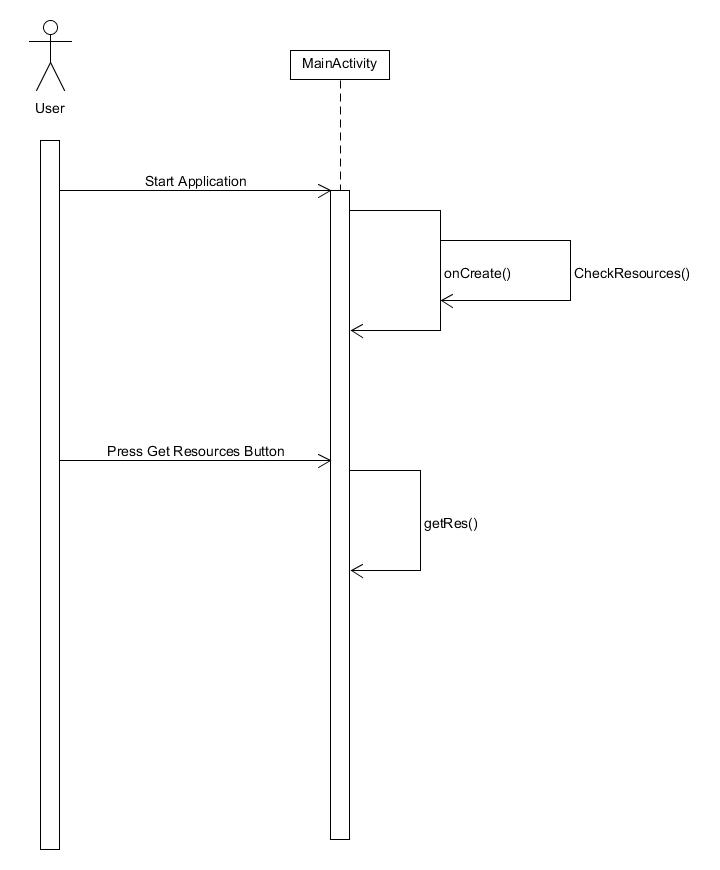
***Design Artefacts - Class Diagrams (Iteration 2)***

Main Class Diagram

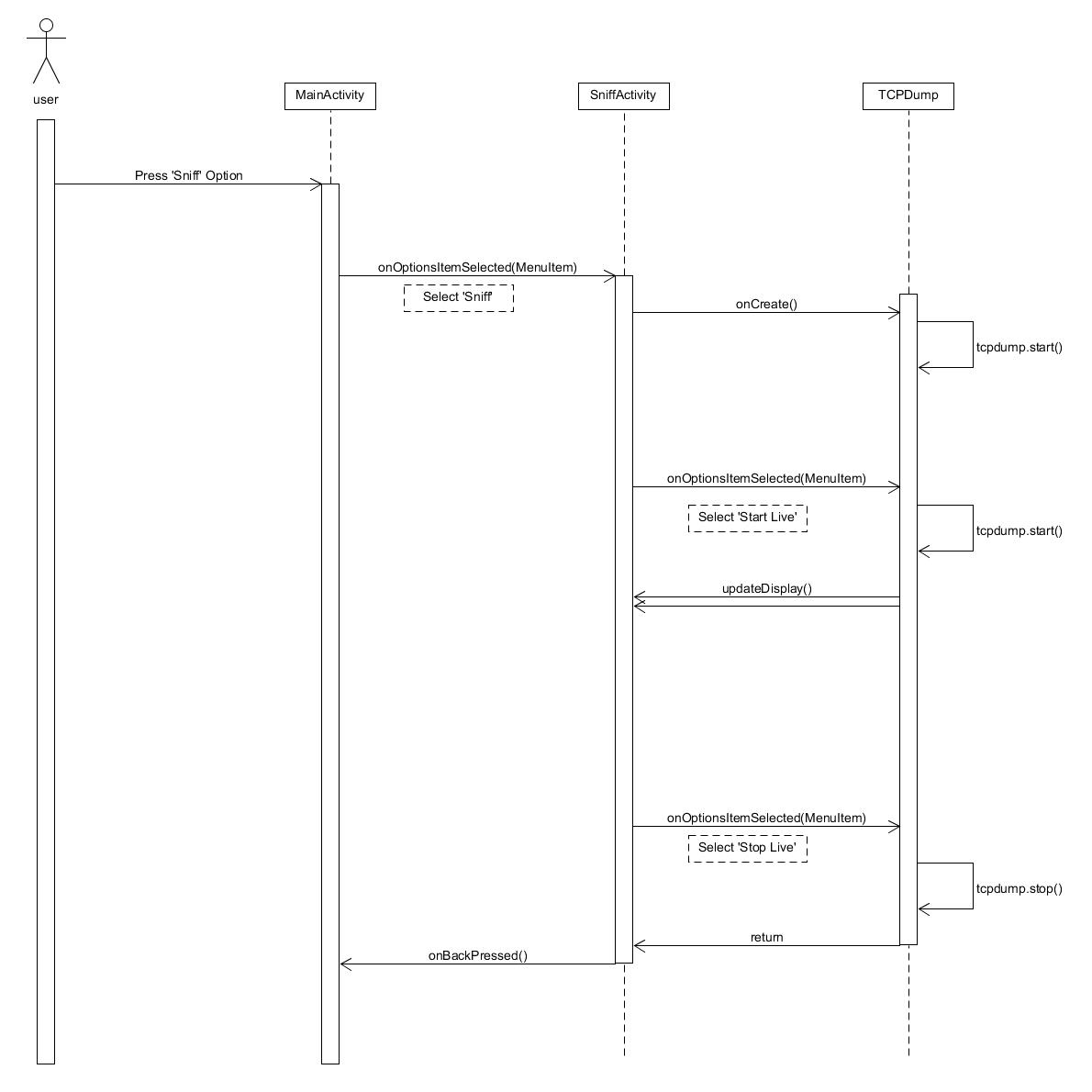


***Design Artefacts - Sequence Diagrams (Iteration 1)***

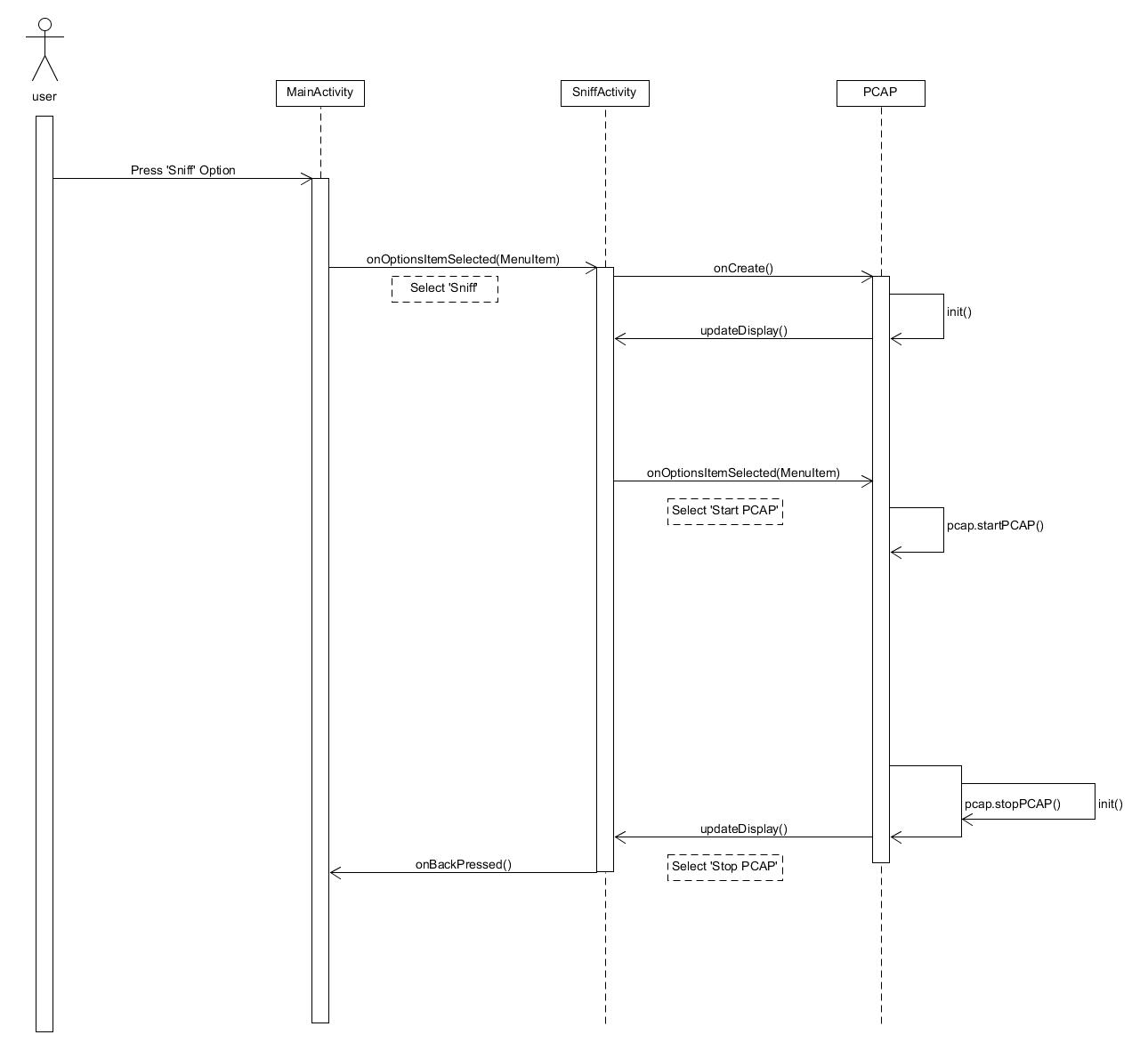
App initialize sequence diagram:



Start live sequence diagram:

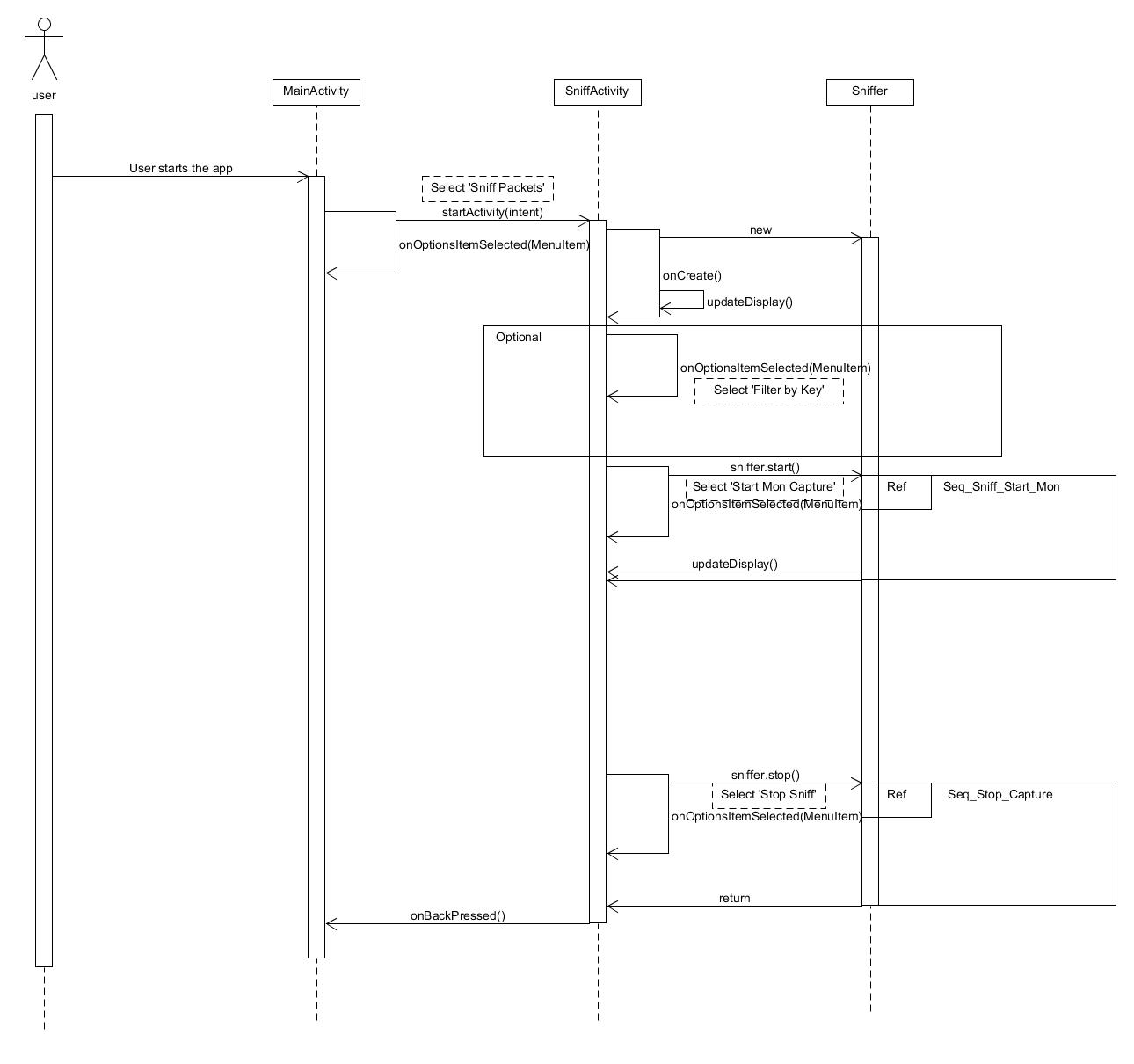


Start pcap sequence diagram:

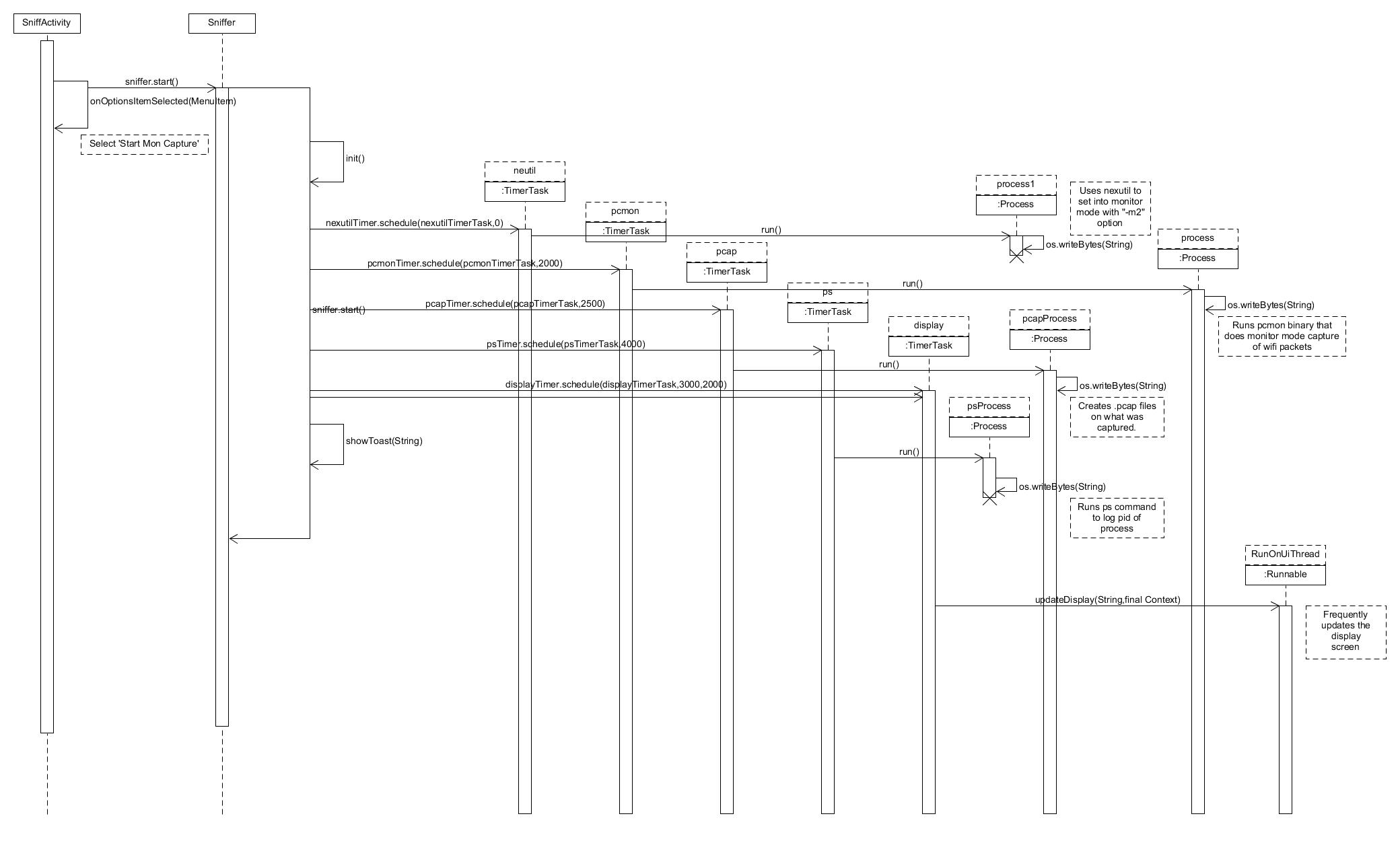


***Design Artefacts - Sequence Diagrams (Iteration 2)***

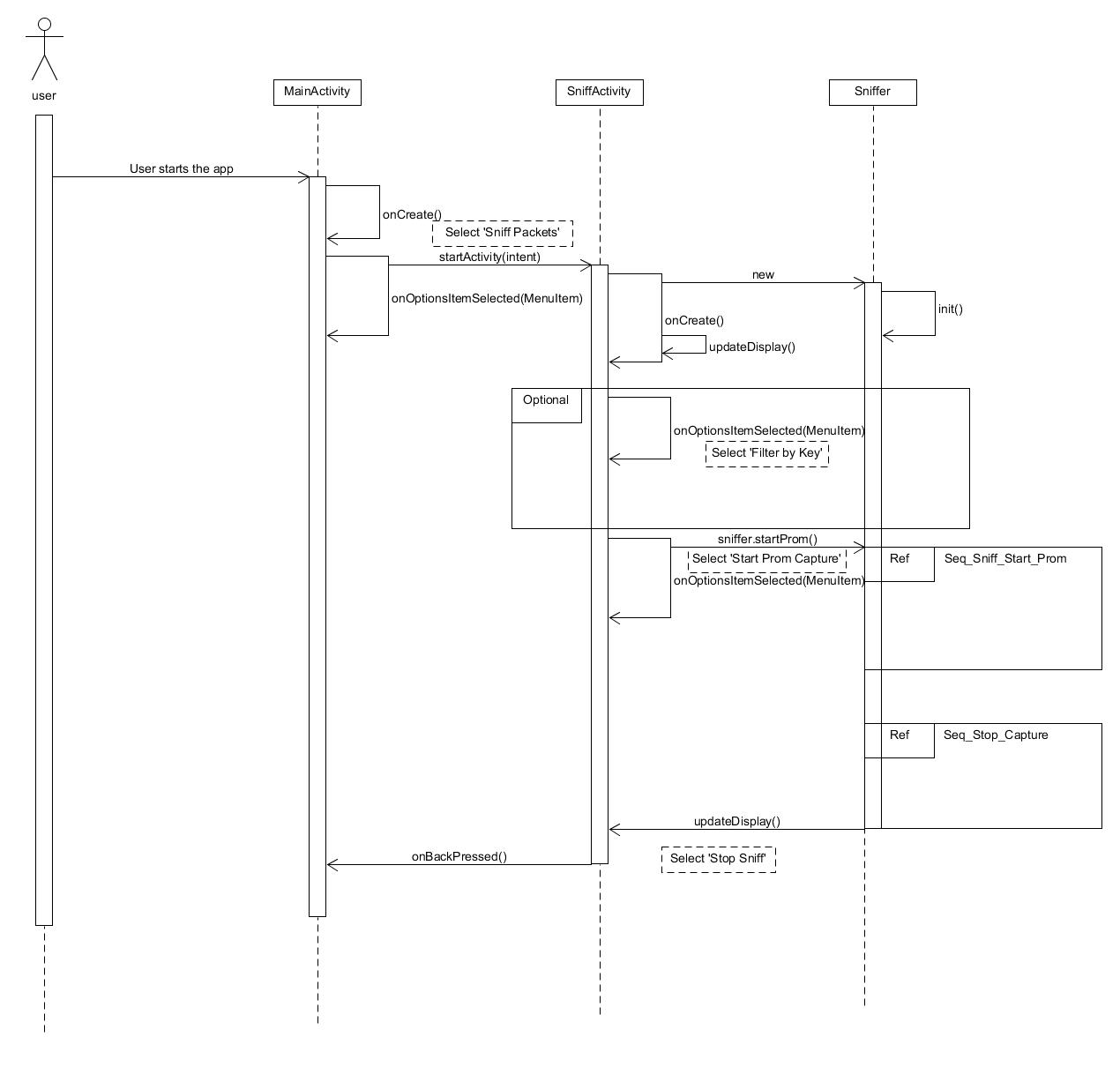
[UC-1.3] Start\_Mon\_Capture



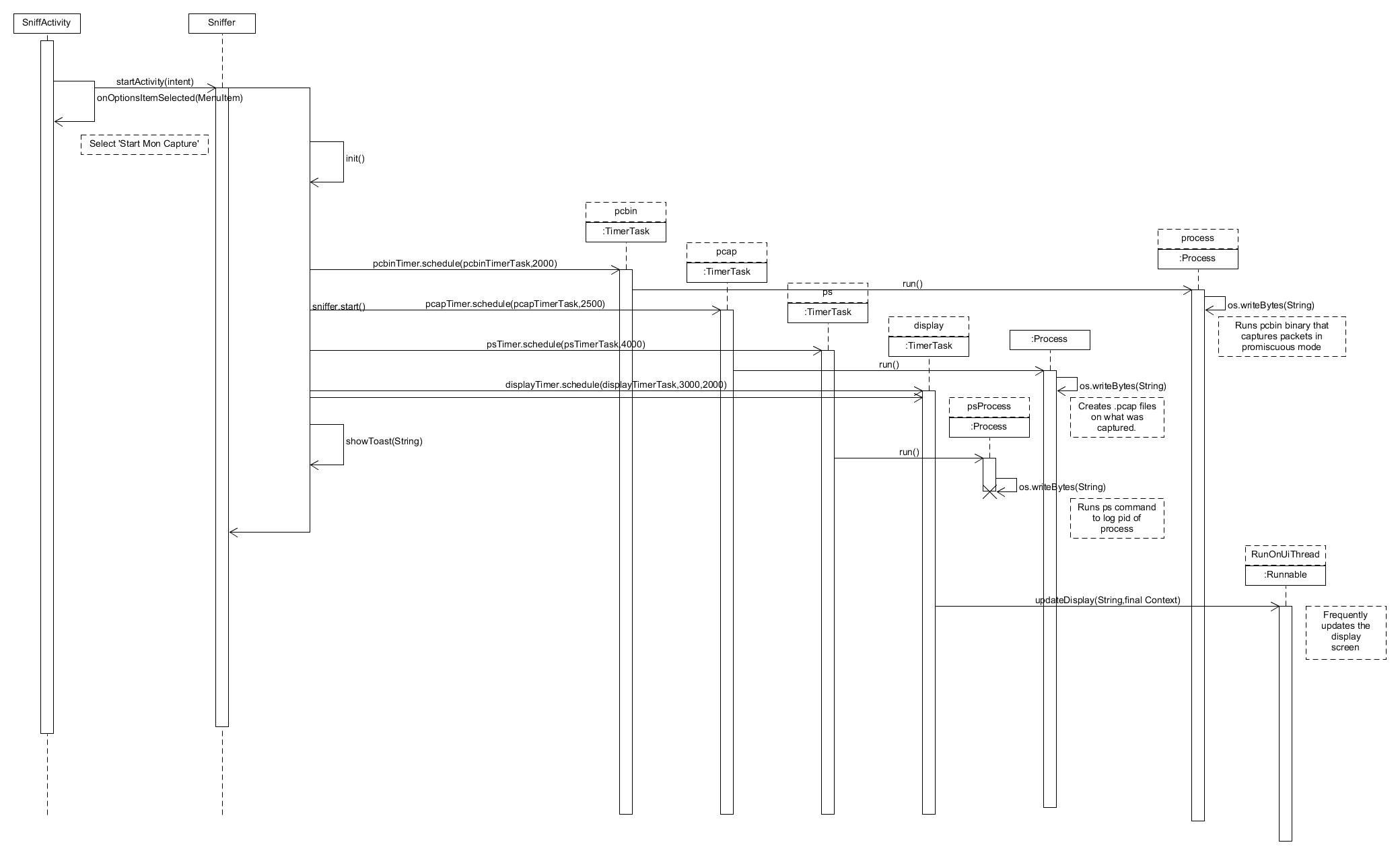
[UC-1.3] Sniffer\_Start\_Mon [ sniffer.start() ]



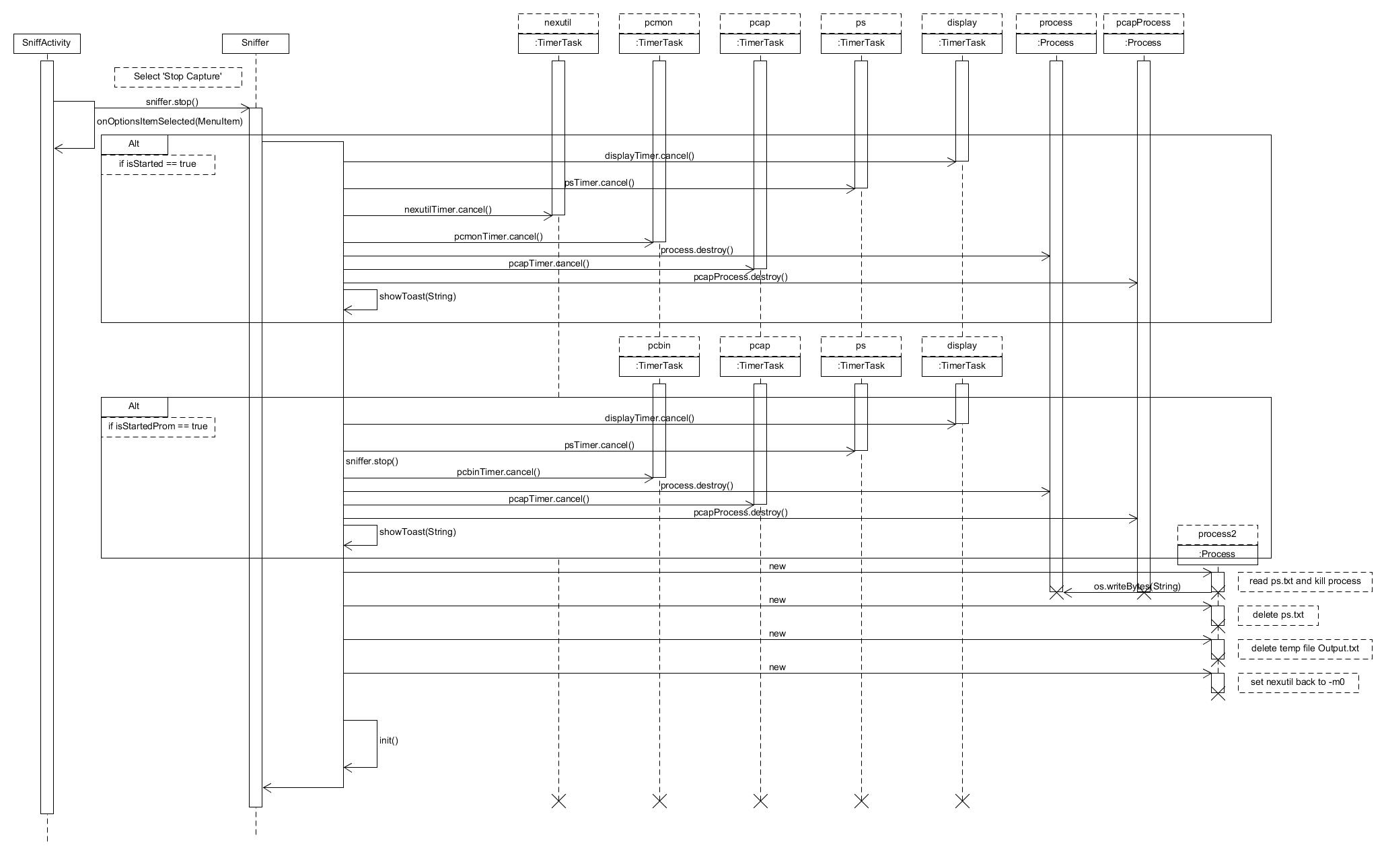
[UC-1.4] Start\_Prom\_Capture



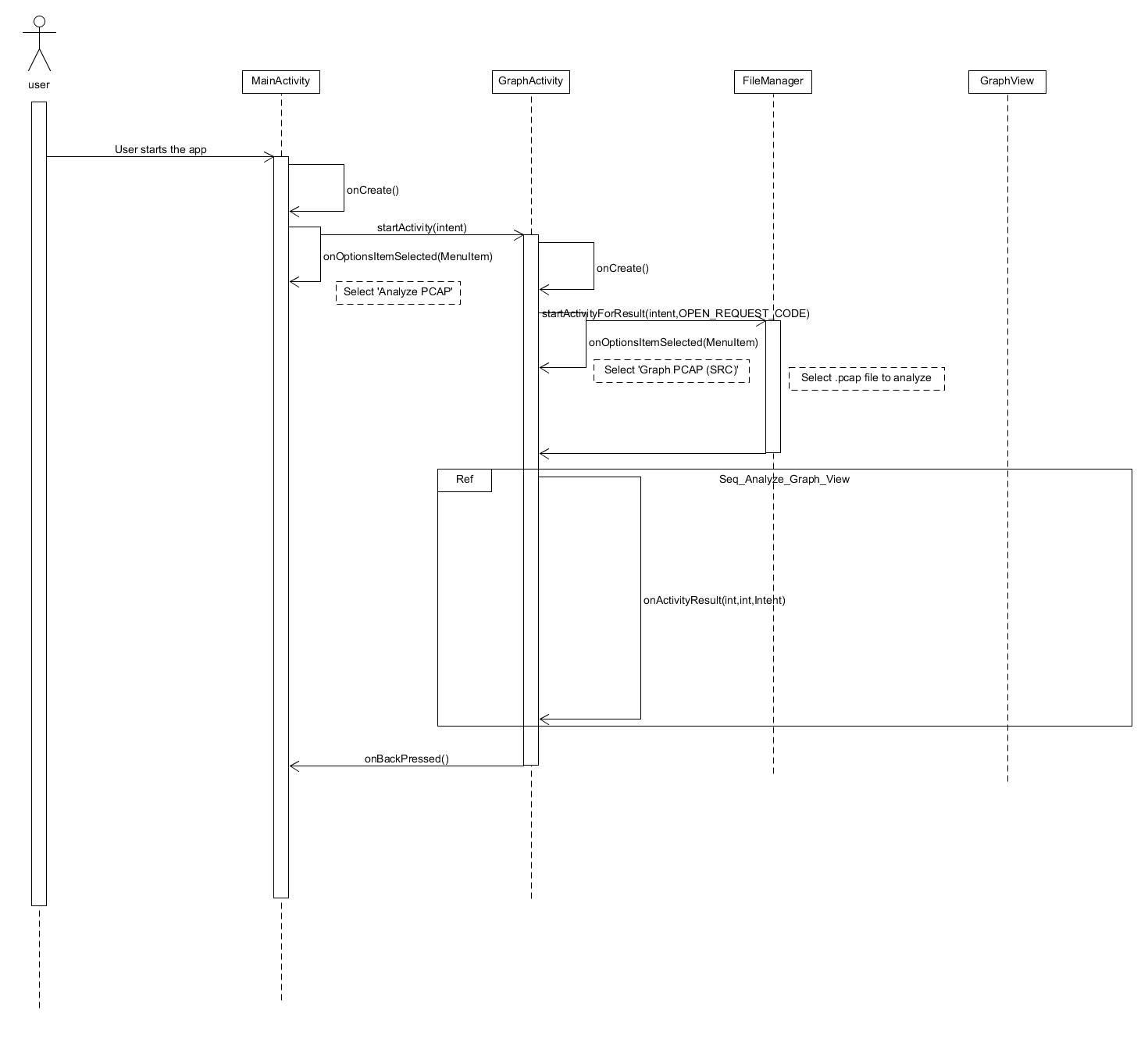
[UC-1.4] Sniff\_Start\_Prom [ sniffer.startProm() ]



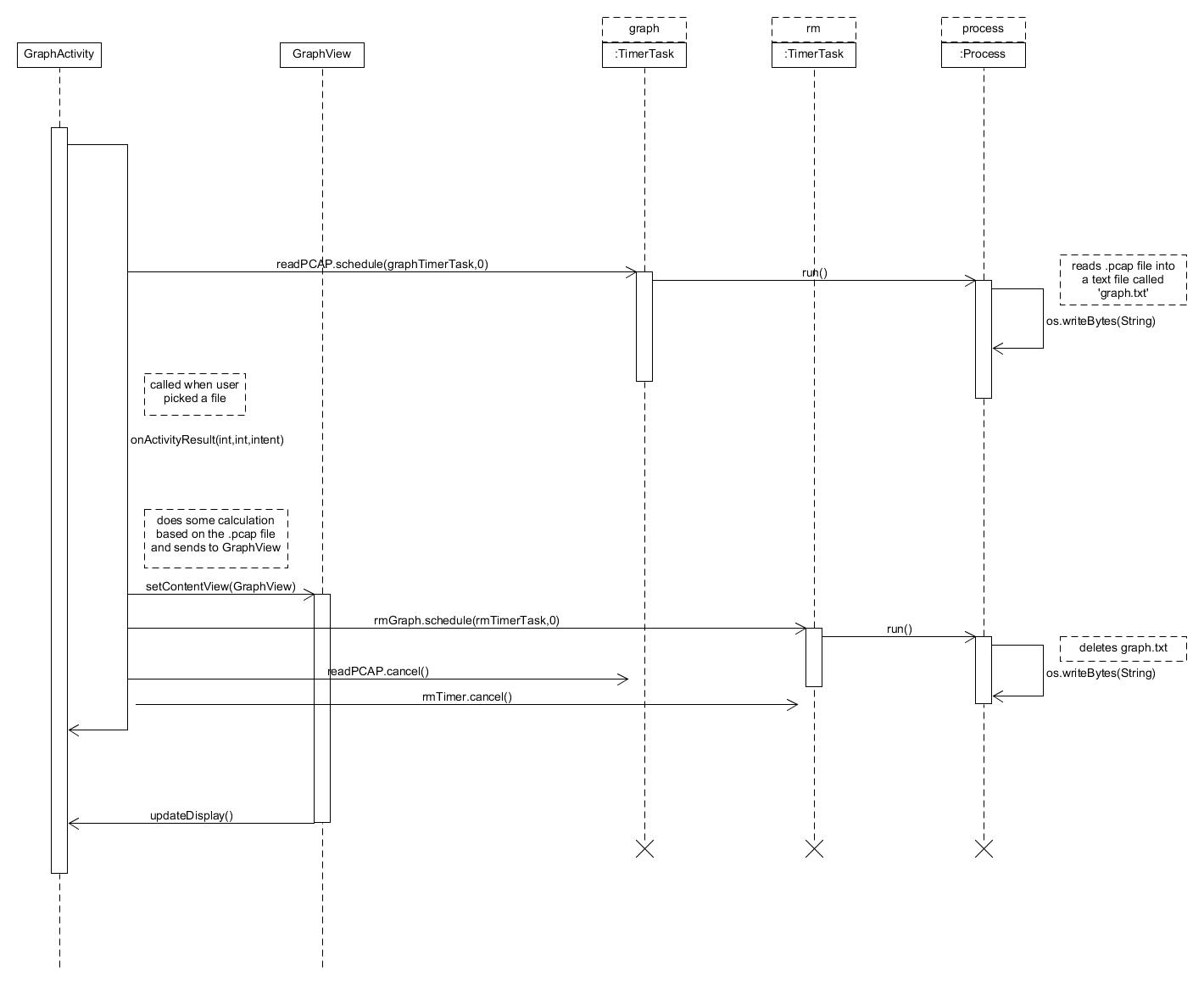
[UC-1.2] Stop\_Capture(Monitor & Promiscous)



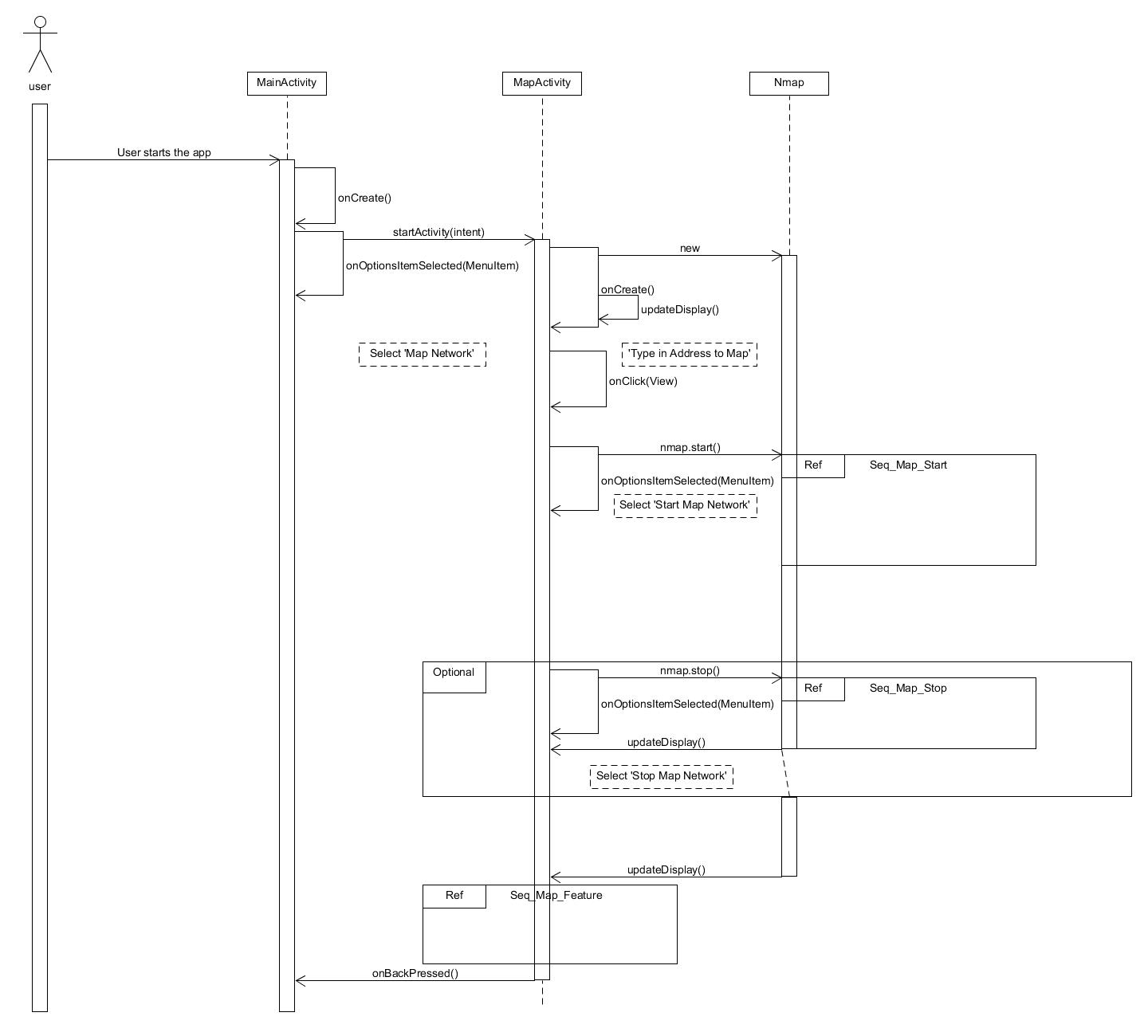
[UC-4] Analyze\_PCAP



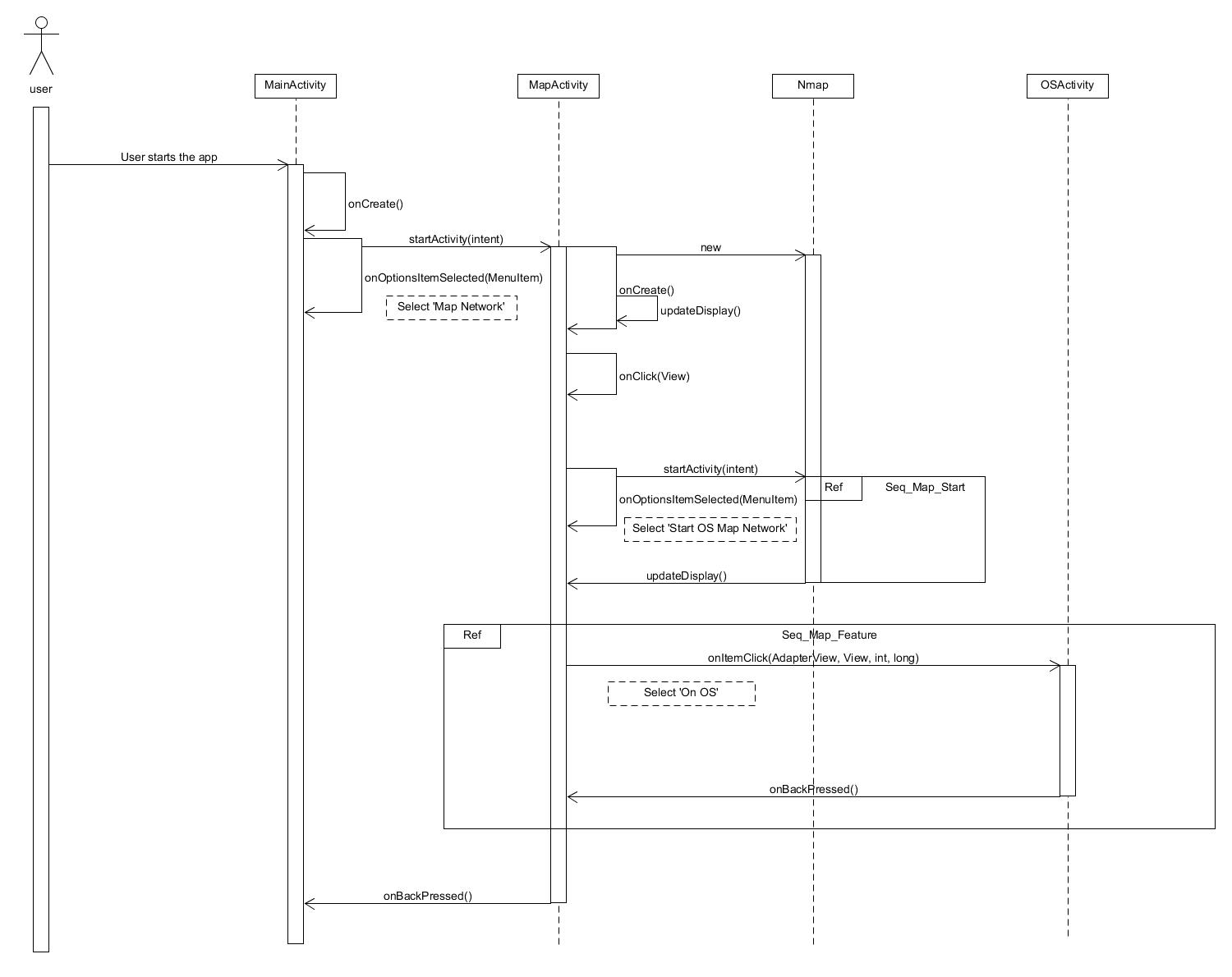
{UC-4] Analyze\_Graph\_View



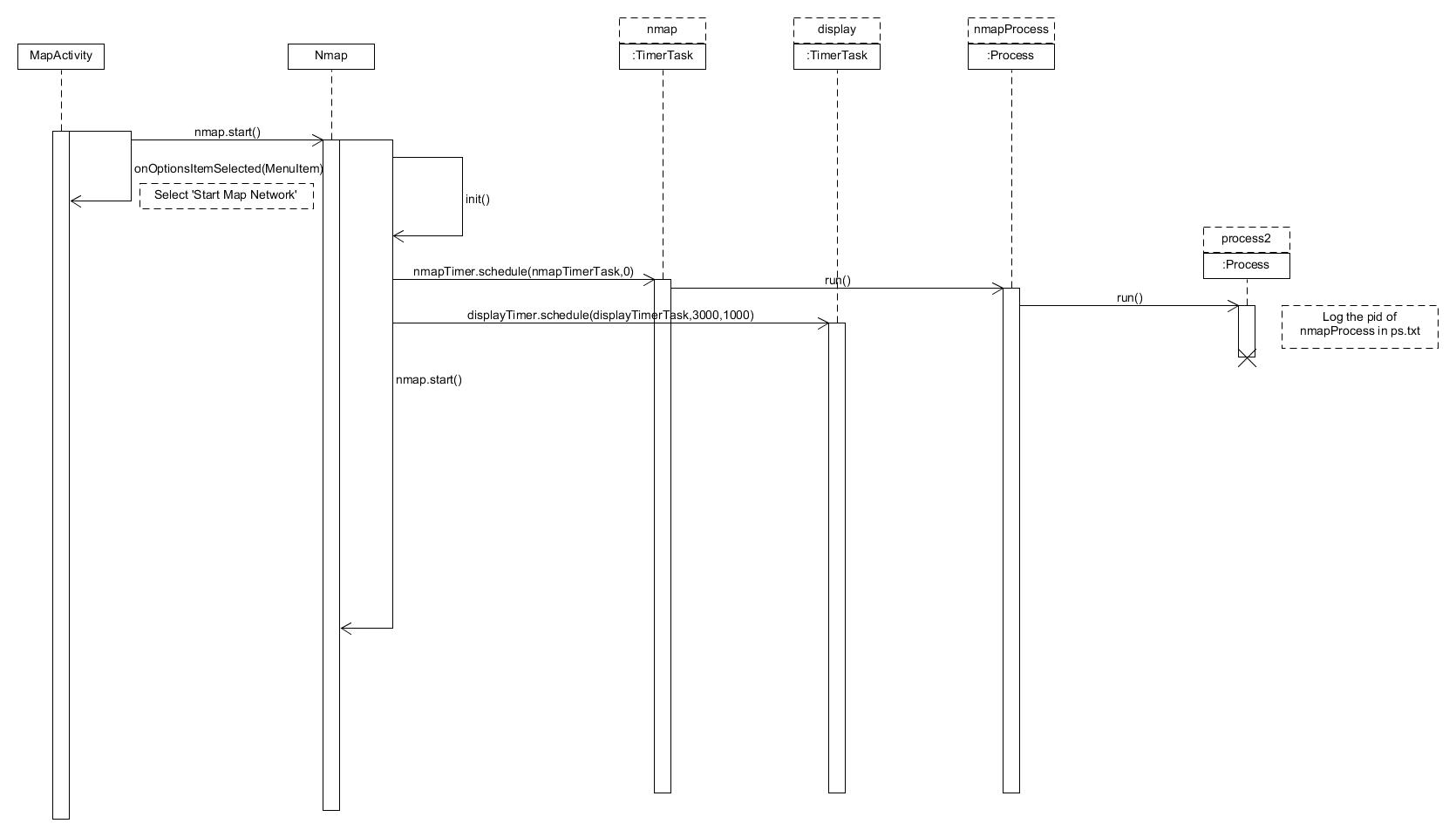
[UC-3.1] Start\_Map\_Network



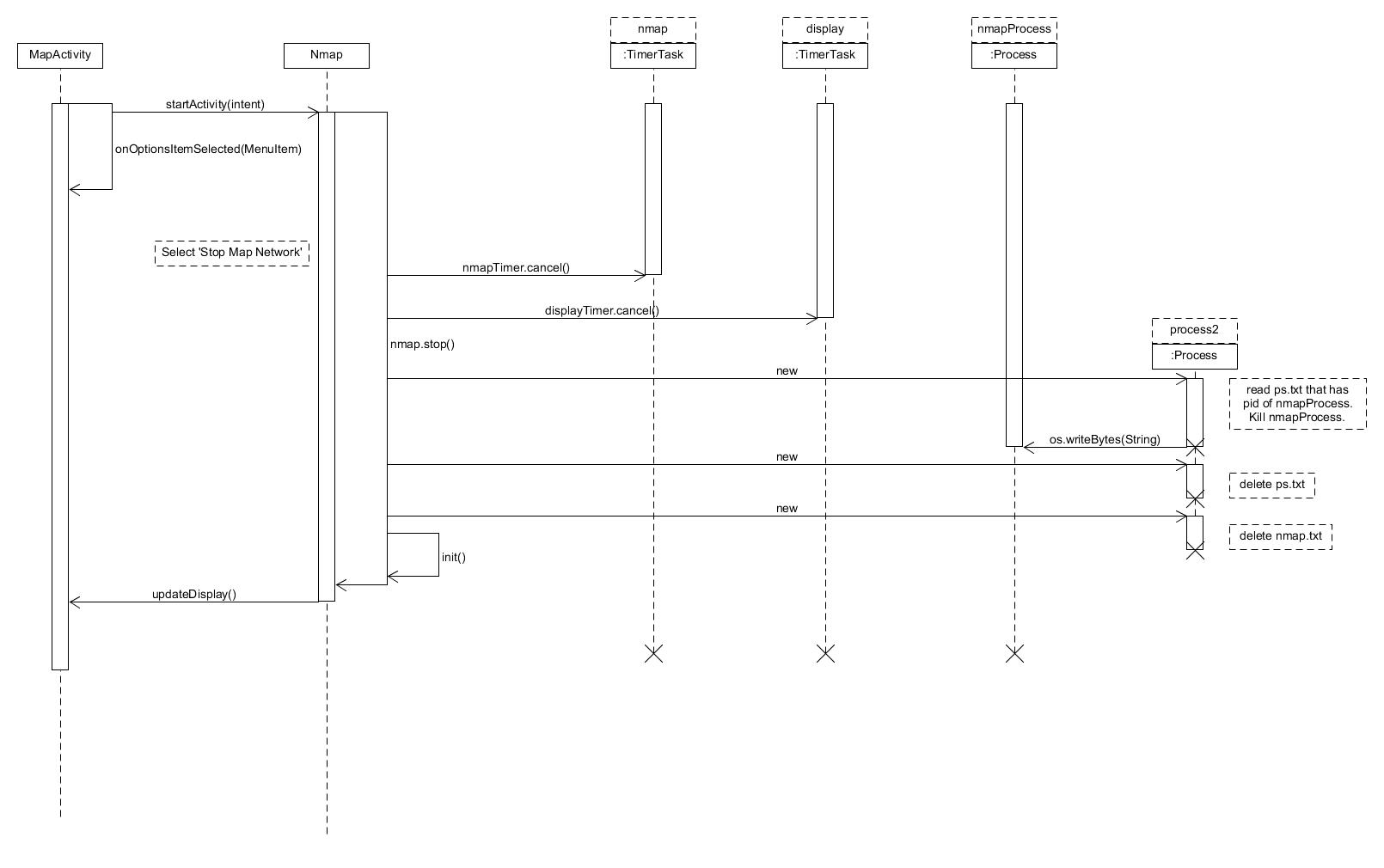
[UC-3.2] Start\_OS\_Map\_Network



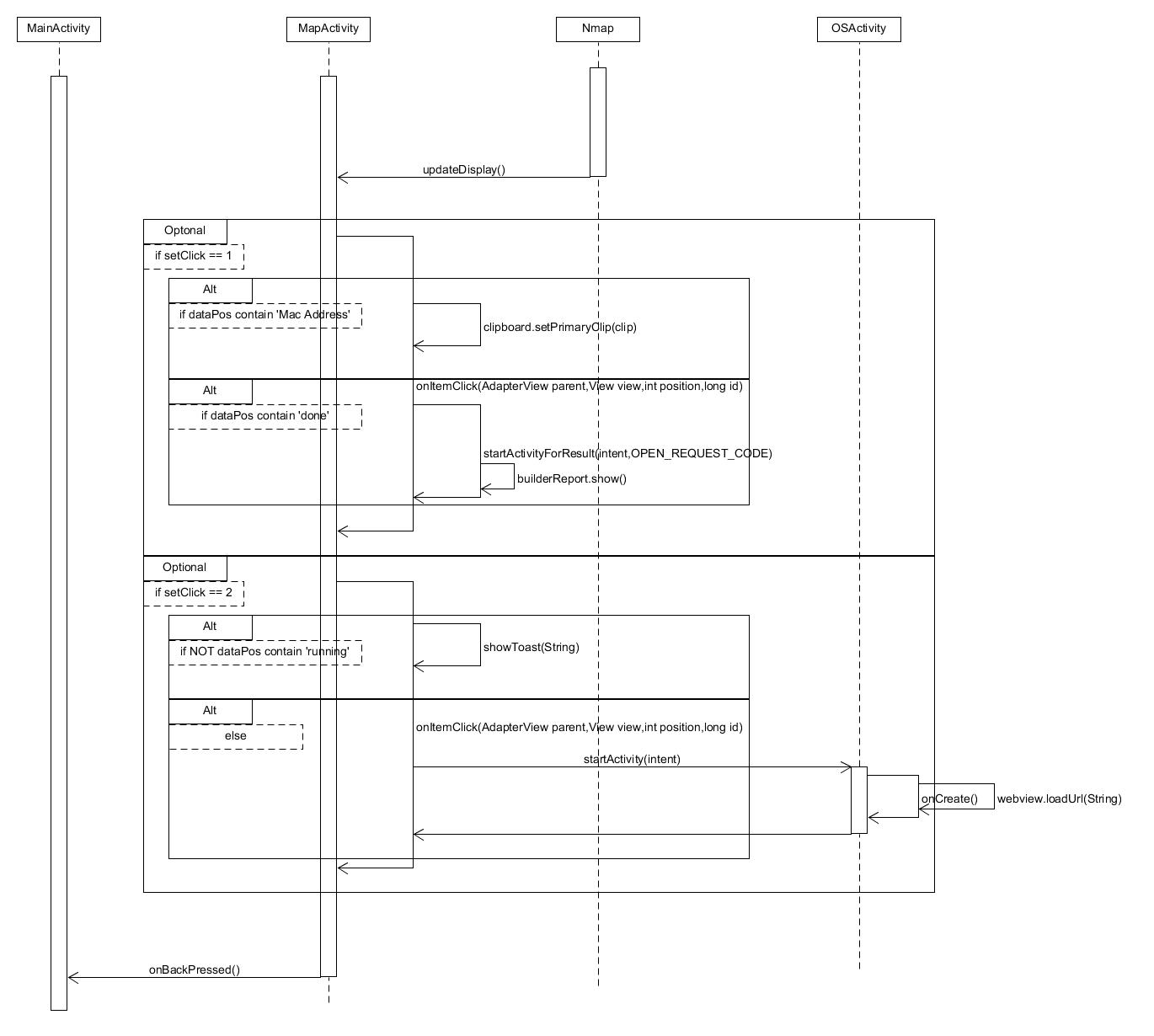
[UC-3.1/3.2] Map\_Start



[UC-3.3] Map\_Stop



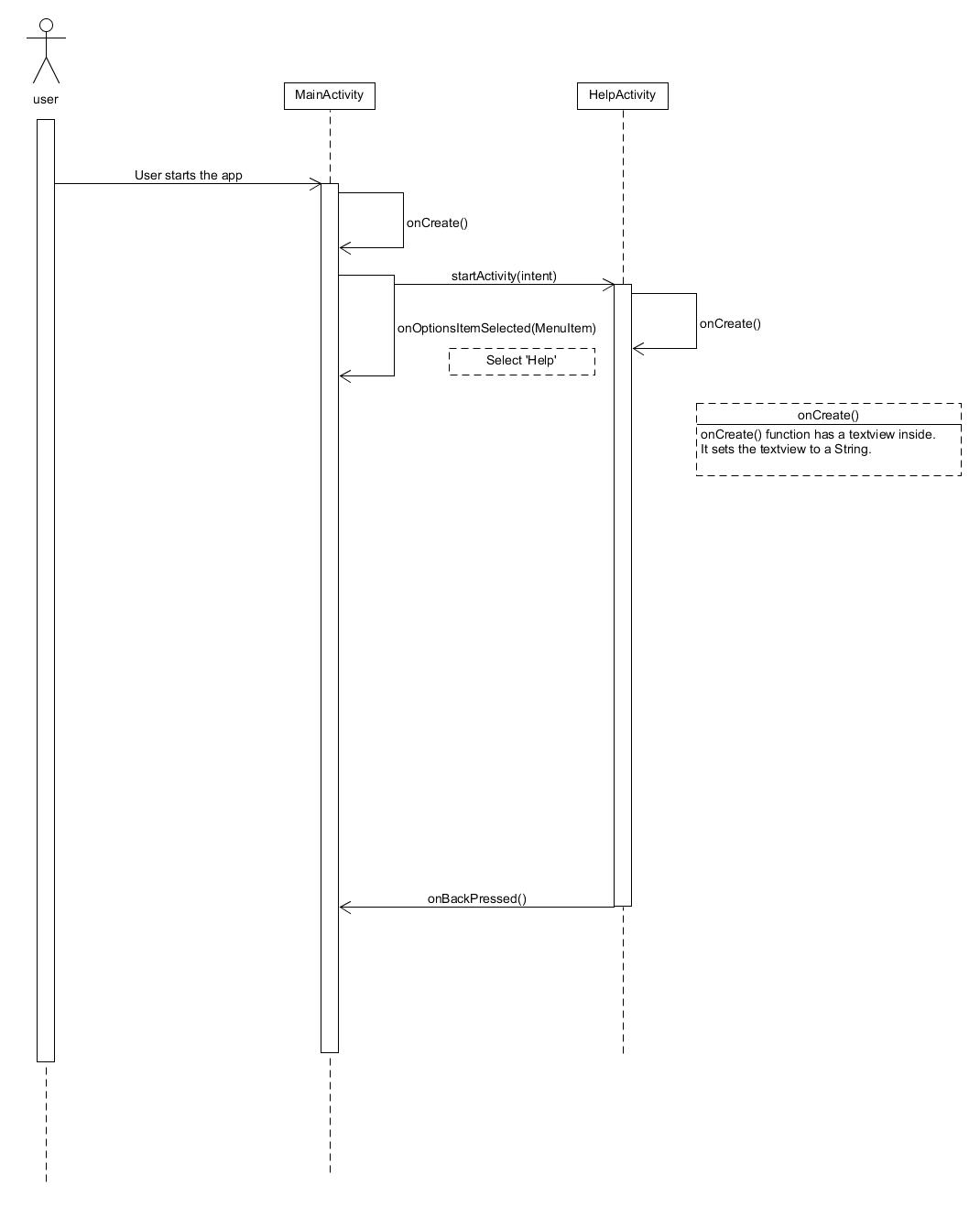
[UC-3.1.2/3.2.1] Map\_Feature



[UC-2] Check\_Resource



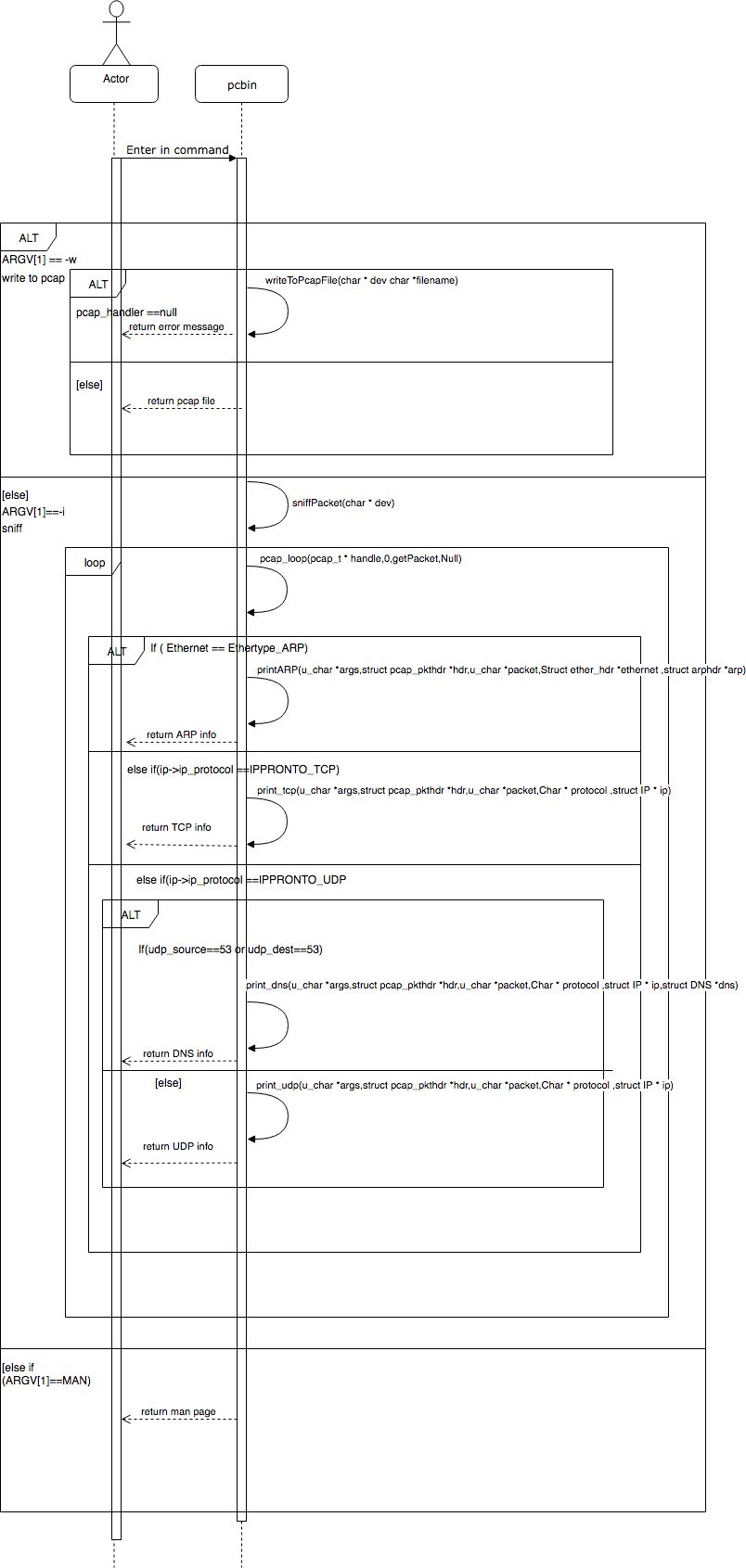
[UC-5] Help



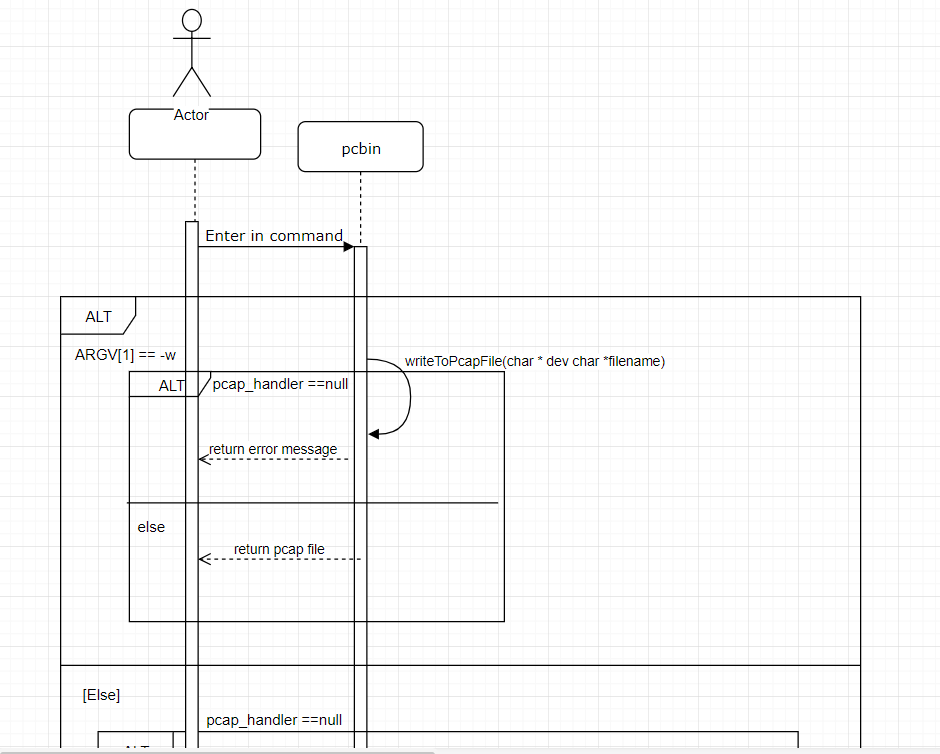
[UC-2] Check\_Resource

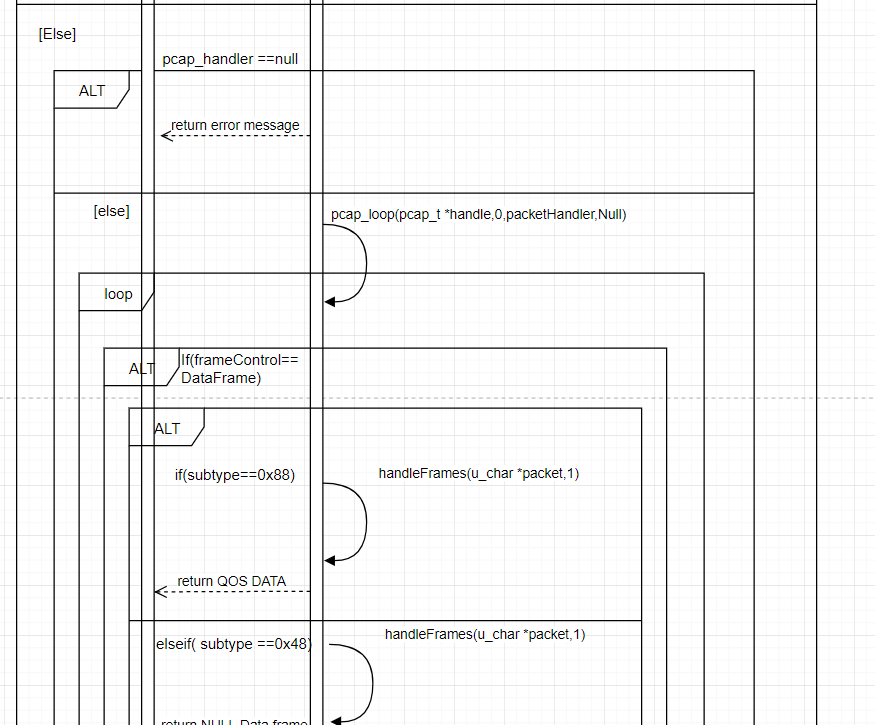


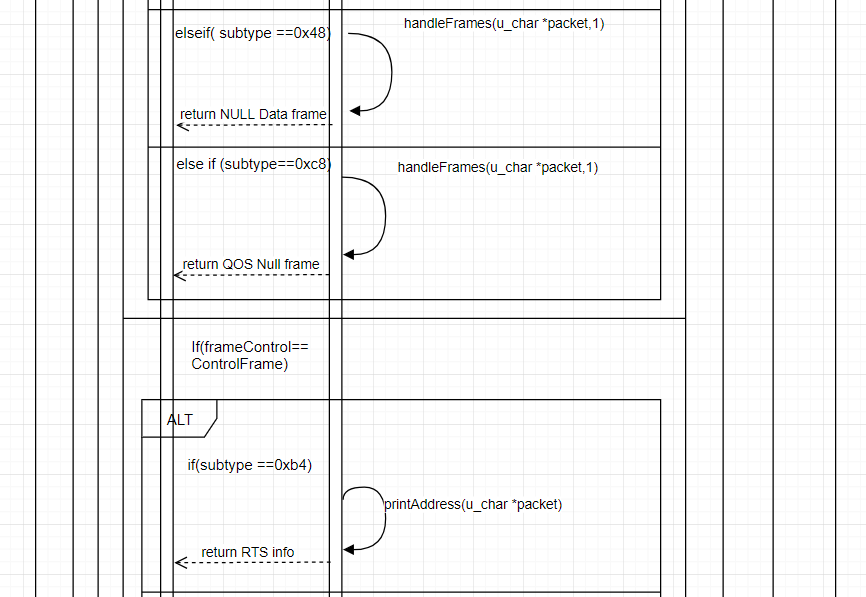
[UC-1.4]Pcbin binary Sequence Diagram:

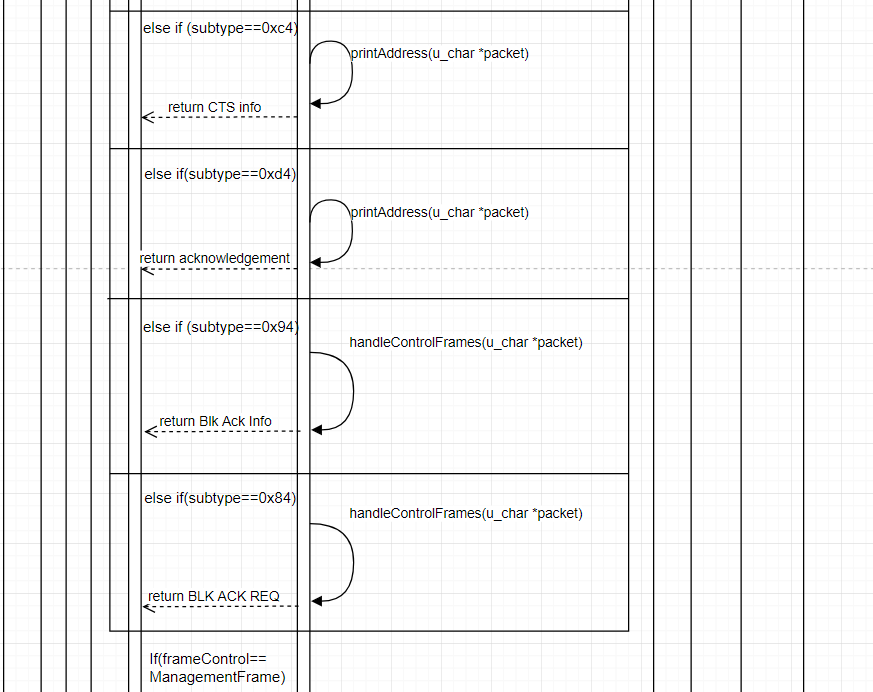


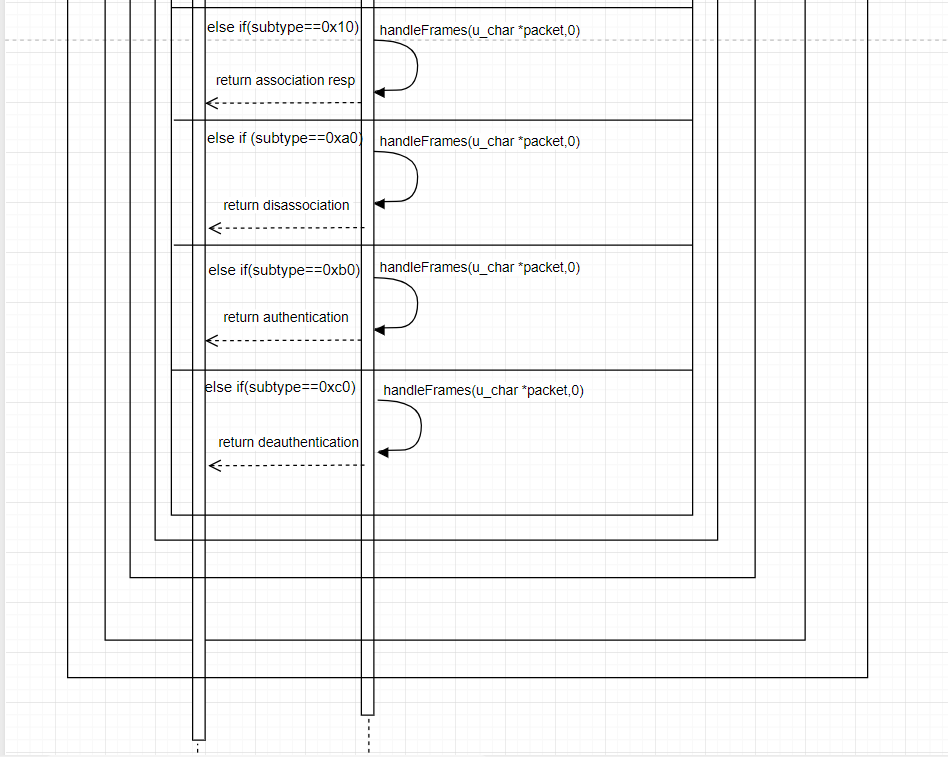
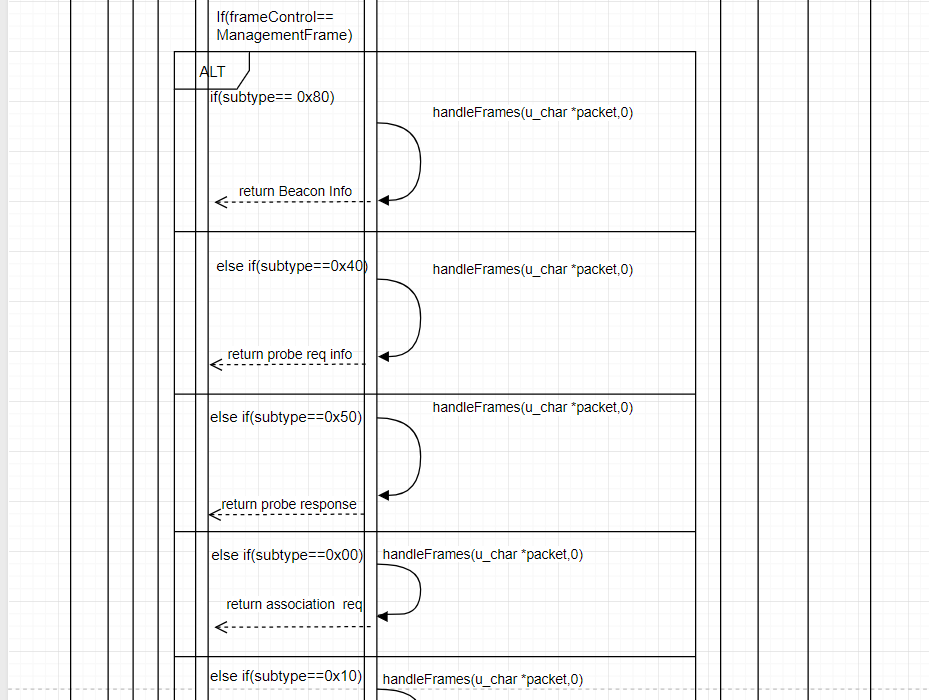
[UC-1.3]Pcmon binary Sequence Diagram:











## 5. Test Plan Design

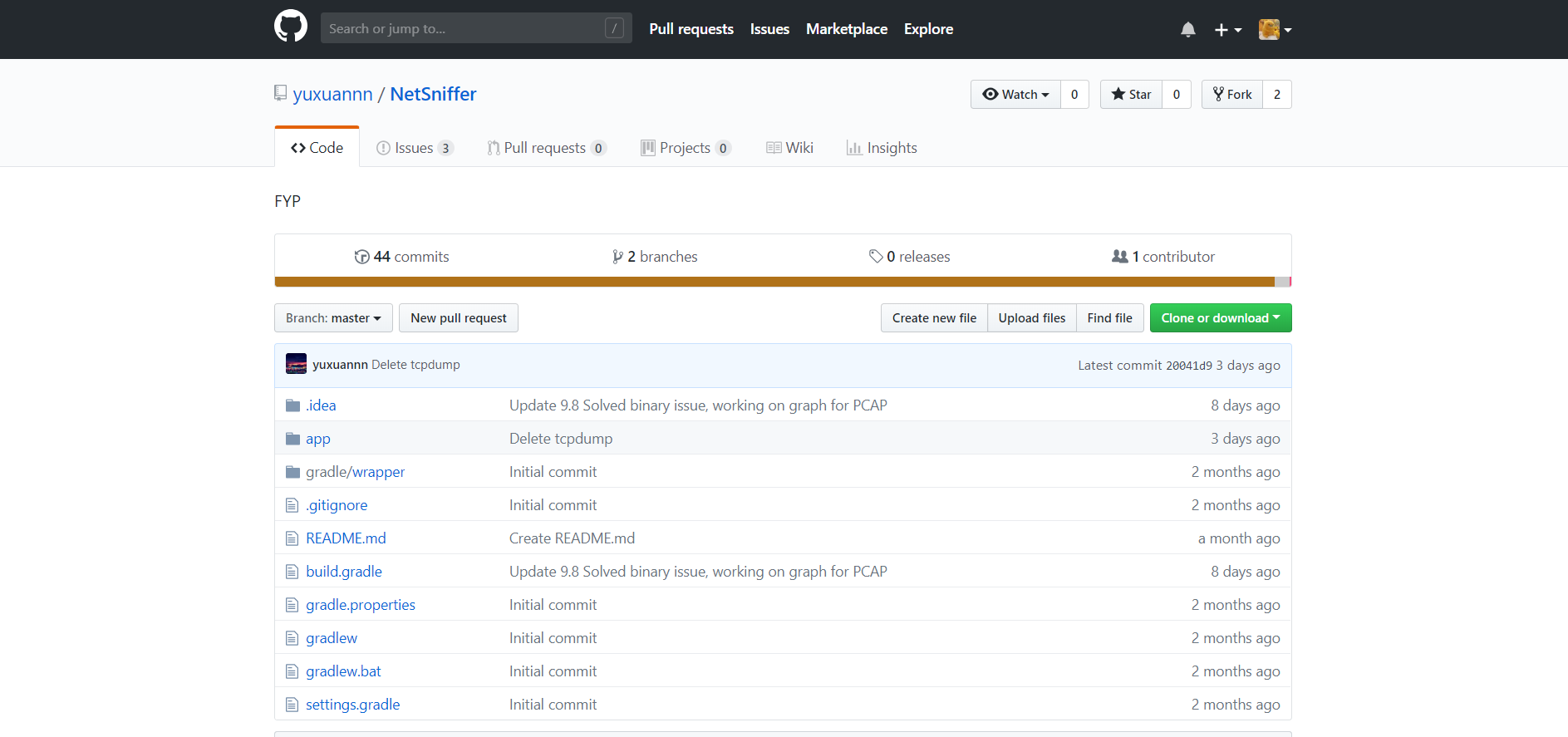
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Plan for :  Android Network Sniffer - Being able to capture/sniff packets | | | | |
| Test Case ID: | Description | Expected Outcome | Participating Program's Class & Method to be tested | Actual Outcome & If Issue was Solved |
| ANS-1 (Sniff) [UC-1.3] | To test if the Monitor mode(pcmon binary) packet sniffing works, being able to capture and see packets. Start Sniffing/Capture functions. | System should be able to receive and display packets to the display screen. The screen should show information relating to packets. | Class Name :  SnifferActivity, Sniffer.  Method Name :  void Sniffer.start() | At first, nothing was being output. Found the issue with threads, processes, update to ui.  Solved. |
| ANS-2 (Sniff) [UC-2] | To Test if the “Get Resource” button will trigger any errors if pressed multiple times, be error free. | System should not crash or have any errors if the button is pressed multiple times. | Class Name :  MainActivity  Method Name :  void getRes() | Pressing many times would “queue” the action up, resulting in efficiency issues. Now checks if has resources first.  Solved. |
| ANS-3 (Sniff) [UC-1.4] | To test if Promiscuous Mode(pcbin binary) packet sniffing works, see if can capture and see packets. | System should be able to receive and display packets to the display screen. The screen should show information relating to packets. | Class Name :  SnifferActivity, Sniffer.  Method Name :  void Sniffer.startProm() | Works. |
| ANS-4 (Sniff) [UC-1.3.1] | To test if sniffing of packets both Monitor and Promiscuous are able to create pcap files after sniffing is stopped. | System should have a pcap file created in its storage, at “sdcard/download” directory. | Class Name :  SnifferActivity, Sniffer.  Method Name :  void Sniffer.start() & void Sniffer.startProm() | Works. |
| ANS-5 (Sniff) [UC-1.1] | To test if sniffing filter works properly, be able to show filter only related results. | System should filter results based on keyword entered in the text and display those to screen. | Class Name :  SnifferActivity, Sniffer.  Method Name :  void Sniffer.start() & void Sniffer.startProm() | Was case-sensitive.  Solved. |
| ANS-6 (Map) [UC-3.1] | To test if Map Network works properly and displays to the screen, be able to show devices connected to network. | System should be able to generate the devices connected to the network and show them to the display screen. | Class Name :  MapActivity, Nmap  Method Name :  void Nmap.start(String) | Did not work, nothing displayed to screen. Figured out needed nmapservices to run.  Solved. |
| ANS-7 (Map) [UC-3.1.2] | To test if “done” choose file doesn’t crash when trying to generate report, being able to generate report. | System should be able to generate a report displayed to the screen showing Mac Addresses. | Class Name :  MapActivity, Nmap  Method Name :  void updateDisplay(final String, int mode, final Context), void onItemClick( AdapterView, View, int, long ) | Report generated had weird output.  Solved. |
| ANS-8 (Map) [UC-3.1] | To test if “Copy Mac Address” works properly, be able to copy address to clipboard. | System should be able to copy the Mac Address properly and be able to paste elsewhere. | Class Name :  MapActivity,Nmap  Method Name :  updateDisplay(final String, int mode, final Context), onItemClick( AdapterView, View, int, long ) | Works. |
| ANS-9 (Map) [UC-3.2] | To test if Map OS works properly, being able to show OS of device. | System should be able to obtain the OS of the device and display to screen. | Class Name :  MapActivity, Nmap  Method Name :  void Nmap.start(String) | Works. |
| ANS-10 (Map) [UC-3.2.1] | To test if Map OS can properly launch OS vulnerability search, be able to show OS vulnerability to screen. | System should be able to launch vulnerability list depending on OS received with no error. | Class Name :  Map Activity, Nmap  Method Name :  updateDisplay(final String, int mode, final Context), onItemClick( AdapterView, View, int, long ), webView(String) | Crashed when didn’t match “Running”, did not launch webpage related to OS vulnerabilities.  Solved. |
| ANS-11 (Binary) [UC-1.4] | To test that binary Pcbin can get IP packet info properly. | System be able to receive IP packets smoothly. | Class Name :  Pcbin.c  Method Name :  void sniffPacket(char\*)  void getPacket(u\_char \*, const struct pcap\_pkthdr \*, const u\_char\*)  void print\_IP(const struct Ip \*ip,char\*) | Works. |
| ANS-12 (Binary) [UC-1.4] | To test that binary Pcbin can get TCP packet info properly. | System be able to receive TCP packets smoothly. | Class Name :  Pcbin.c  Method Name :  void sniffPacket(char\*)  void getPacket(u\_char \*, const struct pcap\_pkthdr \*, const u\_char\*)  void print\_tcp(u\_char\* , const struct pcap\_pkthdr\* , const u\_char \*, char \*, const struct Ip \*) | Works. |
| ANS-13 (Binary) [UC-1.4] | To test that binary Pcbin can get UDP packet info properly. | System be able to receive UDP packets smoothly. | Class Name :  Pcbin.c  Method Name :  void sniffPacket(char\*)  void getPacket(u\_char \*, const struct pcap\_pkthdr \*, const u\_char\*)  void print\_udp(u\_char\* , const struct pcap\_pkthdr\* , const u\_char \*, char \*, const struct Ip \*) | Works. |
| ANS-14 (Binary) [UC-1.4] | To test that binary Pcbin can get DNS packet info properly. | System be able to receive DNS packets smoothly. | Class Name :  Pcbin.c  Method Name :  void sniffPacket(char\*)  void getPacket(u\_char \*, const struct pcap\_pkthdr \*, const u\_char\*)  void print\_IP(u\_char\* , const struct pcap\_pkthdr\* , const u\_char\*, char\*, const struct Ip\*,const struct Dns\*) | Weird Output, wrong packet structure, compare .pcap file in wireshark, analyse structure.  Solved. |
| ANS-15 (Binary) [UC-1.4] | To test that binary Pcbin can get ARP packet info properly. | System be able to receive ARP packets smoothly. | Class Name :  Pcbin.c  Method Name :  void sniffPacket(char\*)  void getPacket(u\_char \*, const struct pcap\_pkthdr \*, const u\_char\*)  void printARP(u\_char\*, const struct pcap\_pkthdr\*, const u\_char\*, const struct ether\_header\* ,const struct Arphdr\*) | Works. |
| ANS-16 (Binary) [UC-1.3] | To test that binary Pcmon can output correct packet info. | System be able to receive 802.11 packets smoothly. | Class Name :  Pcmon.c  Method Name :  void pcapHandler(u\_char\* ,const struct pcap\_pkthdr\*, const u\_char\*)  void printAddress(const u\_char\*)  void printHexAsciiValueOfPayload(const u\_char\*)  void handleFrames(const u\_char\*, int )  void handleControlFrames(const u\_char\*) | Weird output, figured out contained radiotap header, analyze in wireshark.  Solved. |
| ANS-17 (Binary) [UC-1.3] | To test that Pcmon is reading in radiotap headers correctly. | System be able to interpret radiotap headers correctly, resulting in correct packet info. | Class Name :  Pcmon.c  Method Name :  printHexAsciiValueOfPayload(const u\_char\*)  void writeToPcapFile(char\*, char\*) | Weird Output, Figured out radiotap header length, and why different length.  Solved. |
| ANS-18 (Analyze) [UC-4.1.3] | To test that graph is generated correctly based on .pcap file. | System be able to read pcap file and generate graph smoothly to screen. | Class Name :  GraphActivity, AddressPair, GraphView  Method Name :  void onActivityResult(int, int, Intent )  void onDraw(Canvas)  float getMax()  float getMin()  float getAvg()  float[] calculateYCoord(float[], float[]) | Graph first generation does not show up. Subsequent graphs shows up.  Solved. |
| ANS-19 (Analyze) [UC-4] | To test that the choosing of files other than pcap files will not crash app. | System be able to not crash or any errors upon choosing non-pcap files. | Class Name :  GraphActivity  Method Name :  void onActivityResult(int, int, Intent) | Works. |
| ANS-20 (Analyze) [UC-4] | To test that pcap analysis can both read in Monitor and Promiscuous pcap files. | System be able to detect and use appropriate code based on file content. | Class Name :  GraphActivity, AddressPair  Method Name :  void onActivityResult(int, int, Intent) | Works. |

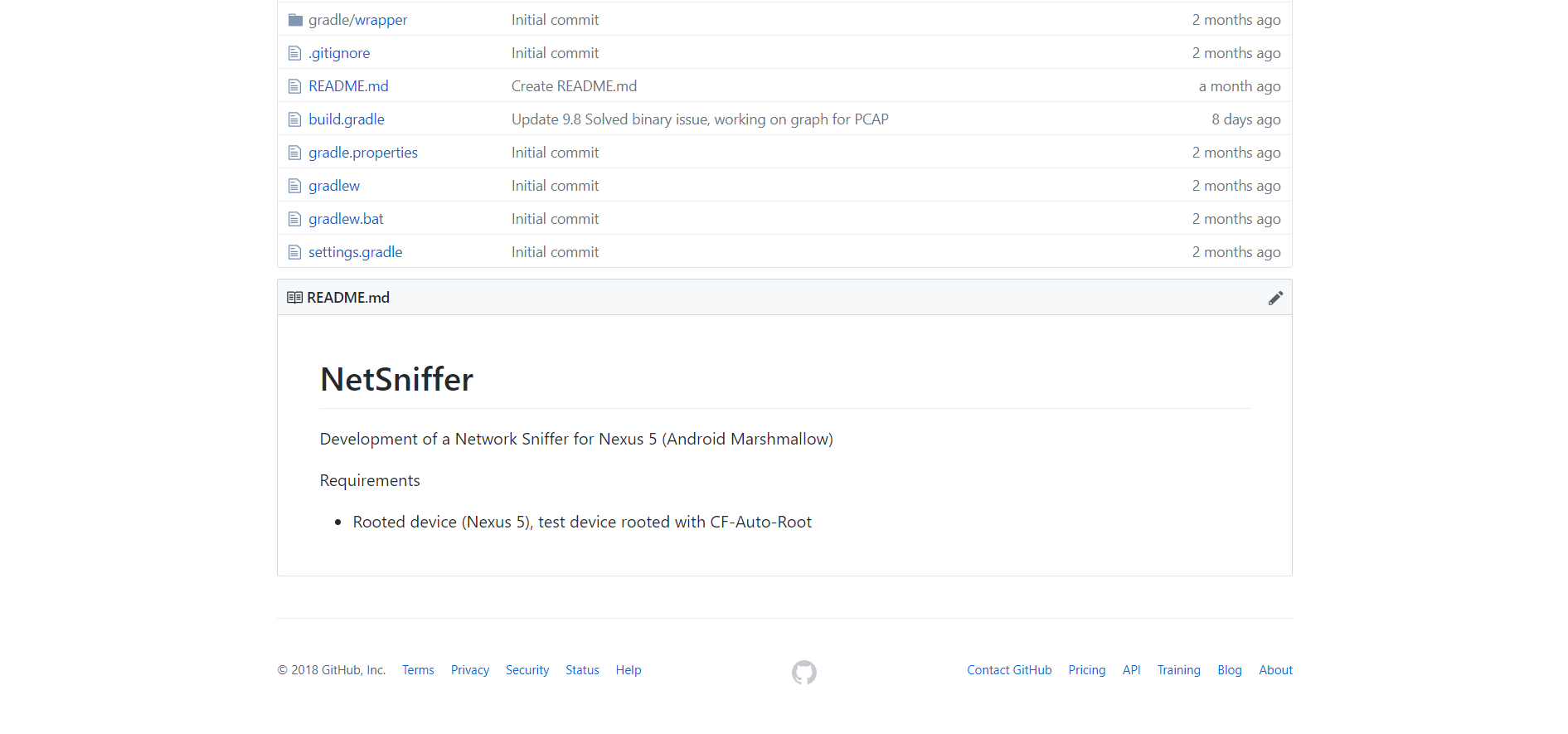
## 6. Evidence of VCS (Version Control Software)

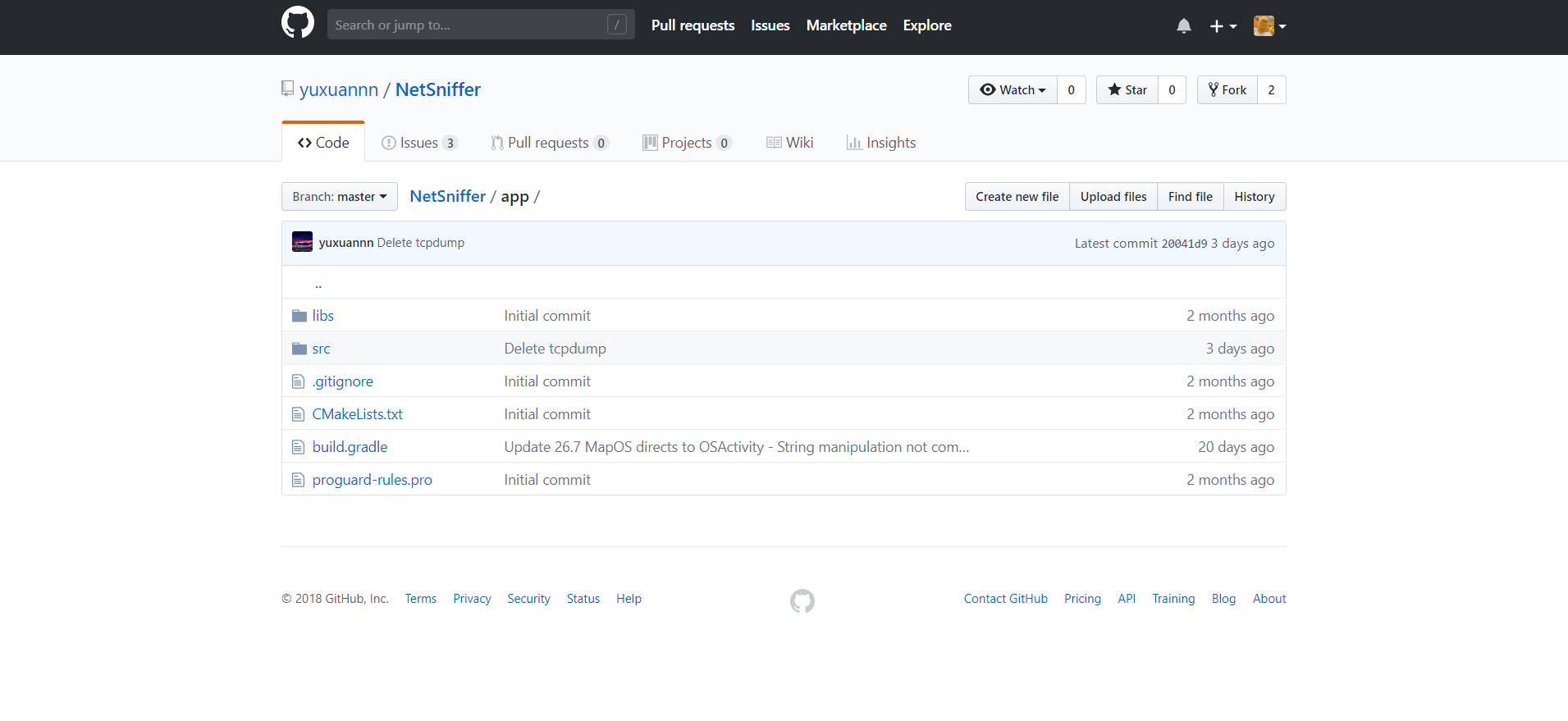
Version Control Software used : Github

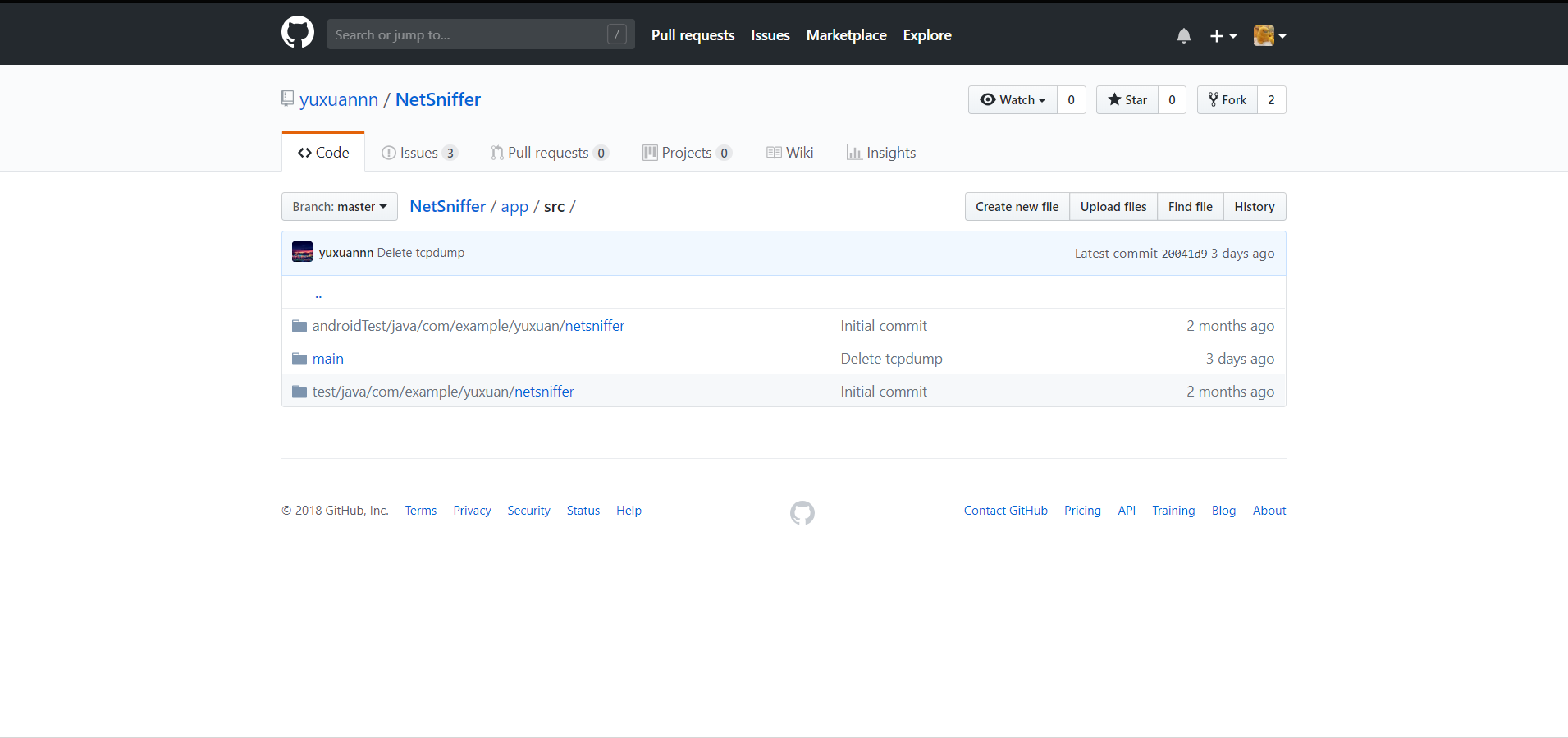
***Contents in the VCS Root Folder containing all Repository Project Files***

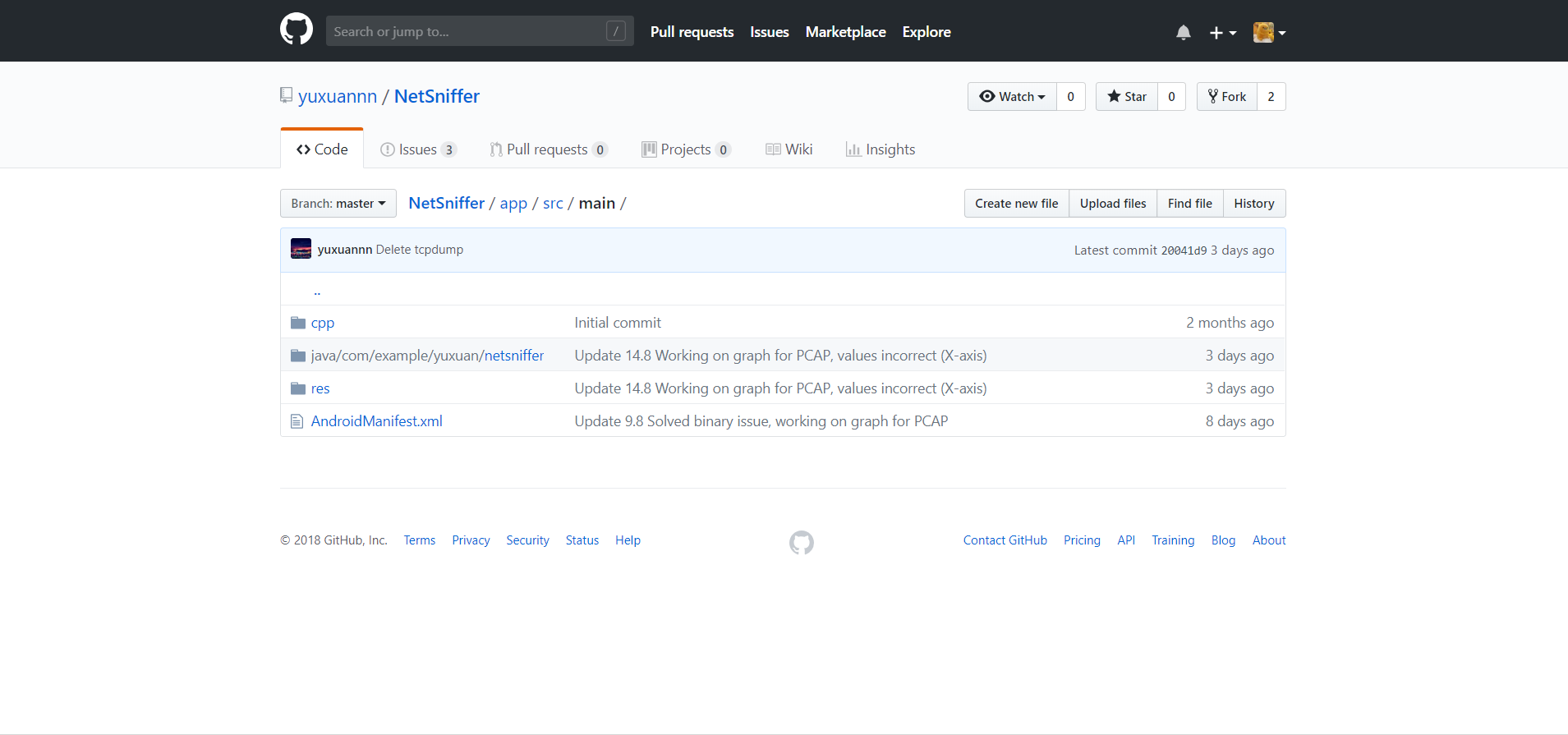
Netsniffer Repository:

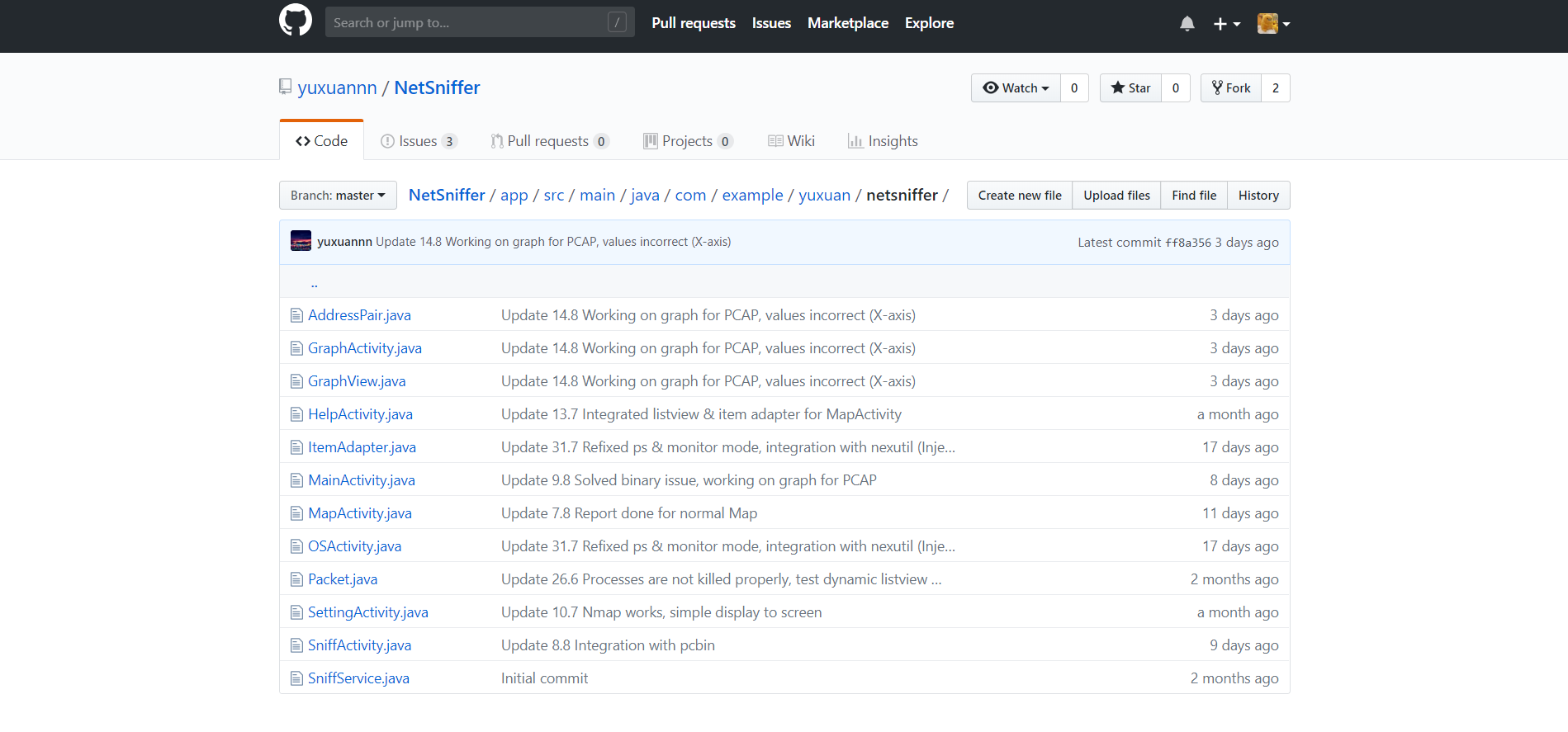




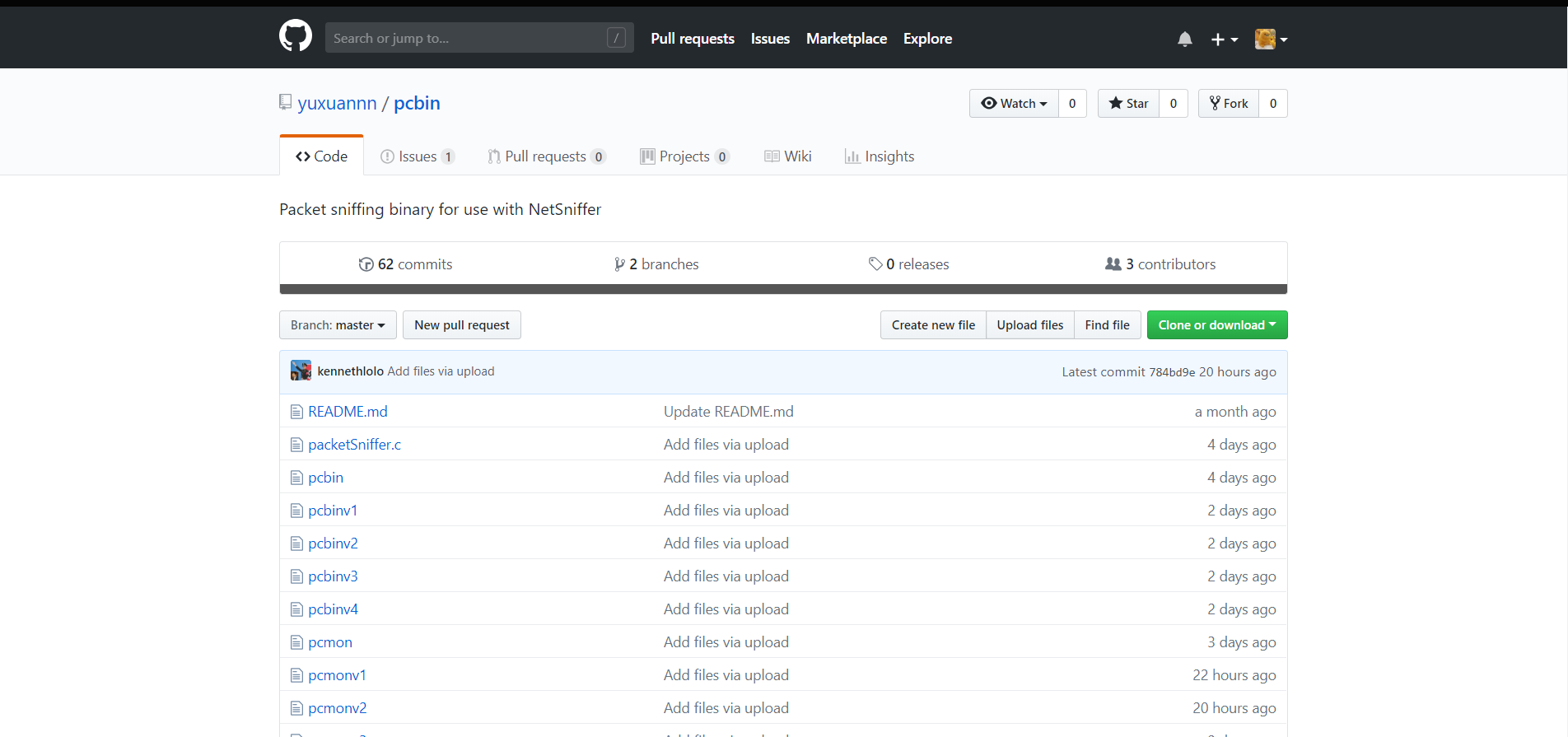




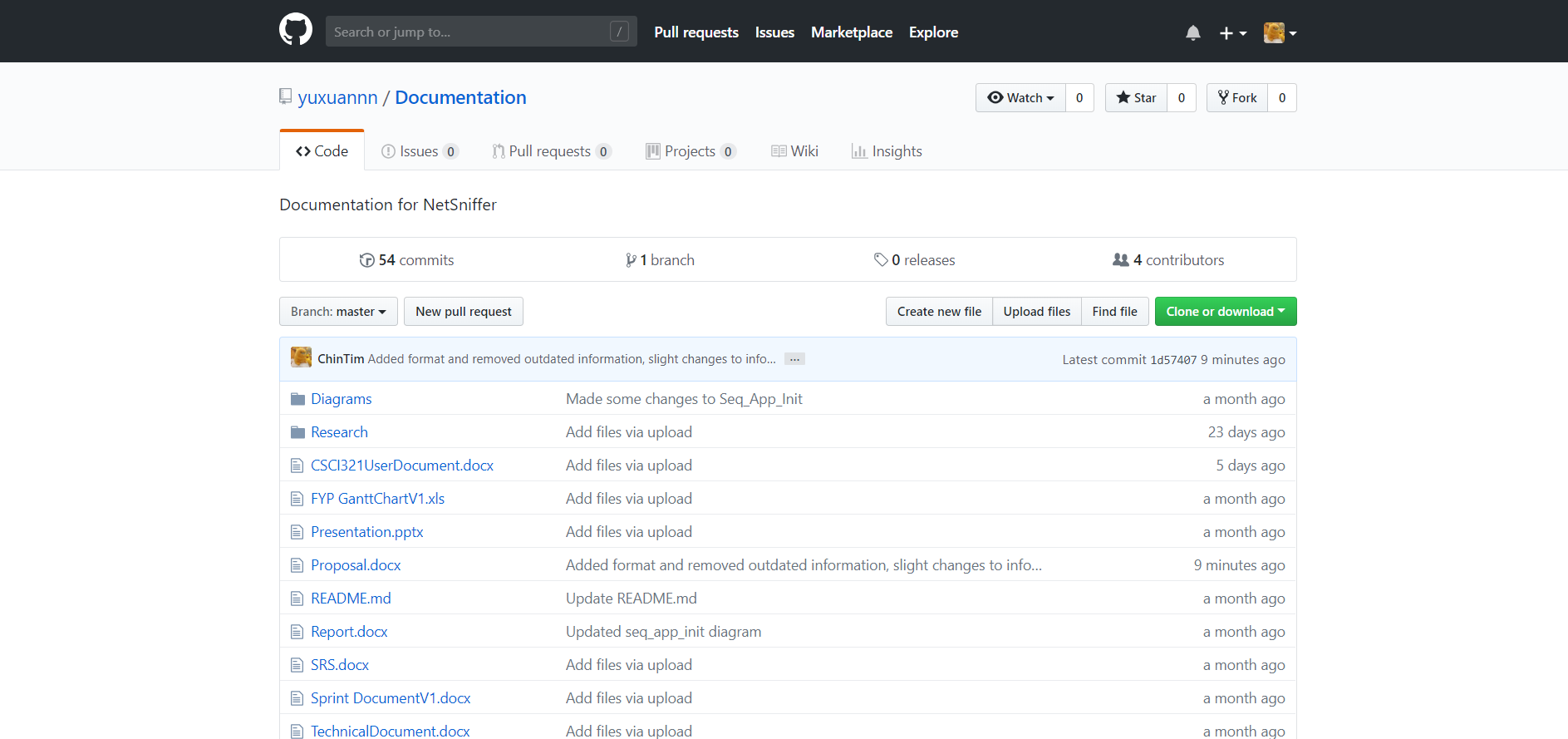




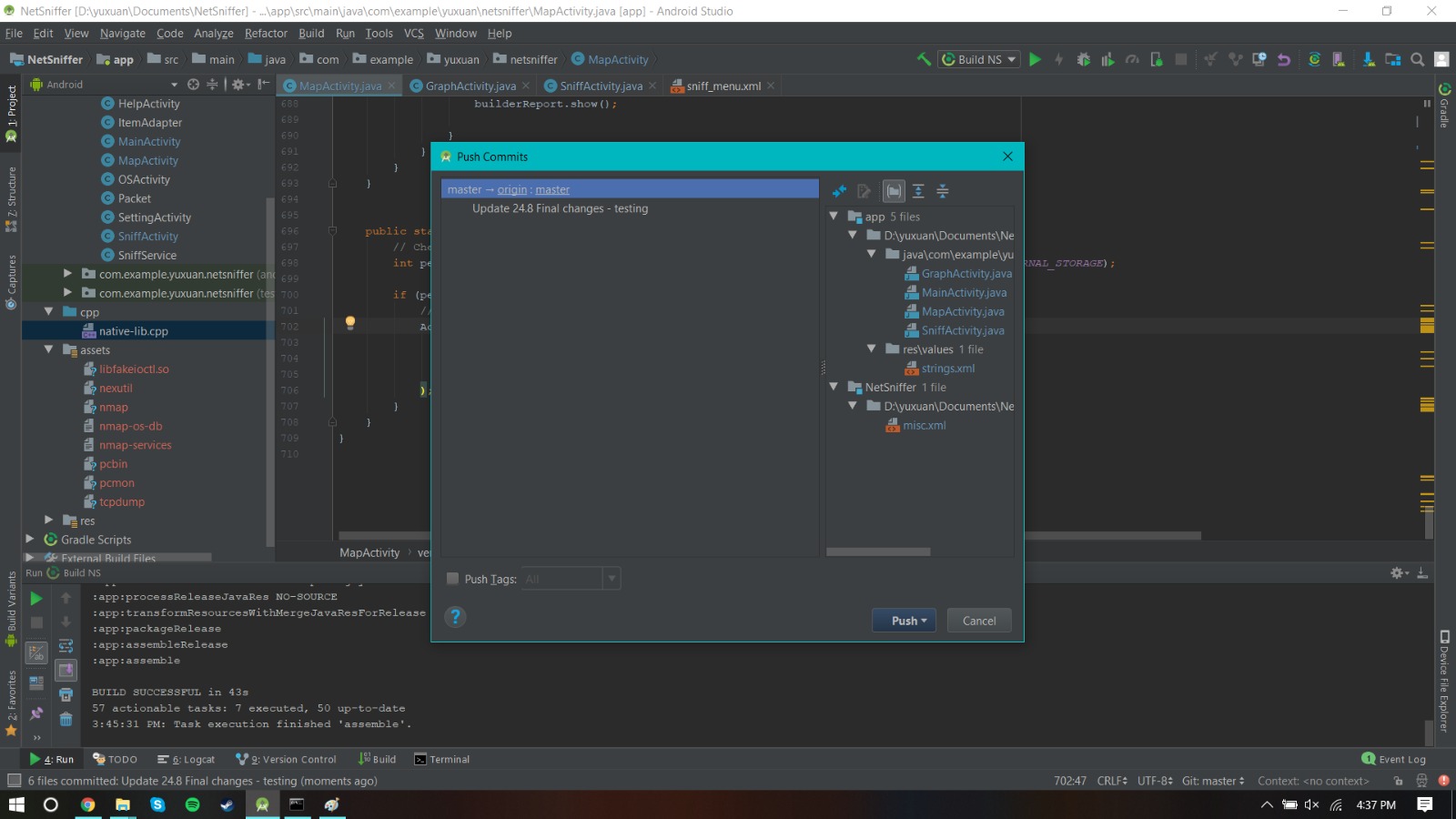
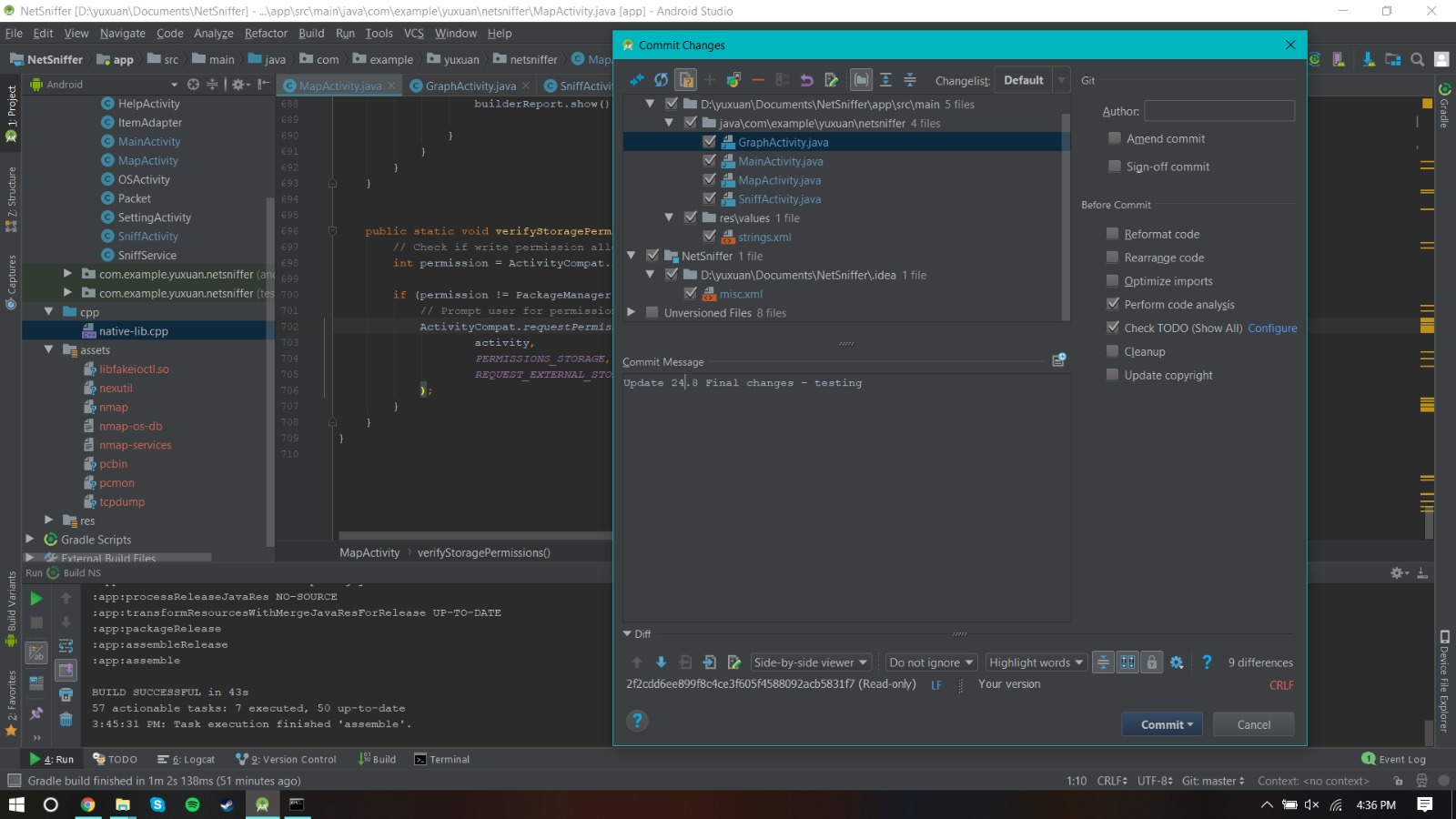
Pcbin Repository:



Documentation Repository:

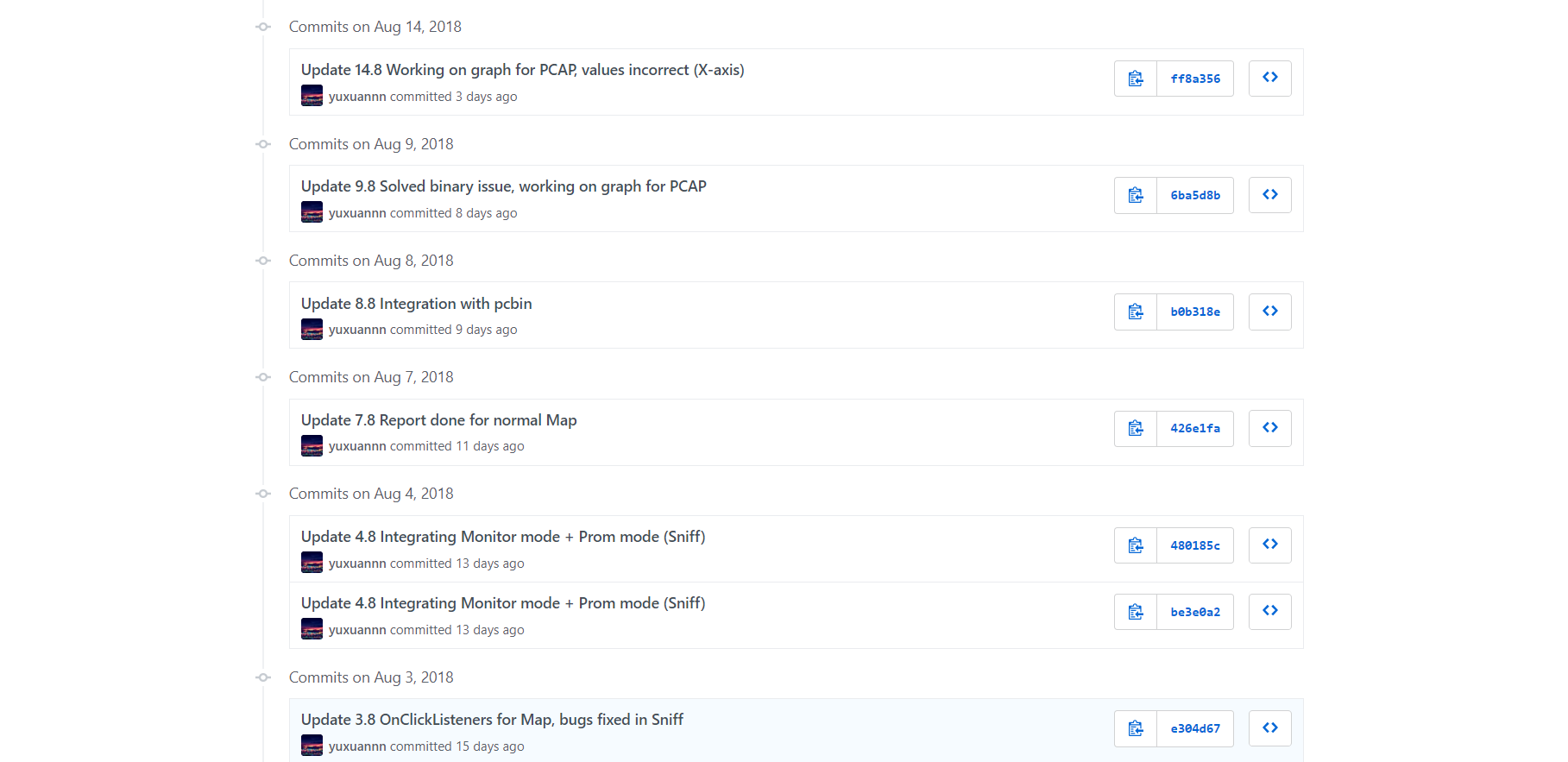


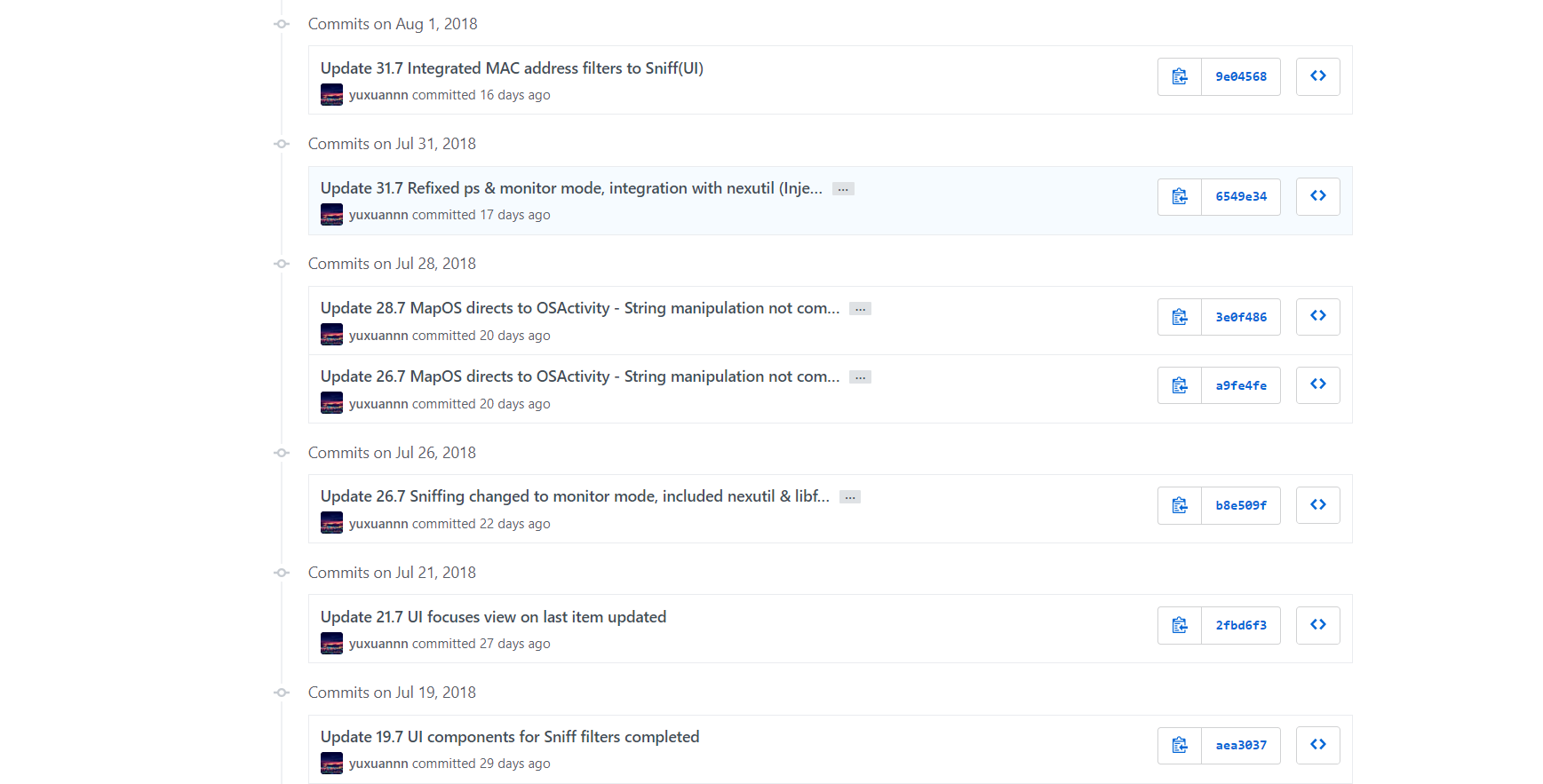
***Using VCS to check-in source files***



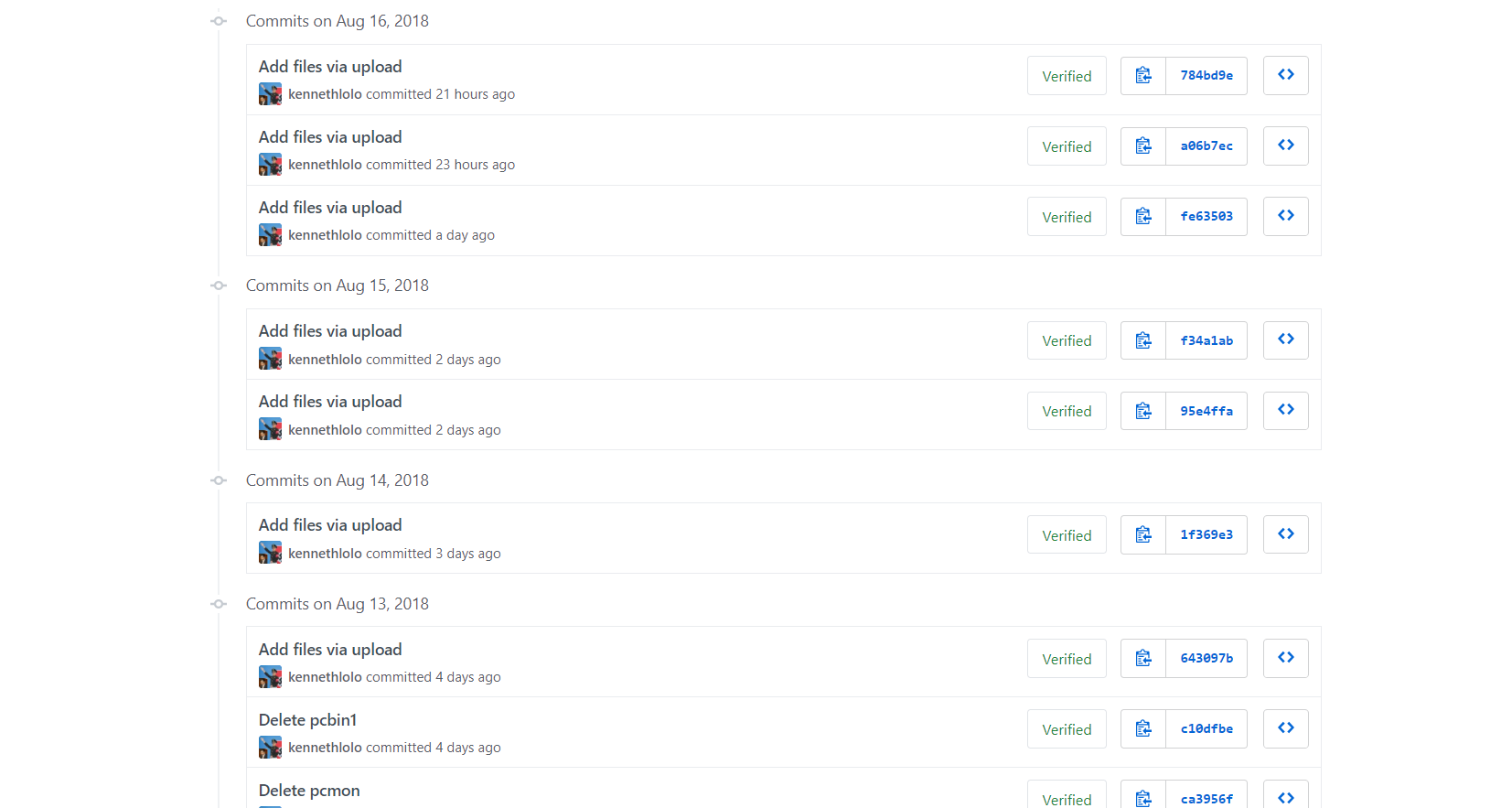
***Using VCS to display the change history / log***

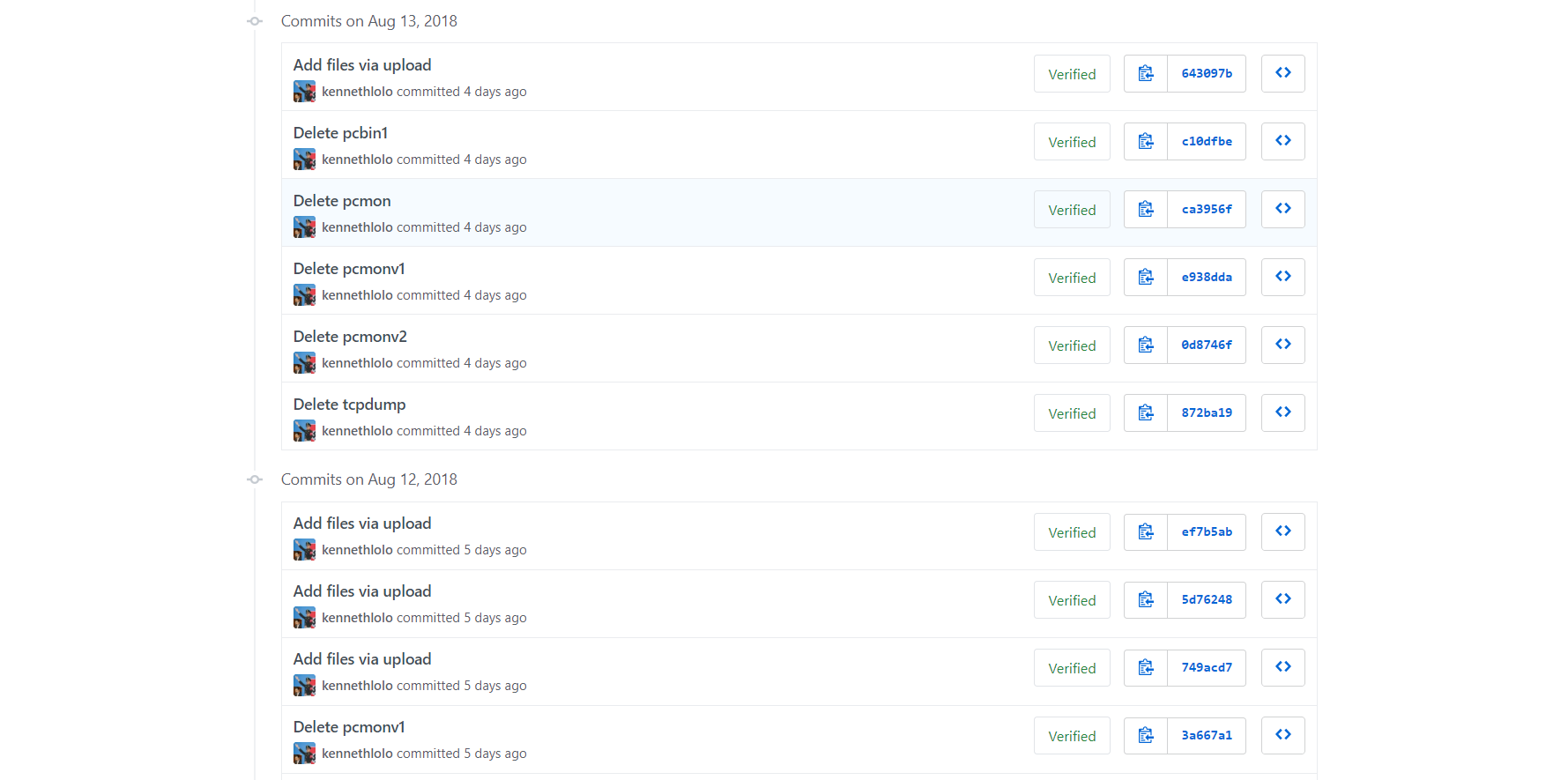
NetSniffer History Log (Commits):



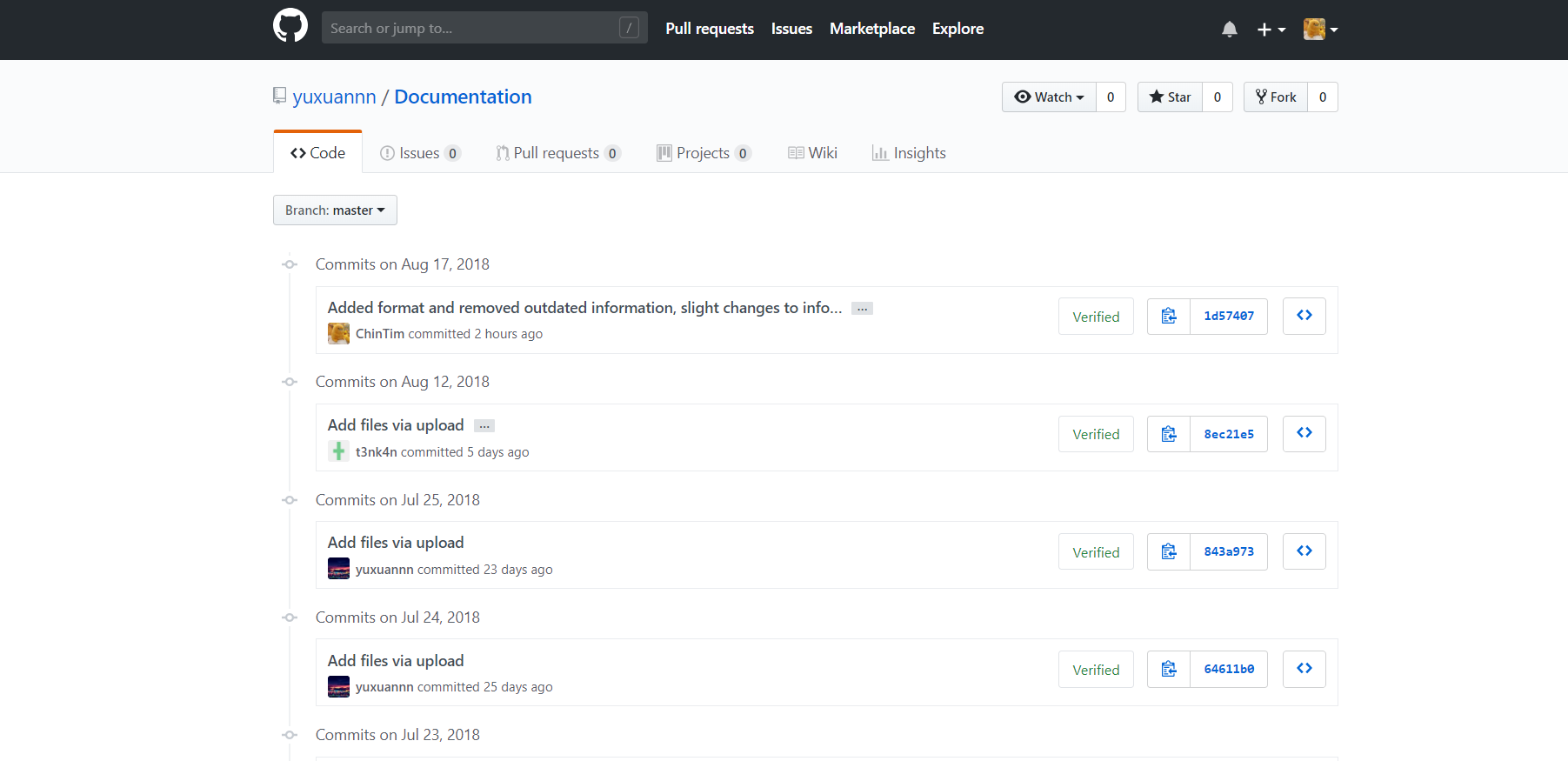


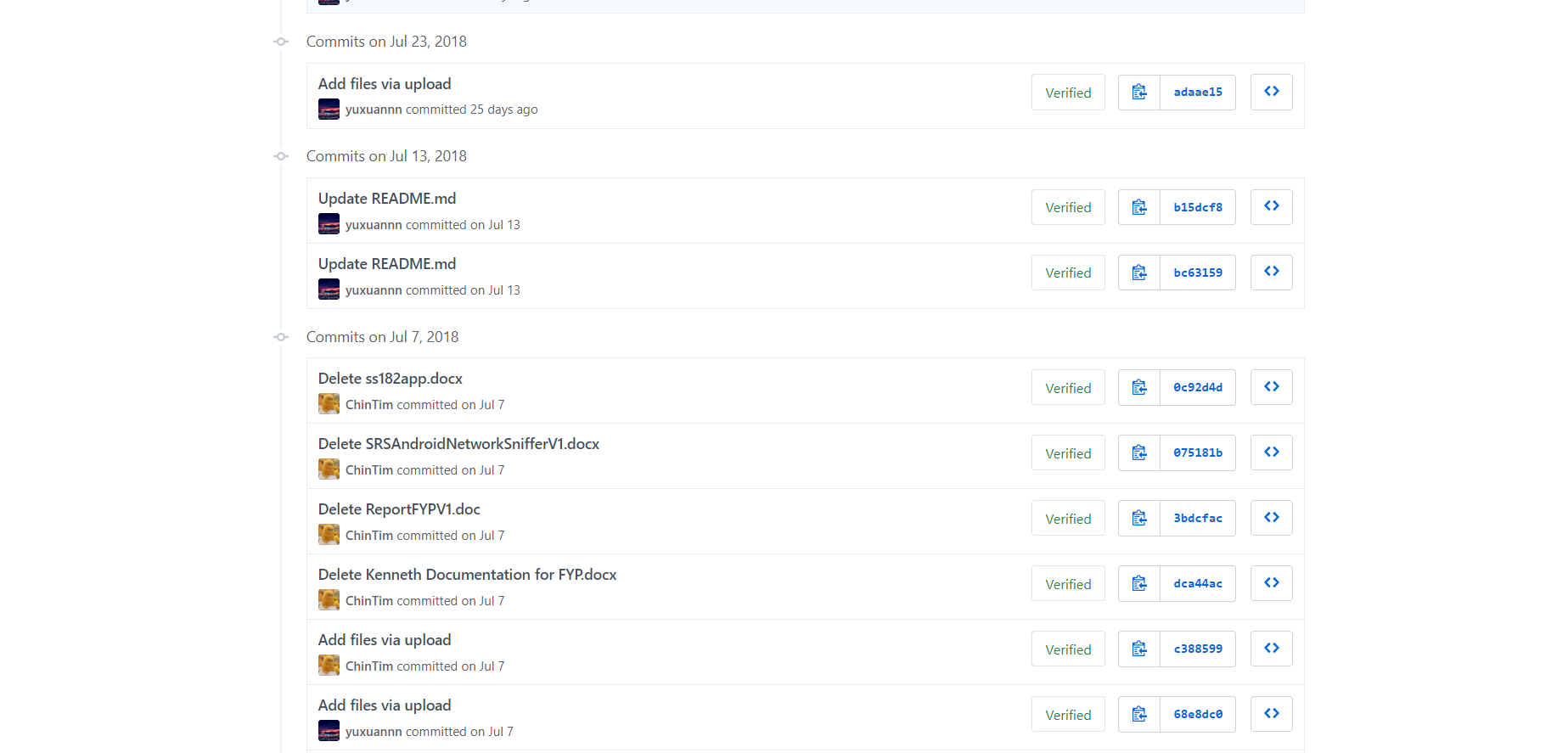
Pcbin History Log (Commits):





Documents History Log (Commits):





END OF TECHNICAL MANUAL