|  |  |  |
| --- | --- | --- |
| IT3527 InfoSecurity & Forensic Case Study AY2013/2014 Project Report | October 22  2013 | |
| MODULE GROUP IT3527-01 PROJECT TEAM Kok Jian Hui Caleb (Leader) Sim Renhe Joshua Alvin Lee Yong Teck | |  |

Table of Contents

[Document Control iii](#_Toc379579059)

[1. Introduction 1](#_Toc379579060)

[2. Project Proposal 1](#_Toc379579061)

[2.1 Literature Review and Feasibility Study 1](#_Toc379579062)

[2.1.1 iPhone Forensic 1](#_Toc379579063)

[2.1.2 Android Forensic 3](#_Toc379579064)

[2.1.3 Splunk 6.0 5](#_Toc379579065)

[2.2 Business Case and Security Considerations 8](#_Toc379579066)

[2.2.1 Business Case 8](#_Toc379579067)

[2.2.2 Security Considerations 8](#_Toc379579068)

[2.3 Architecture Diagram 9](#_Toc379579069)

[2.4 Team Organization 10](#_Toc379579070)

[2.5 Task Allocation 10](#_Toc379579071)

[2.6 Project Schedule 11](#_Toc379579072)

[2.7 Resources 12](#_Toc379579073)

[2.7.1 Hardware 12](#_Toc379579074)

[2.7.2 Software 12](#_Toc379579075)

[3. Requirements Specifications (Indicate individual contributions) 12](#_Toc379579076)

[3.1 Functional and Non-Functional Requirements 12](#_Toc379579077)

[3.2 Use Case Diagram 13](#_Toc379579078)

[4. Design Specifications (Indicate individual contributions) 14](#_Toc379579079)

[4.1 System Architecture and System Components 14](#_Toc379579080)

[4.2 Sequence Diagrams 15](#_Toc379579081)

[4.3 Design Class Diagrams 16](#_Toc379579082)

[4.4 Interface Specifications 16](#_Toc379579083)

[5. Implementation Updates (Indicate individual contributions) 17](#_Toc379579084)

[5.1 Initial Prototype 17](#_Toc379579085)

[5.2 Implementation Update 1 22](#_Toc379579086)

[5.3 Implementation Update 2 40](#_Toc379579087)

[5.4 Final System 76](#_Toc379579088)

[6. Test Plan and Test Report (Indicate individual contributions) 77](#_Toc379579089)

[6.1 Test Setup 77](#_Toc379579090)

[6.2 Test Cases 77](#_Toc379579091)

[6.3 Summary of Test Results and Corrective Actions 82](#_Toc379579092)

[6.4 Counter Test Cases (SOCKS) 83](#_Toc379579093)

[7. Conclusion 88](#_Toc379579094)

[7.1 Accomplishments 88](#_Toc379579095)

[7.2 Evaluation and Future Enhancements 91](#_Toc379579096)

[7.3 Lessons Learnt 92](#_Toc379579097)

# Document Control

|  |  |
| --- | --- |
| **Version Number (0.1)** | **Project Deliverables** |
| 0.1.0 | Project Proposal  Project Plan(PP) |
| 0.2.0 | Functional and non-functional requirements  Use case and sequence diagrams  System architecture and system components  System prototype demo |
| 0.3.0 | System implementation  Implementation progress update  System presentation |
| 0.4.0 | System implementation and integration  Test plan/test cases  Integrated system demo |
| 0.5.0 | Integration and system testing  Error rectification and regression testing  Test results presentation  Final project report with test report |
| 0.6.0 | Conclusion  Accomplishments  Evaluations and Future Enhancements  Lessons Learnt |
| 1.0.0 | Final Project Report  Counter Test Cases (SOCKS) |

# 1. Introduction

Mobile device forensics is the branch of digital forensics that relates to the extraction of digital evidence from a mobile device. Mobile phones, like personal computers, leave behind a trail of digital data which is often not visible for deletion. This trail of digital data is extracted by the mobile device forensic investigator to serve as digital evidence. The digital evidence is then used to aid legal teams or police detectives build the case against the criminal.

So why is mobile device forensics becoming increasingly relevant now? In Asia-Pacific alone, the smartphone market is expected to double to 200 million by 2016, with the Google Android platform leading in market share. This explosion in smartphone ownership has created great demand in the area of mobile device forensics.

In today’s age of information, a smartphone is quickly becoming an essential asset. Users now surf the internet, access email and edit important documents on their smartphones; phone calls and SMS are no longer the primary function of a smartphone. As such, within the smartphone lies a trove of forensically valuable data that will aid in the investigation process.

# 2. Project Proposal

## 2.1 Literature Review and Feasibility Study

An important message that can be taken from the Mobile Device Forensics white paper published by SANS Institute 2009 (<http://www.sans.org/reading-room/whitepapers/forensics/mobile-device-forensics-32888?show=mobile-device-forensics-32888&cat=forensics>) is that there are no free tools capable of analyzing every single smart phone on the market. Several vendors have surfaced in an attempt to fill the need for a common analysis platform for the many **different** phone manufacturers and models. In addition, the variety of forensic toolkits for mobile device is diverse. A considerable number of software tools and toolkits exist, but the range of devices over which they operate is typically **narrowed to distinct** **platforms** for a manufacturer’s product line, a family of operating systems, or a type of hardware architecture.

This therefore creates a need to conduct mobile device forensic using different operating system for a huge range of mobile smart phones. For literature review and feasibility study, our product can be split into three main categories namely, iPhone forensic, android forensic and Splunk 6.0

### 2.1.1 iPhone Forensic

iPhone Forensic is a process of data acquisition of iOS mobile operating system. Mobile devices that was using iOS operating system such as iPhone 4S (iOS 5.0), iPhone 4 (iOS 4.0), iPhone 3GS (iOS 3.0), and other iPhone’s models. Our project will be targeting on iPhone 4S and below models, as the newer generation of model such as iPhone 5, iPhone 5C, and iPhone 5S which does not have much resources on the web for us to research on.

#### Backups made by iTunes

iPhone Forensics can be performed on the backups made by iTunes or directly on the live device. This article (<http://resources.infosecinstitute.com/iphone-forensics/>) explains the technical procedure and the challenges involved in extracting data from the live iPhone. In this article, it aims to extract data and artifacts from an iPhone without tampering the data on the device. This aim is what we wanted to achieve for our project. However, due to the amount of tedious and deep technical procedures and knowledge, it serves as a great challenges to us too even if we got our required resources. **Feasibility: Low**

#### Library: libimobiledevice

One of the potential technologies that may be useful for our project will be **libimobiledevice.** Libimobiledevice is a cross-platform software protocol library and tools to communicate with iOS devices (such as iPhone, iPod Touch, iPad, and Apple TV devices) natively. It will allow other software to easily access the device's file system, retrieve information about the device and its internals, backup/restore the device, manage installed applications, and retrieve calendars/address book/notes and bookmarks. It aims to bring support for these devices to the Linux Desktop. It is tested to be able to run on Linux, Mac OS, as well as Windows. However, if it was unable to run on Windows, we will require VMware Workstation running with a Linux OS image in order to proceed with our project. As libimobiledevice is a library, there will be a need to update the applications that use libimobiledevice in order for it to become useful after installation of the library. However, this serves as a challenge to us as we will have to find an external software that is compatible to use the library, and also how to integrate both the external software and the library together to achieve our main goal – data acquisition. **Feasibility: Medium**

#### Oxygen Forensic Suite 2013

Another potential technology that may be useful for our project will be **Oxygen Forensic Suite 2013.** Oxygen Forensic Suite 2013 is a mobile device forensic software for logical analysis of mobile phone devices developed by Oxygen Software. It will be able to extract device information, contacts, calendar events, SMS messages, event logs, and files. For the recent changes with new patch released on 2nd October 2013, it is now able to support for Apple iPhone 5S and iPhone 5C, as a result support for Apple iOS 7.0.2. Being able to support for Apple iPhone 5S and iPhone 5C (latest release model), it will also supports for Apple iPhone 4S and below models as well, which will hence increase the feasibility level. We will have higher confidence in getting things done using Oxygen Forensic Suite as it was quite efficient for the user to get things done on data acquisition. The only downfall is that the Oxygen Forensic Suite only provides a trial version for the public. To solve this issue, we will need to have VMware images to install the Oxygen Forensic Suite and once the trial version is up, either revert back to the snapshot before installation, or use a different images to install the Oxygen Forensic Suite over again. **Feasibility: High**

In conclusion, iPhone Forensic can be a tedious and challenging process for our team to further research on to know more deeply. We would like to start on **libimobiledevice**, as it is a good approach for us to gain more knowledge and know what’s really happening during iPhone data acquisition. What’s more of our concern now is the resource of an iPhone for our team to proceed on with our project. Without an iPhone, we will not be able to perform iPhone Forensics for our project and hence, need to have an alternative or backup plan.

### 2.1.2 Android Forensic

Android Forensic is the recovery of digital evidence or data from an Android device under forensically sound conditions. Unlike iPhone which only have one operating system such as the iOS, there are many different variants of Android, specifically developed individually by different Android hardware vendors and custom Android operating system developers. Although the base of these variants is still the Android operating system, different variants of Android might exhibit different type of behavior on testing. Therefore for our product, we will be focusing more on the Samsung variant of the Android operating system as Samsung have the biggest user market for the Android platform. Our product might still work for the other variants of the Android operating system as they share a same android code base.

#### Android default backup

The default backup function of Android is to back up the phone’s data to the Google server. This backup data is encrypted in transit, accessible only when the user has an authenticated connection to Google and stored at Google data centers. This has the **least** feasibility as the data contained in the backup will be difficult to extract out and the examiner will not be able to obtain various user’s data which the backup do not cover.

#### Oxygen Forensic Suite 2013

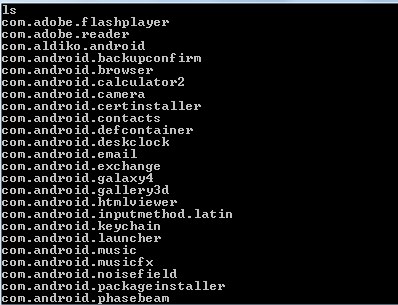
Another potential technology is **Oxygen Forensic Suite 2013.** It is a mobile device forensic software that goes beyond standard logical analysis of mobile devices. Using advanced proprietary protocols, it is able to extract much more data than usually extracted by logical forensic tools. It is able to support a huge range of android devices thus being relevant for our project. Rooting a device reveals the complete set of user data to the examiner, however using this suite; it will guide the examiner through the whole process of getting the root rights of the device. An important thing to note is that the root access will be revoked immediately after rebooting the device thus making rooting and further extraction forensic and safe. Oxygen Forensic Suite 2013 provides a good way of conducting mobile device forensic, however it only support Windows operating system which might be a downside for examiners using the Linux operating system. In addition, the Oxygen Forensic Suite 2013 only provides a 30 day trial version to the public. Subsequent users will require a price quote from Oxygen Forensic itself. A way to bypass it is to actually install the suite on a virtual machine itself and reverting to the first snapshot in order to “renew” the license. This entire process is tedious and therefore resulting in a **medium** feasibility.

#### Manual extracting of data

A **high** feasibility way of conducting mobile device forensic is to **manually** extract the data from the android device with the use of **adb** (Android Debugging Bridge). Adb is a protocol that helps to connect to Android device and perform some commands.

It is therefore possible to extract the data of all the files on the system or only the relevant file that the examiner wants from and Android phone through the use of **adb.** This however requires the device to be first unlocked or USB-debugging is previously enabled. If USB debugging for the Android device is enabled and rooted, the screen lock of the device can be bypass by updating the SQLite file using adb. This meddles with the system file, thus it should have little to no impact regarding the tampering of user data. However a safer method will be to use **hashcat** which is a brute force tool to obtain the pin if the md5 and the salt are known, which can be obtained from the various directories in the android device.

Everything is an **application** in android, be it browser, gallery or contact. They are applications that come together with the phone. A simple “ls” command on the adb will show the various applications as shown below. In the image shown below, important applications such as contacts, gallery and email are shown.



Data belonging to the various applications will be stored in /data/data folder. In the ‘databases’ folder, there are DB file types which are SQLite files where the data is stored. To view the data present in them, a browser such as Sqlite browser can be used to show the database structure and aid in helping the examiner to browse the data stored in them. This allows the extraction of various user data which can be used for the android forensic process simply by using a pull command on the adb, for example “C:\android-sdk-windows\platform-tools>adb.exe      pull      /data/data/com.facebook.katana C:\test”. This will extract the user’s facebook application data to a stored local directory to do analysis on.

In conclusion, for this project, our product will aim to automate the manual extraction of mobile device forensic data and sent it to the database for the splunk application to display the collected data. Therefore our product will be a python script as it allows multiple platform usage of our product as most operating system will be able to run python script. Our script will utilize **pyadb 0.1.1** (a simple python module to interact with the ADB tool) to extract user’s data from the mobile device. A clone of the current mobile device will also be first created so as to prevent the actual tempering of the user Android device’s data.

### 2.1.3 Splunk 6.0

Splunk is an operational intelligence platform that is capable of indexing large amounts of machine data sets and generating charts and dashboards to support the business intelligence. It can be installed on a laptop or scaled up to multiple servers.

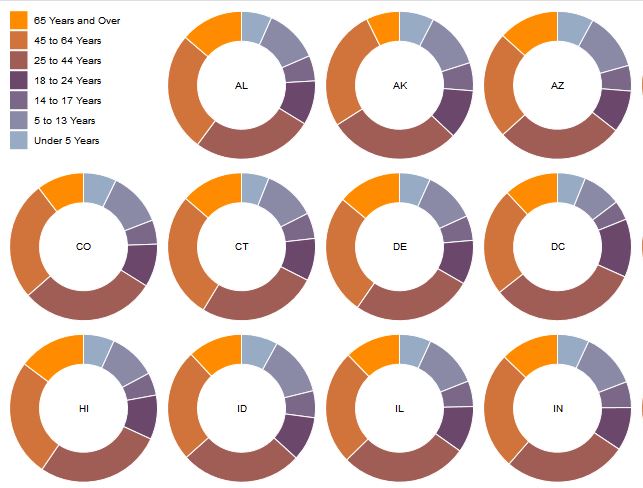
Splunk has recently upgraded to version 6.0 on 1st October 2013, with the new version comes a new web framework that enables developers to convert dashboards directly to HTML and JavaScript.

The new web framework allows the developer to incorporate tools and libraries that are not limited to the Splunk views in the Web Framework.

#### Visualization

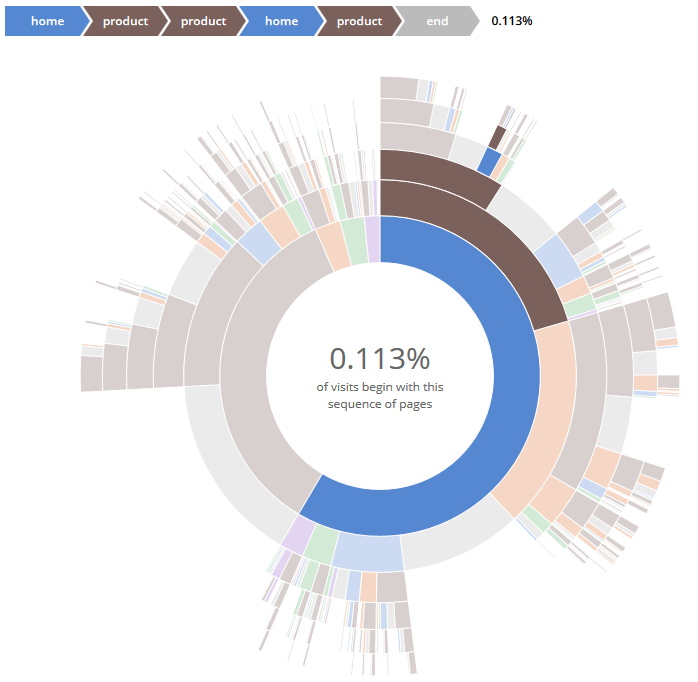
D3.js is a JavaScript library that provides rich and interactive charts for data visualization.

Below are descriptions of a few D3 charts that are feasible for Splunk for Mobile Device Forensics



This the Donut Multiples chart. Each donut may represent a single device, with the slices representing the different source types.

It not only provides a quick overview of all devices that are indexed by Splunk, it also shows the analyst a breakdown of the percentages of source types within the device



This is the sequence sunburst diagram.

The sequence sunburst diagram is effective at displaying a directory or navigation paths. This may be a good diagram to display all the directories within the mobile device itself. The challenge is whether Splunk is able to properly pass the data set into D3.



Splunk 6.0 comes with an integrated map feature that allows app developers to display geographic data. But the map is only capable of zooming as far as the picture above, zoom any closer and the map will disappear to a white background. It is not effective for showing show street level detail that is required to present mobile gps data. The mapping feature is only detailed in the USA region.



This is the Google Map add-on for Splunk and it extends the features of Google Maps into Splunk.

This is effective for displaying the gps coordinates that are extracted from the mobile devices. A feasible dashboard would allow the analyst to select a particular mobile device and Splunk will display the locations that the phone has been.

#### Getting data into Splunk

For this project, there are three main ways of passing the mobile data into Splunk namely manually adding the data through the Splunk web browser, through a TCP socket or through a SQLite database.

Adding the data manually through the browser is not a preferable solution and only should be done as a last resort when there are no other methods available to push the data into Splunk.

Another option is configuring Splunk to listen to a TCP port and accepting the data input sent through. This method is only preferable when the source of the data is from a remote machine, e.g. syslog. However, since it is the assumption that the machine extracting data from the mobile phone is the same machine running the Splunk server, opening a port is therefore not necessary.

The last option is configuring Splunk to retrieve the data from a SQLite database. Once data is extracted from the mobile phones, each event is inserted into the SQLite database. Splunk will then periodically run a python script to retrieve any new events which are entered into the SQLite database. This method is the most preferable compared to the previous 2 examples, data is stored within a single SQLite file which makes it easy to back up data that has been extracted.

## 2.2 Business Case and Security Considerations

### 2.2.1 Business Case

Even though there are many mobile device forensic tools in the market currently, most of them require the installation of their application into the device before mobile device forensics of the phone can be conducted. This contradict the main purpose of mobile device forensics as installing an application into a suspect’s mobile device might tamper with the data contained in it, thus defeating the purpose of conducting the mobile device forensics examination. There are several other mobile device forensic tools that are very useful in the forensic process. One such example is Oxygen Forensic Mobile Suite, however it only works on the Windows operating system therefore causing analyst using other operating system is unable to use this application.

Our product, Splunk for mobile device forensics, aims to provide a convenient and effective way to conduct mobile device forensics and also display a comprehensive overview and classification of the data collected in a simple format for the mobile device forensic examiner to view. Before conducting mobile device forensics, there might be many steps that have to be done in order to extract data from a suspect’s mobile device. This might cause an inexperienced mobile device forensic examiner to accidentally tamper with the suspect’s mobile device data resulting in a failed examination.

In conclusion, our product will aim to automate this mobile device data acquisition process and assist the mobile device forensic examiner in speeding up the forensic analyst process.

### 2.2.2 Security Considerations

#### Ensuring Data Integrity

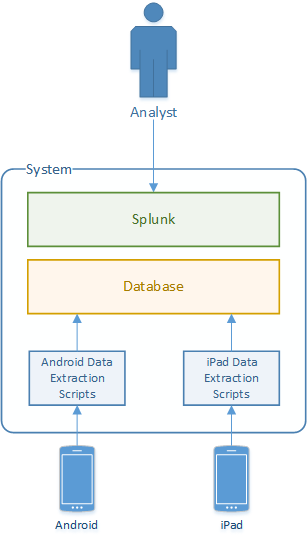
There are different types of approaches in performing data acquisition for mobile device forensic on both the android and iPhone mobile devices. A more direct approach is to install application straight into the mobile devices and then use the application to retrieve the data out from the mobile devices. However in this case, the data was already tempered by installing the application into the mobile devices.

When performing data acquisition in a forensic crime scene case study, it is important to take into considerations not to modify the data in mobile devices as it will be tampering the evidence, which is not what we would want to achieve. Our project will ensure data integrity when extracting data out from the mobile devices.

#### Read access only for extraction and analyzing of data

In order to be completely sure that the data is kept in its original state, all data extraction scripts will use read-only permissions to extract the data on the mobile device. This is to ensure that there will be no chance of accidentally tainting the digital evidence.

2.3 Architecture Diagram



The analyst will first execute scripts to extract data that resides on both the Android phone and the iPad. The scripts will save a copy of the data extracted and proceed to categorically insert into the Database.

Splunk will periodically extract data from the database and store it within the app index. Once the data is indexed by Splunk, the analyst will be able to generate intelligence through the dashboards within the Splunk for Mobile Device Forensics app.

## 2.4 Team Organization

**Module Group:** IT3527 – 01

**Team Name:** Blazers

**Team No:** 6

**Team Leader:**  Kok Jian Hui

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Name** | **Admin No** | **Hand phone No** | **Email Address** |
| 1\* | Kok Jian Hui | 113799R | 96754549 | [113799R@mymail.nyp.edu.sg](mailto:113799R@mymail.nyp.edu.sg) |
| 2 | Joshua Sim Renhe | 116274W | 82330149 | [116274W@mymail.nyp.edu.sg](mailto:116274W@mymail.nyp.edu.sg) |
| 3 | Alvin Lee Yong Teck | 113871T | 83664422 | [113340E@mymail.nyp.edu.sg](mailto:113340E@mymail.nyp.edu.sg) |

\*denotes the team leader

## 2.5 Task Allocation

|  |  |  |  |
| --- | --- | --- | --- |
| S/N | Name | Business Functions | Security Functions |
| 1. | Kok Jian Hui | * Python Script targeting at android mobile device with the ability to   + Detect the type of the android device connected and the platform the script is currently running on   + Choose the variety of information to be collected from the Android mobile device   + Extract and create a copy of the suspect’s data   + Start the examination of the data on another copy of the suspect’s data   + Compile and format the data collected and store into the database * Creation of relevant tables in database | * Conduct mobile device forensic without tampering the data contained in the mobile device * Secure coding |
| 2. | Joshua Sim Renhe | * Splunk 6.0   + Dashboard creation   + Mapping of GPS coordinates   + D3 visualization * Database management   + Normalization of data   + Administration of SQLite database | * Secure coding |
| 3. | Alvin Lee Yong Teck | * Python Script targeting at iPhone mobile device with the ability to   + Detect the type of iPhone connected and the platform the script is currently running on   + Choose the variety of information to be collected from the iPhone   + Extract and create a copy of the suspect’s data   + Start the examination of the data on another copy of the suspect’s data   + Compile and format the data collected and store into the database * Creation of relevant tables in database | * Conduct mobile device forensic without tampering the data contained in the mobile device * Secure coding |

## 2.6 Project Schedule

|  |  |  |
| --- | --- | --- |
| **Phases/Deliverables** | **Week** | **Deadlines of Deliverables** |
| **Project Briefing**   * Team Organisation Form | **1** | **18th Oct 2013** |
| **Project Inception**   * Literature Review * Proposed system and its aim * Threats identification & mitigation * Plan (Task Allocation/Schedule/Equipment) * Project proposal presentation | **2** | **25th Oct 2013** |
| **Project Requirements and Design**   * Functional and non-functional requirements * Use case and sequence diagrams * System architecture and system components * System prototype demo | **4** | **8th Nov 2013** |
| **Project Implementation Update 1**   * System implementation * Implementation progress update * System presentation | **7** | **29th Nov 2013** |
| **Project Implementation Update 2**   * System implementation and integration * Test plan/test cases * Integrated system demo | **12** | **3rd Jan 2014** |
| **Project Integration and Testing**   * Integration and system testing * Error rectification and regression testing * Test results presentation * Final project report with test report | **14** | **17th Jan 2014** |
| **Project Completion**   * Presentation and system demo * Source codes and other related documents * Optional: Implemented system (VM or URL to web app if hosted on the cloud) | **17** | **7th Feb 2014** |

## 2.7 Resources

The following resources are needed for the development of our project

### 2.7.1 Hardware

* An iPhone 4S, or below (including iPhone 4, iPhone 3GS and iPhone 3G)
* An iPhone USB Cable
* An Android Mobile Phone (preferably Samsung)
* A Micro USB Cable for the Android Mobile Phone

### 2.7.2 Software

* Splunk
* Eclipse

# 3. Requirements Specifications (Indicate individual contributions)

## 3.1 Functional and Non-Functional Requirements

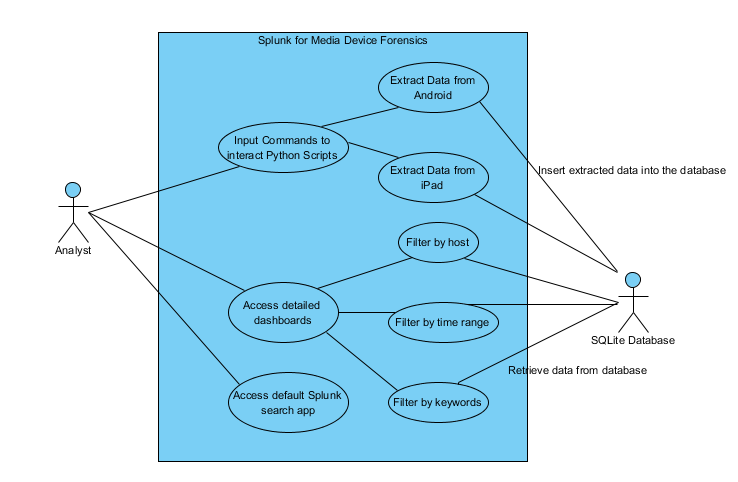
#### Functional Requirements

1. Analyst will be able to access dashboards on Splunk
2. Analyst will be able to filter search results by device, time range and keyword
3. Analyst will be able to extract data through automated scripts
4. Analyst will be able to generate PDF reports through Splunk

#### Non-Functional Requirement

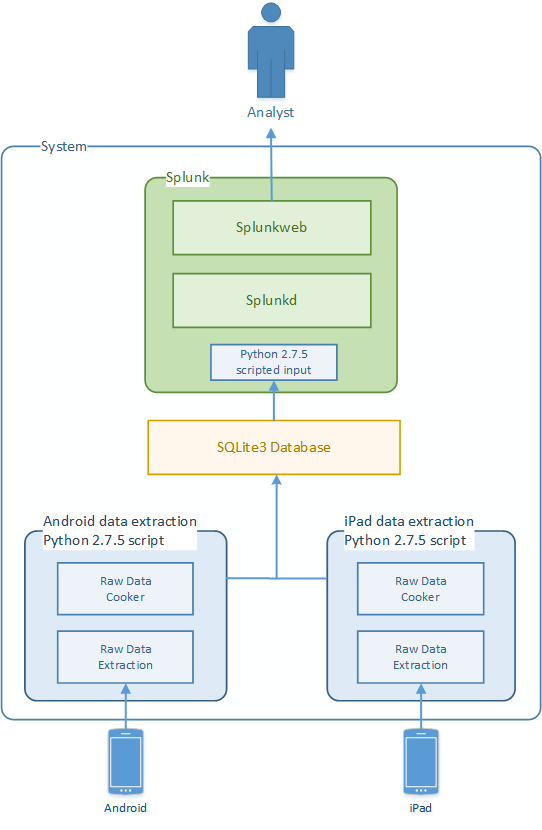
1. Splunk 6.0 shall be used
2. SQLite3 Database shall be used
3. Splunk shall use Python 2.7.5 script to extract the latest information from database
4. Data extraction scripts shall be written in Python 2.7.5

## 3.2 Use Case Diagram



# 4. Design Specifications (Indicate individual contributions)

## 4.1 System Architecture and System Components



The analyst will first execute Python 2.7.5 scripts to perform data extraction on both the Android phone and the iPad. The Python 2.7.5 scripts will save a copy of the data extracted and proceed to categorically insert it into the SQLite3 Database.

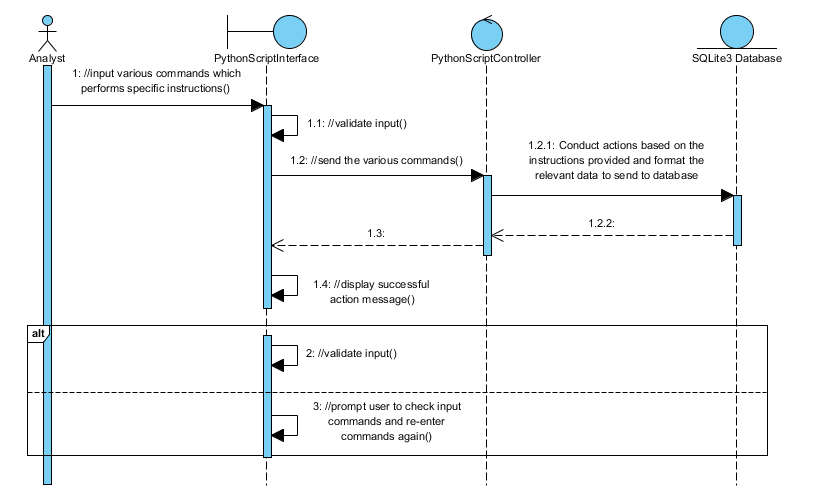
Splunk will periodically extract data from the Python 2.7.5 scripted input in the SQLite3 database and store it within the app index. Once the data is indexed by Splunk, the analyst will be able to generate intelligence through the dashboards within the Splunk for Mobile Device Forensics app.

#### System Components

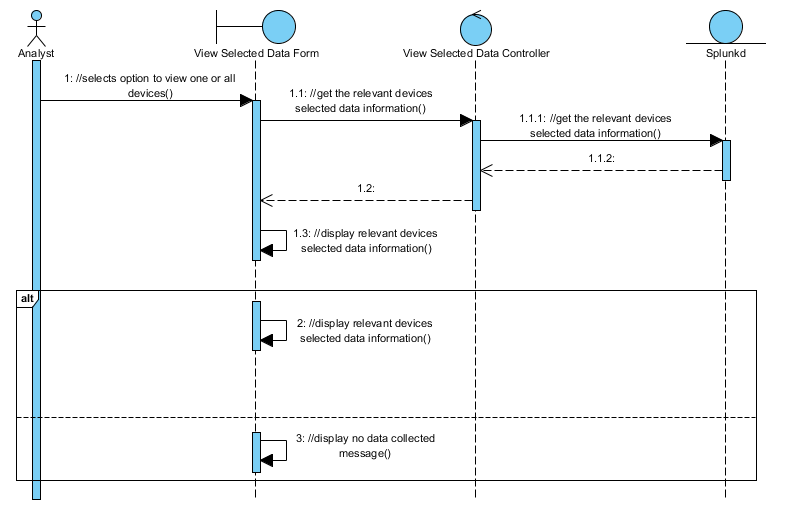
1. Python 2.7.5
2. SQLite3
3. Splunk 6

## 4.2 Sequence Diagrams

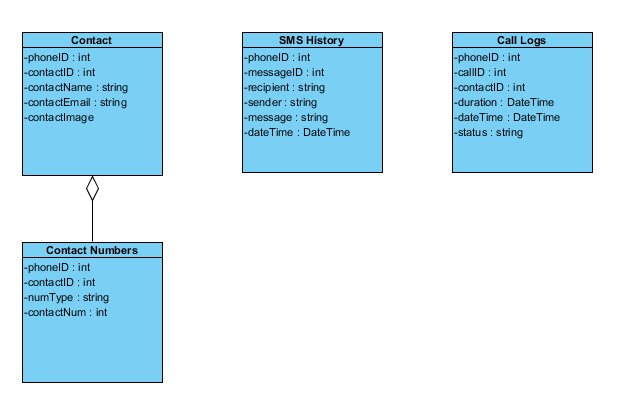
**Extraction of data from mobile device (Alvin & Caleb)**



**Viewing of User selected data (Joshua)**



## 4.3 Design Class Diagrams



## 4.4 Interface Specifications

The software components within the system shall interface and exchange information through the following method:

* The Python data extraction scripts for Android and iPad will retrieve the raw data through various forensic methods
* The data is then processed and inserted into the SQLite3 database.
* Splunk will periodically run a Python script to extract the latest datasets in the SQLite3 database

Thus the SQLite3 database will form the boundary between each component of the system. The challenge is to constantly normalize and synchronise any changes made to the database among all members of the team. This is a much preferable method as compared to using sockets to transfer data to Splunk, members are able to concentrate on their software development without having to worry about an additional socket layer.

# 5. Implementation Updates (Indicate individual contributions)

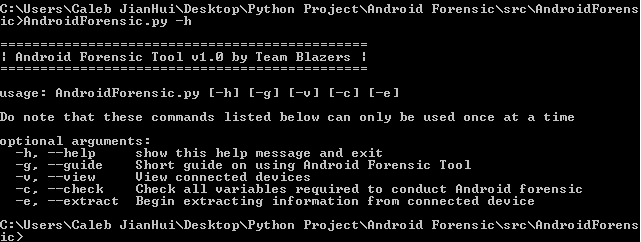
## 5.1 Initial Prototype

#### Python Script for Android (Caleb)

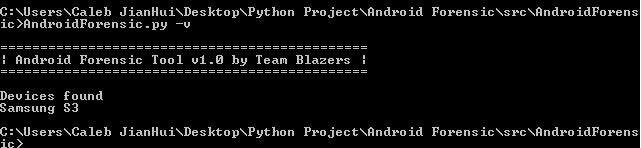
Our project aim to create a python script that can be used in the command line to extract information from an android device and Ipad. Therefore, we try to reduce the process it takes to extract the information from the device by using command line arguments for our python script. Below is a simple preview of the prototype for the Python Script for Android devices.



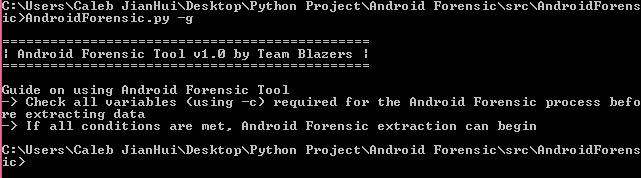
As shown above, upon attempting to execute our python script, it will prompt the user to input a valid command and to use –h to view a list of commands. This message will also be shown whenever an invalid command is inputted.



A list of commands that can be used is shown to the user upon using the –h command. One thing to note however that only one command can be used at a time. Upon using two commands at the same time, it will result in the first present command in the list from top to bottom. The list is shown above under the optional arguments.



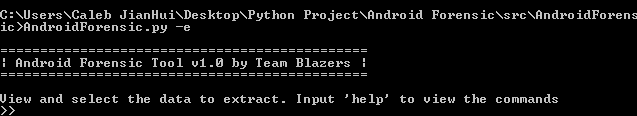
Devices that are currently connected are shown using the –v command. It is recommended to only have the target device connected so as to prevent extraction of data from a wrong device.



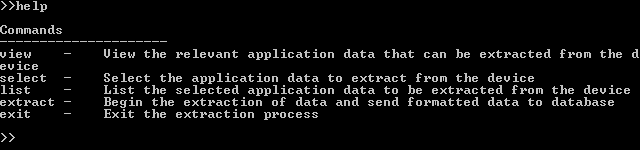
A short guide on using our python script is shown to the user upon using the –g command. This should be a simple guide for the user to use our python script



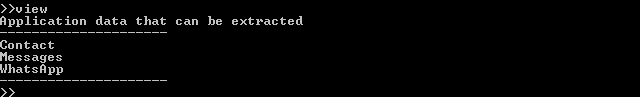
Before the extraction of data from the android device, it is recommended to use the –c command to check if all conditions required for the extraction is met.



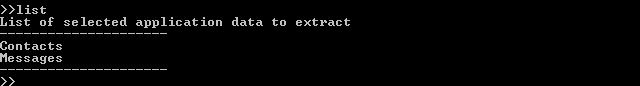
With the fulfillment of all of the conditions, the extraction process can begin using the –e command. A new set of commands is used in the extraction mode.



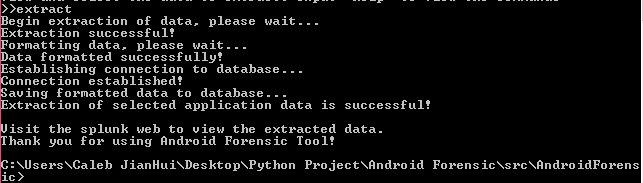
These new set of commands can be viewed using the help command. The relevant information about the relevant commands is also shown with these commands. The exit command will exit this interface and redirect the analyst back to the previous page.



The application data that can be extracted from the device is shown using the view command. Analyst can then select the various application data which he want to extract from the device using the select command.

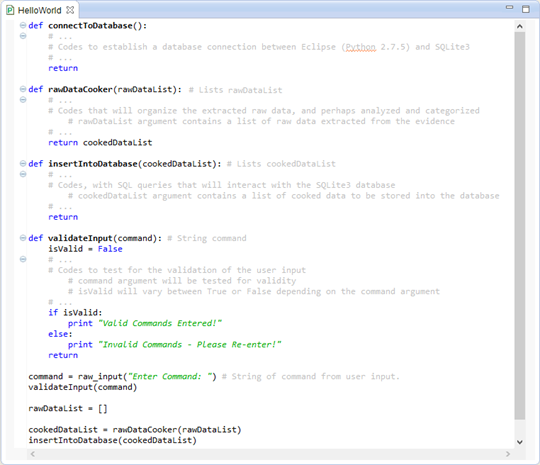


Analyst can view the various application data that he have selected to extract from the device using the list command. A default list of applications is selected at the start to ease the overall process.



Upon using the extract command, the selected applications data will be extracted and formatted before sending it to the database. The script will automatically close upon successful extraction of data

#### Python Script for IPad (Alvin)

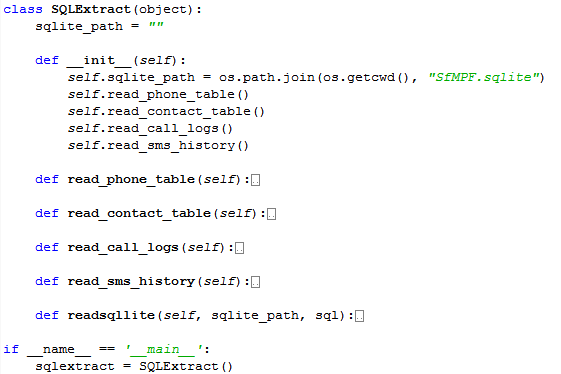


In the initial prototype, there will be four main functions of the Python 2.7.5 Scripting for Ipad based on the Sequence Diagram:

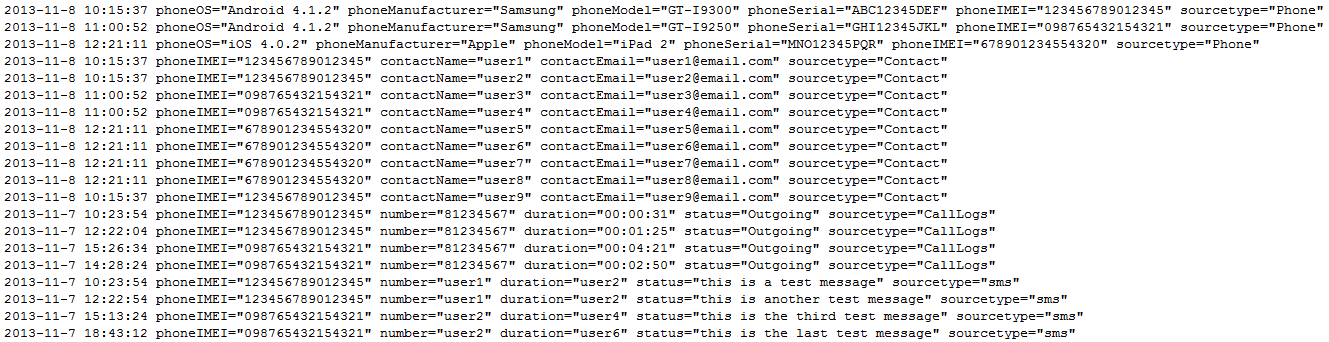
1. **validateInput(command):** This function will take in the *command* **String** argument which was input by the analyst, then the codes in the function will **perform validation testing** of the *command* input and print out whether is it a valid command or not. If yes, successful message will be prompt, else an error message will prompt the analyst to re-enter the command.
2. **rawDataCooker(rawDataList):** This function will take in the *rawDataList* **Lists** argument which contains the raw data extracted from the evidence (such as Android & iPad). The function will **organize** the extracted raw data, and perhaps **analysed** and **categorized** the data to achieve **cooked data**. This function will then finally return the *cookedDataList* generated in the function itself.
3. **connectToDatabase():** This function is straightforward that when this function was called, it will establish a database connection between the eclipse (Python 2.7.5) and the SQLite3 database to perform specific instructions, such as inserting of the data.
4. **insertIntoDatabase(cookedDataList):** This function will take in the *cookedDataList* **Lists** argument which contains the cooked data generated and returned from the function ***rawDataCooker*.** The codes in the function, with SQL queries will interact with the SQLite3 database, which will then store the list of cooked data in the *cookedDataList* into the SQLite3 database.

#### Splunk (Joshua)

The initial work before creating the Splunk application is to focus on the method in which the data is injected into Splunk. Splunk has a built in support for Python 2.7.x scripts, as such the script will be done in Python. The python script itself will be located in the same directory as the SQLite3 database.



The initial foundation for the script will focus on extracting the data from the database and printing it out for Splunk to capture. The next planned addition into the script is to add a function where the timestamp for the last event is saved into a text file, to ensure that the periodic extraction will only retrieve the rows that are created after the last timestamp.



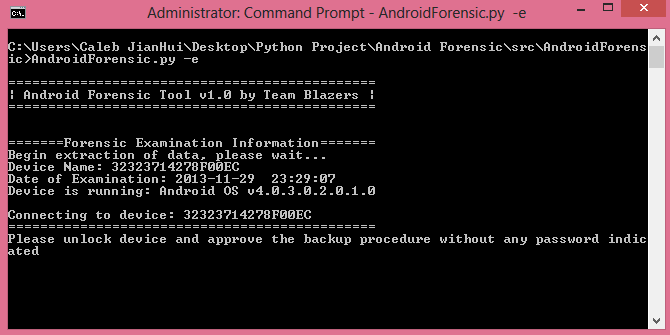
The extraction will be done in a table by table process. The format that is optimized for Splunk is the ***timestamp key=”value”***format. After extracting the latest data, the script will proceed to print it in the optimized format for Splunk. The image above illustrates what will be the output of the script will be. The IMEI is the International Mobile Station Equipment Identity, and it is 15 decimal digit value that is used to uniquely identify a device. Therefore each phone/tablet will be identified using its IMEI number within Splunk. Another alternative way to uniquely identify each device is to use its serial number.

## 5.2 Implementation Update 1

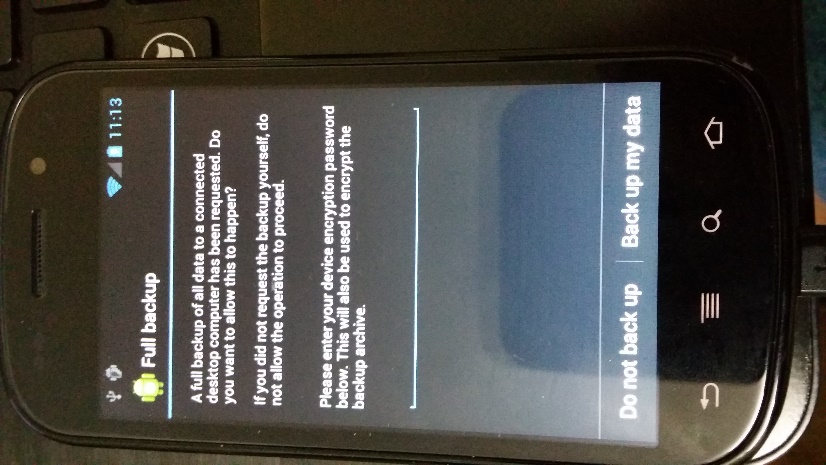
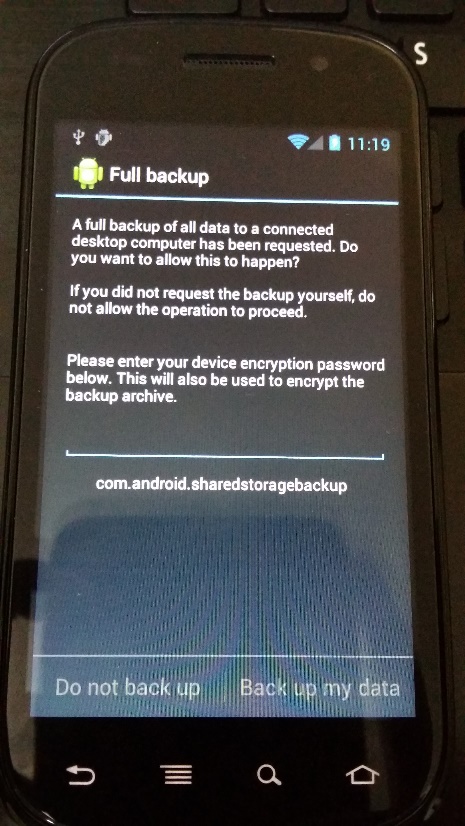
#### Python Script for Android (Caleb)

As mentioned earlier for the python script for Android, we aim to automate the process as much as possible. This is to allow both technical and non-technical user to utilize our python script. Therefore there are various updates done to the python script to make that possible.

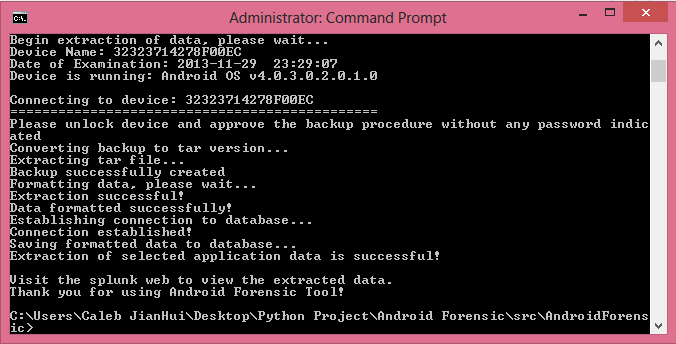
The extract function is modified to be instantaneous whereby the user just have to type in “AndroidForensic.py –e” in order to start the forensic examination. There is also no longer the option to select the various application data to extract, as we aim to extract as much data in the device as much as possible. This is because every single application data might be important in the forensic process.

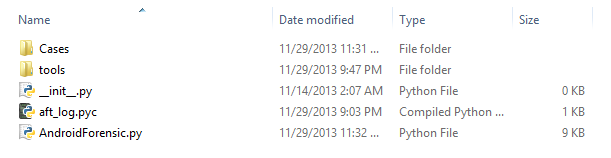


As shown above, the extraction process is made to be instantaneous. The date and time of the examination is also recorded for logging purpose. The device name and the operating system operating is also shown to the analyst for logging purpose. These various information will also be sent to splunk to provide the analyst with the device details.

   
**Before After**

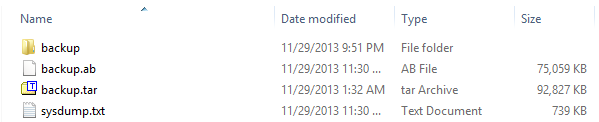
The user will be prompt to accept to do a full backup to do forensics on. Upon accepting, the backup process will start. It will take a moment before the backup process is completed depending on the storage size of the device. The rest of the process is the same as before except the conversion of the backup file which will be explained later.



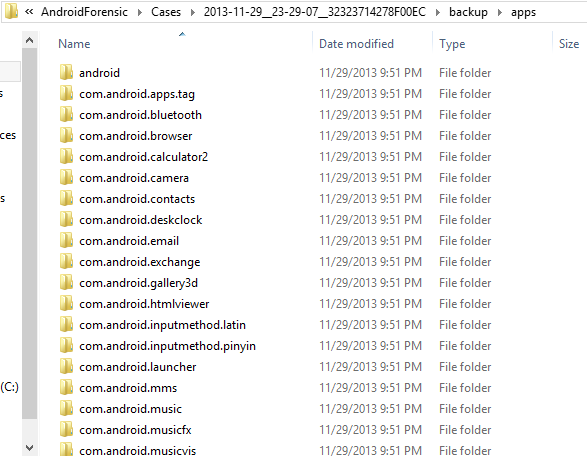


A new folder will be created in the same folder as the Android Forensic Tool named “Cases” which will contain all the different examined device named in the <date> and <device number> format.



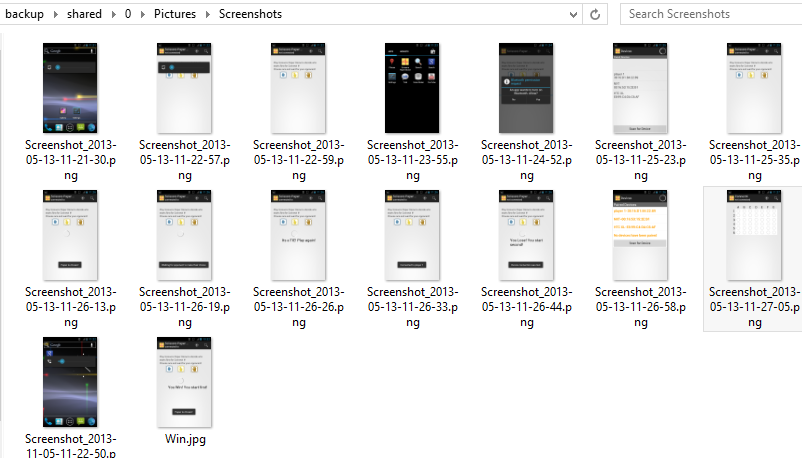


In this folder contain the backup.ab which is the backup file created by android itself. The python script automatically tries to convert it to a tar format before further extracting the files out. This is transparent to the user as the various command and files need to convert and extract the files is done by the python script itself

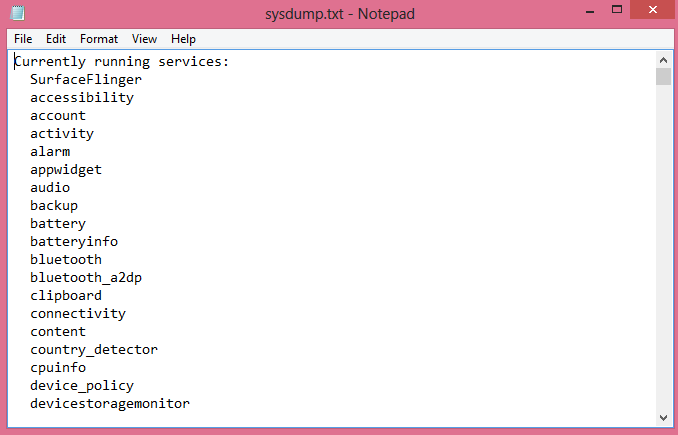


In the app folder contains the various application folder. The various application data are then able to be analyzed by the python script to format and sent to the database. This work is still in progress.

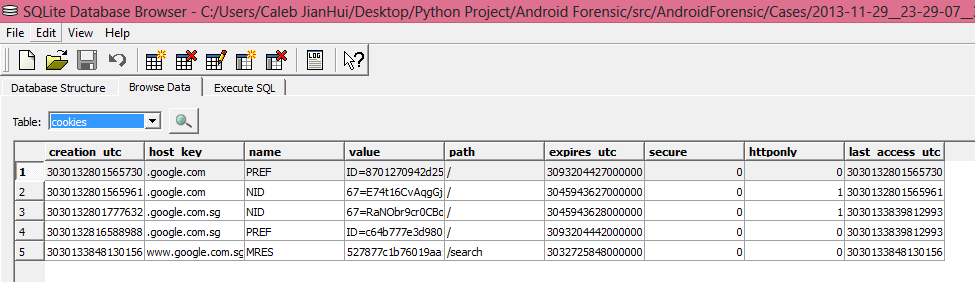
A notable thing is that in the share folder contains various important data which can be extracted to be examined by the analyst. An example is shown below where the screenshot folders contain the various images in the device itself.



A system dump of the device is also done to obtain the various processes running in the device itself. These data are placed in the sysdump.txt. A sample data is shown below.



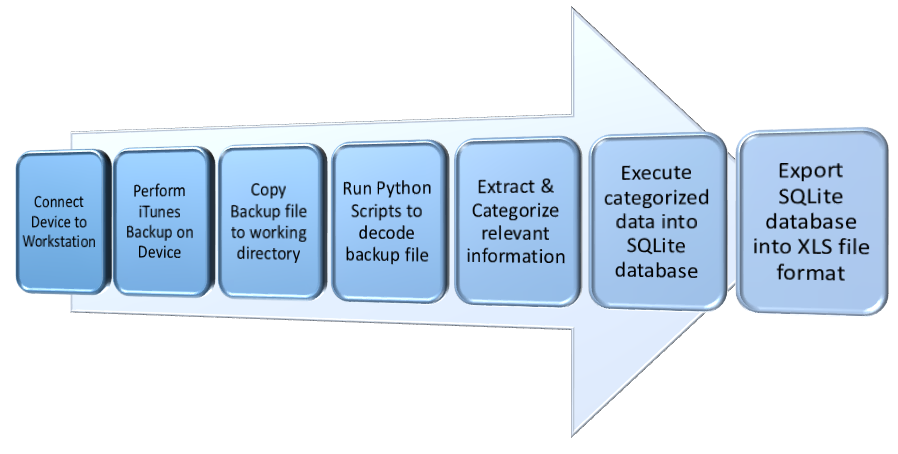
Using sqlitebrowser to temporarily examine the database of the different applications show important data that can be used in the forensic examination. A screenshot of the databases information for the default browser cookies are shown below.



##### Why utilize android default backup to extract information

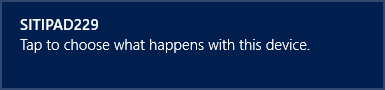
Utilizing exploit to gain access the system is risky as it have a probability to destroy the data in the device itself. Therefore, it is better to utilize the android default backup as it is safer. However for devices with android version lower than Ice Cream Sandwich will require the help of exploits to obtain the root access to conduct forensic examination on them.

#### Python Script for IPad (Alvin)

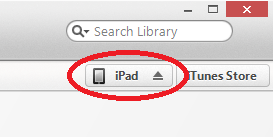


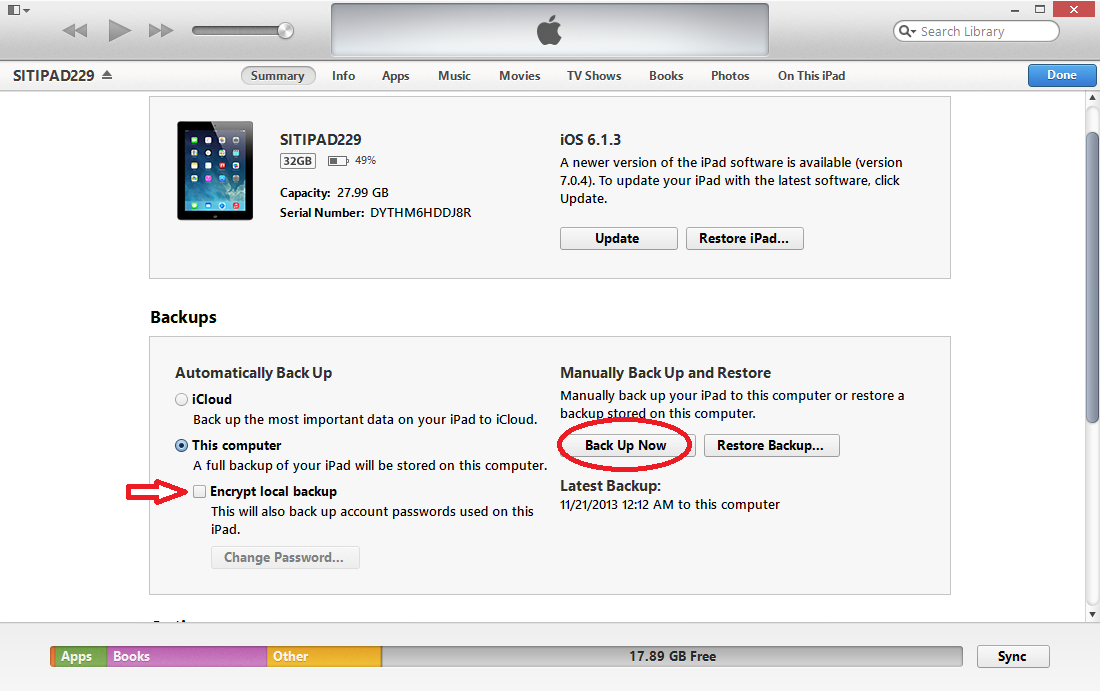
There are a total of 7 steps in the project process flow for iPad Forensic Data Extraction process:

1. **Connect Device to Workstation:** Using Apple iOS USB Cable to establish connection between the iPad device and the forensic workstation. Once the device is connected, on the Windows 8 machine, the following will pop up to show successful connection between iPad and workstation:

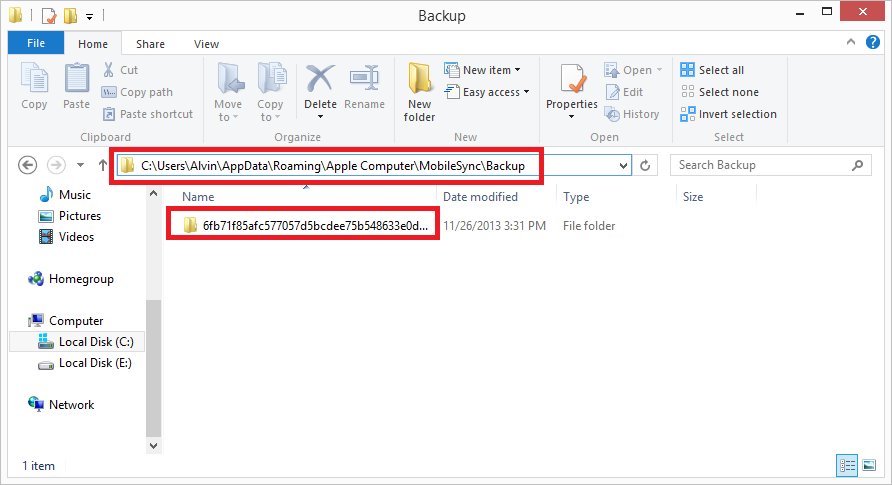


1. **Perform iTunes Backup on Device:** Once the device is connected to the workstation, the device icon will be shown at the top right of the iTunes window as shown below:

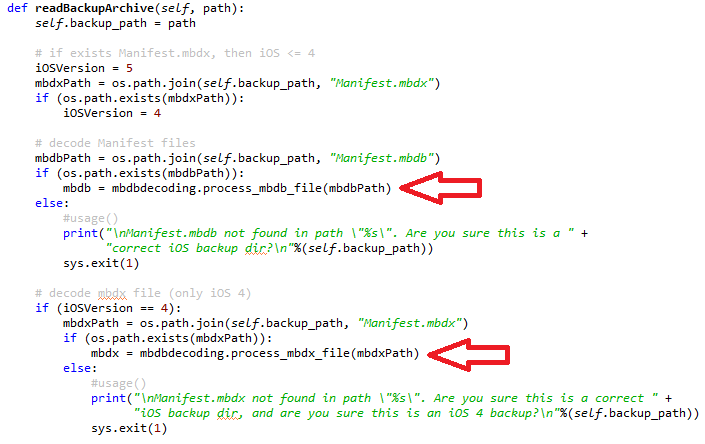


Click it, and it will lead to the screen as shown in the picture below. Under the ***Backups*** section, make sure that the ***Encrypt local backup*** was unchecked before performing the backup process. We would not want to encrypt the backup folder we are going to create. Once everything is set, click on the ***Back Up Now*** button and it will start the backup process.

**\*Note:** Make sure that iTunes were already installed onto the forensic workstation. If not, go to this [link](http://www.apple.com/sg/itunes/download/) to download and then install iTunes.

1. **Copy Backup file to working directory:** Once the backup was made, an encoded folder will be generated at the following directory by default: ***C:\Users\Alvin\AppData\Roaming\Apple Computer\MobileSync\Backup***. The encoded folder is the backup of the iPad that was connected to the workstation. 

Although the directory path of the backup folder was fixed, the user (Alvin) is dynamic that will change and varies among every users. Hence, the backup file will be copied to the working directory of the python scripts to allow easier access to the backup folder using python scripts.

1. **Run Python Scripts to decode backup file:** Once the python scripts are able to access the backup folder, it will parse the backup file path into the function ***readBackupArchive.***

The python scripts above will first determine the iOS version of the backup before performing the decoding process. For iOS version lesser or equal to 4, the backup will have 2 files – ***Manifest.mbdx*** and ***Manifest.mbdb***.

***Manifest.mbdx:*** Acts as an index file for the backup and indexes the elements that will be found in Manifest.mbdb.

***Manifest.mbdb:*** A binary file that contains information about all other files in the backup along with the file sizes and file system structure data.

For iOS version greater or equal to 5, index file (Manifest.mbdx) is eliminated and the backup is managed by a single Manifest.mbdb file. There will be different techniques, algorithms, and scripts to decode the backup file, hence, there is a need to identify the iOS version before going in depth to know which file to process as shown in the picture above; the mbdx and mbdb file. In the era of technology advancement, most of the iOS devices are now having iOS version 5 and above. The iPad for the project has an iOS version of 6. Hence, we will go in depth to talk more about the decoding of mbdb instead of the mbdx.

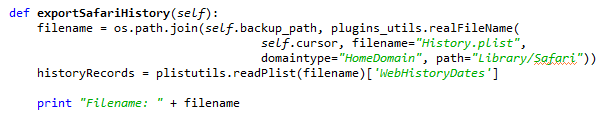
The following is the python scripts that shows the process of decoding the Manifest.mbdb file. The script will create a dictionary called ***mbdb***, which map the offset of the info in this file to the file info. A dictionary is a data structure that maps one value to another.



1. **Extract & Categorize relevant information:** After the Manifest.mbdb file was decoded, the following image will show the database created in memory will later be populated by parsing the decoded manifest file items, and the creation of the cursor, which is used to manage the context of a fetch operation.



Once the database created in the memory has the populated data, the extraction process can be executed to extract relevant information out and categorize all the information to be extracted. The following is the method to extract the **Safari History** information decoded from the backup file. It first makes full use of the cursor created earlier on and various attributes to find out the **realFileName** of the **Safari History** created in the backup folder. The filename will then parse into the **readPlist** method in **plistutils** to get a list of **historyRecords** to be access later.



After the print of the filename processed out from the **plugins\_utils** after the joining the backup path (in **green**) and the **realFileName** (in **red**)

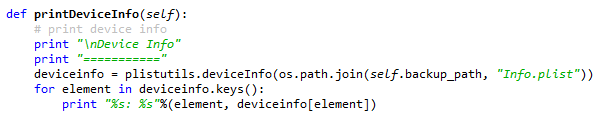
C:\Users\Alvin\Desktop\121.PNG

The **historyRecords** mentioned earlier on form a loop of available record and then retrieve out the information out based on the key input. E.g. **‘title’ in record.keys()** will get the title of the webpage in the safari history extracted from the backup.

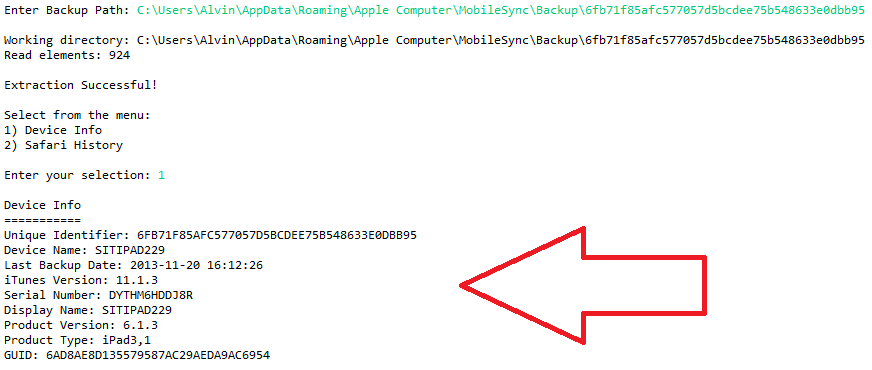


The list of data that can be extracted out from Safari History are:

1. Title of the webpage
2. URL
3. Last Visited Date time
4. Visit Count: Total number of visits of the webpage
5. Redirect URLs

The function **printDeviceInfo** will make full use of the **Info.plist** file found from the backup folder created using iTunes Backup to extract the information of the device out. The **plistutils.deviceInfo** function will take in the **Info.plist** and then process out into **deviceinfo** list, then from there print out the device info. Do note that the actual feature will not be just printing out the information, but will export out the information into sqlite3 database file.

The following is the output of the python scripts above, displaying out the device information out from the input backup path from the user. The output is from the Device Info onwards:



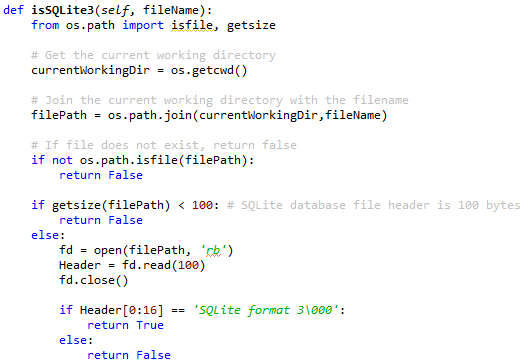
The list of data **related to the device** that can be extracted out from the device info are:

1. Unique Identifier
2. Device Name
3. Serial Number
4. Display Name
5. Project Version: iOS Version of the device
6. Product Type
7. GUID

**Additional Information:**

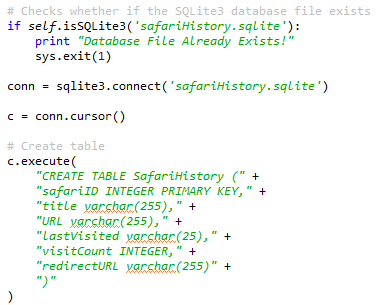
1. iTunes Version: Version when conducting the iTunes backup
2. Last Backup Date: Date time when conducting the iTunes backup
3. **Execute categorized data into SQLite database:** Referring to the python scripts above in the 5th process on how to extract the **SafariHistory** data out, this process will explain on how to make use of the categorized data and export out into the SQLite database. The python scripts in **Figure 2** shown below, will first checks whether the **safariHistory.sqlite** file are already existed or not. This will lead to the **isSQLite3** function shown below in **Figure 1**, which will first takes in the string of the output SQLite file, and then join path together with the working directory to check whether the file exist or not. If it exists, it will then continue to check against the size of the file whether is it lesser than 100, as SQLite database file header is 100 bytes. If the size is greater than 100, it will then read the first 100 bytes of the file and check whether the header does it equal to **SQLite format 3\000**. If there are equal, it will return **true,** indicating that the SQLite file is already exists, prompt error message and exits. Reason doing this is because some files have the same naming convention, but then it is not the genuine SQLite file.

**Figure 1**



Once the **safariHistory.sqlite** file does not exists, a connection to the file will be established, a cursor will be created, and then the cursor will execute the SQL scripts for the creation of the **SafariHistory** table to store the information.

**Figure 2**



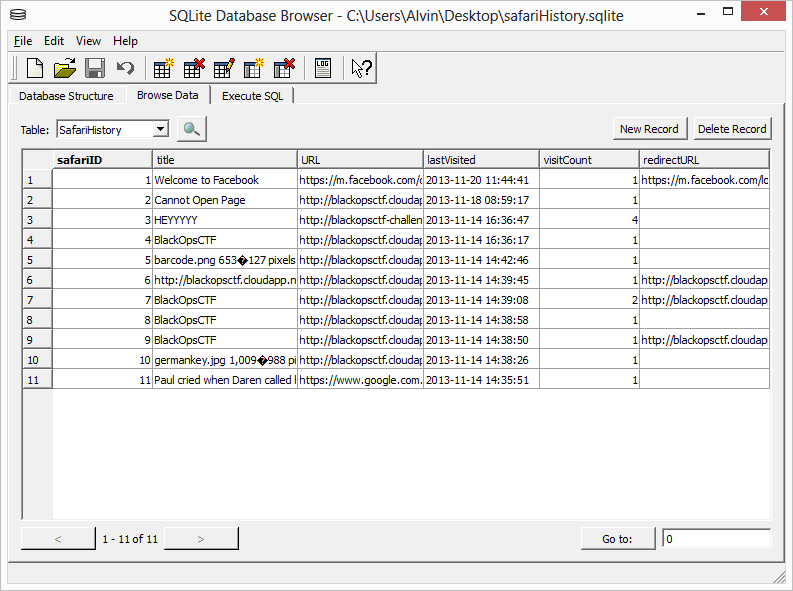
After the **SafariHistory** table was created, the cursor will then execute the SQL script to insert all the relevant data into the table using the attributes mentioned above in the 5th process. This insertion process will re-executed base on how many available data found in the *for* loop of the **historyRecords** mentioned in the 5th process.

C:\Users\Alvin\Desktop\6_3.PNG

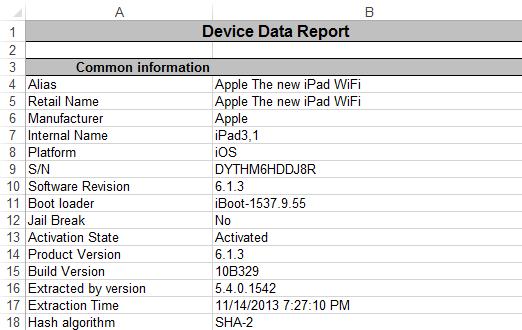
After the insertion of data into the **SafariHistory** table, the connection will commit and the closes.

C:\Users\Alvin\Desktop\6_4.PNG

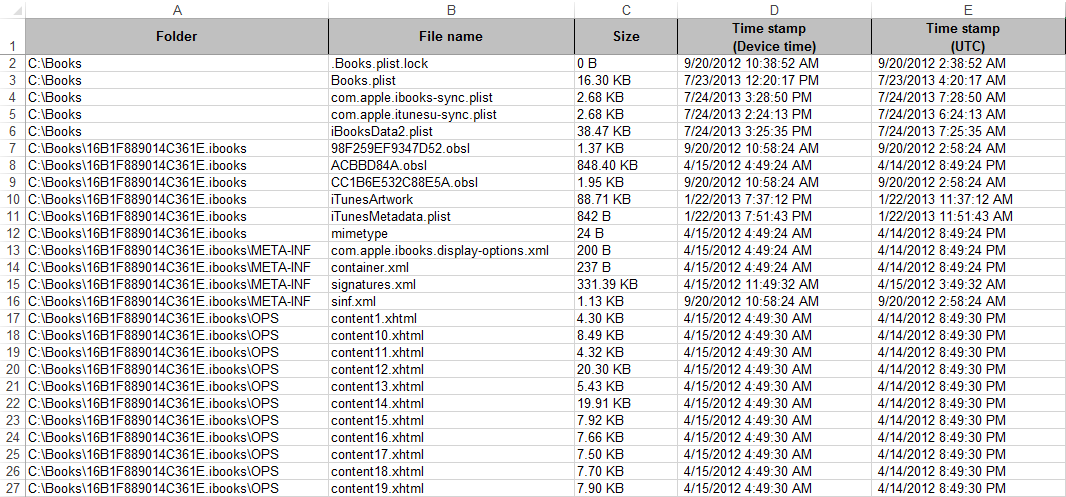
Once this is done, the **safariHistory.sqlite** file was successfully exported out, containing the **Safari History** data information inserted earlier in the process. The following is the image showing the SQLite file being browse by the **SQLite Database Browser**, showing the information:

****

1. **Export SQLite database into XLS format:** This will be an additional feature which serve as a backup reference of the data being exported out into SQLite database file. Oxygen Forensic Suite 2013 demonstrates an export of the data extracted into a XLS file. The following is the example of what will be contained inside the XLS file:

****

The image above shows the **Device Data Report** exported out from the iTunes backup using Oxygen Forensic Suite 2013.

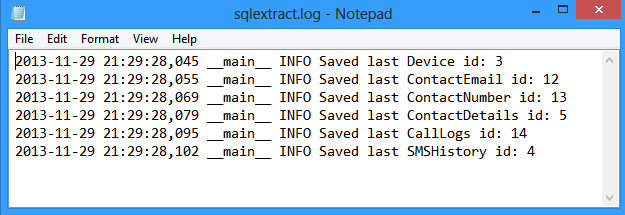
****

The image above shows the **File Browser Information** exported out from the iTunes backup using Oxygen Forensic Suite 2013.

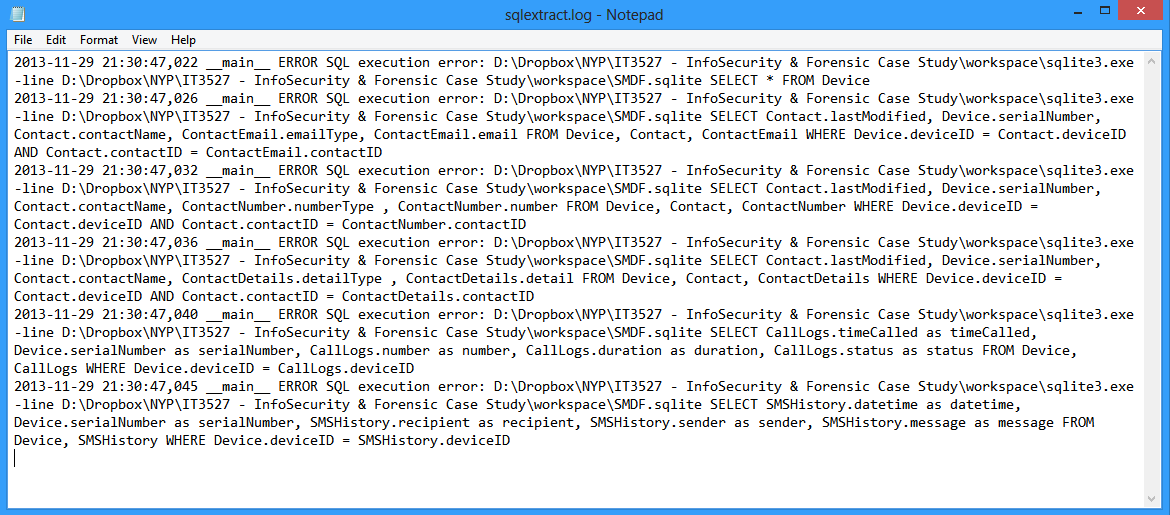
#### Splunk (Joshua)

##### Extraction Process

The extraction process from the SQLite3 database to Splunk is done through a python script; this script is executed every 10 seconds by Splunk. Once it extracts the datasets it saves a file containing the last ID of each table in the database, this is done to prevent duplication of data within Splunk’s index. The python script also saves information about the extraction process in a log file.

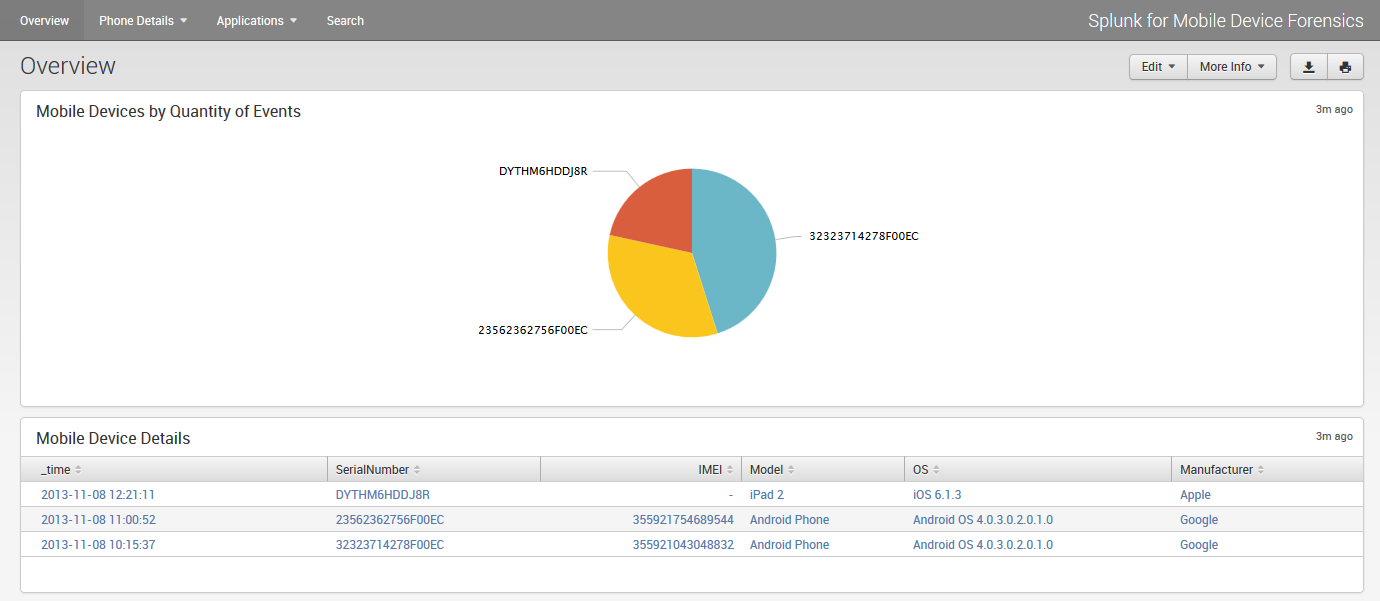


The log file above shows the log information that has been saved, when the python script successfully extracts a dataset from a table in the database



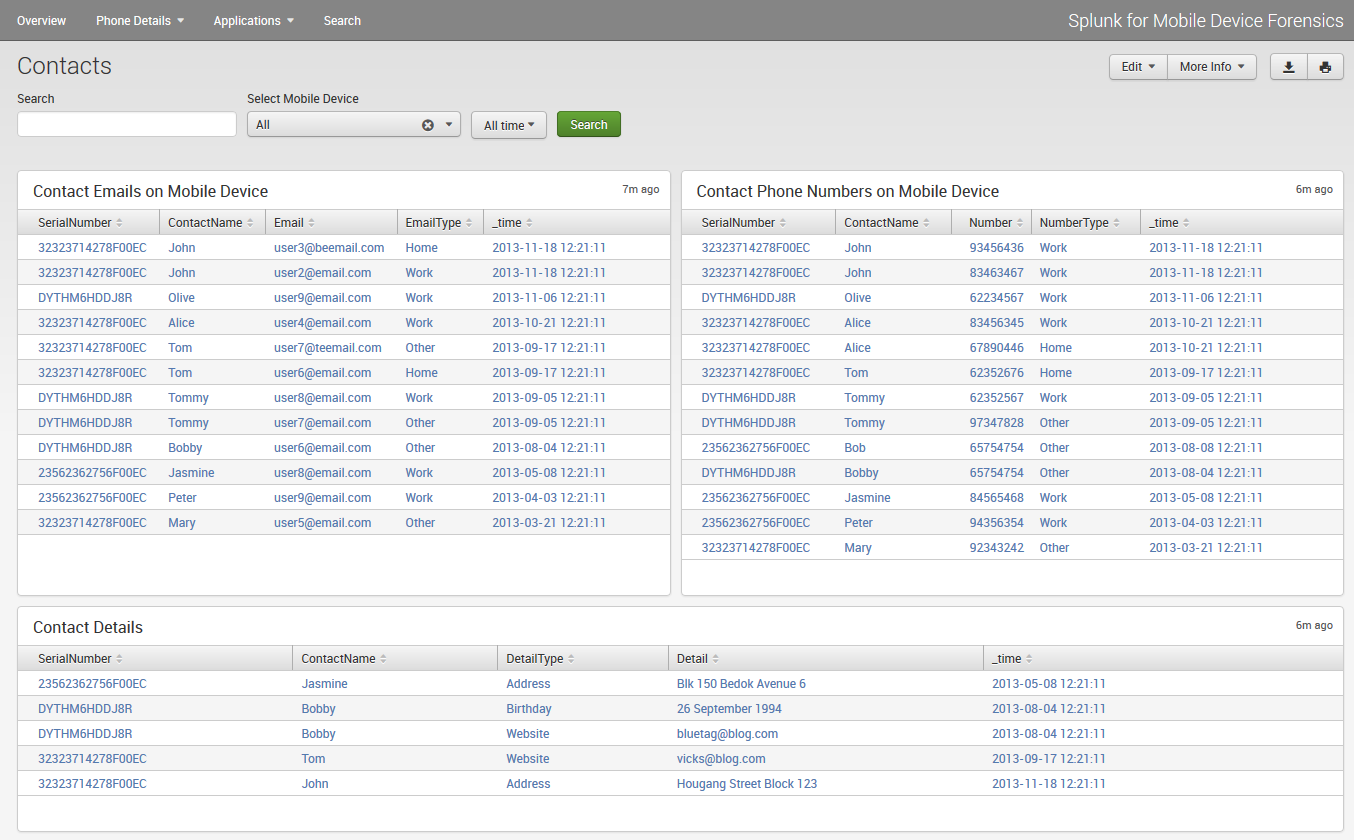
The above log shows an example when python script encounters an error. In this example the database was missing and the python script produced an error when it couldn’t execute the SQL statements.

##### Overview Dashboard



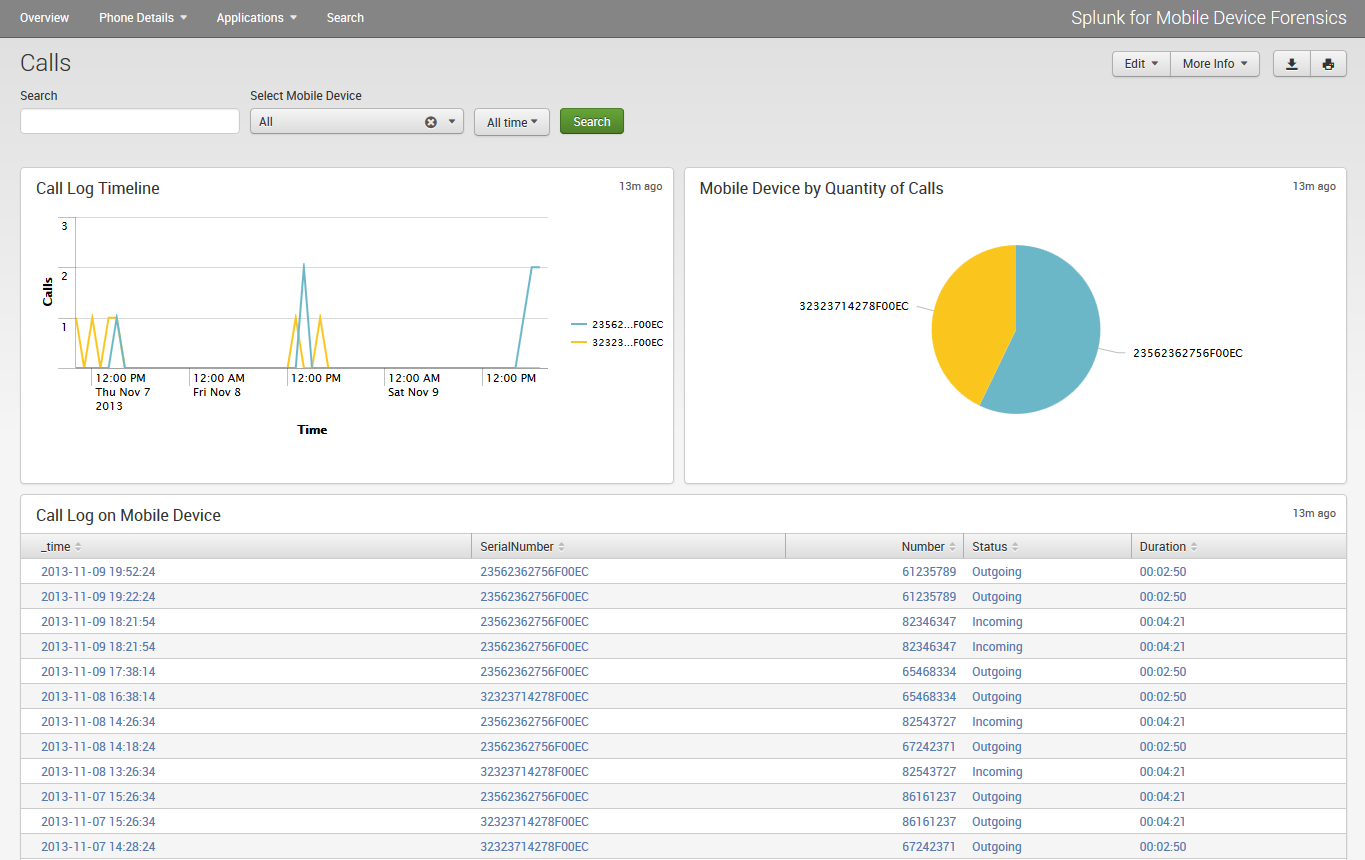
The dashboard above is the Overview dashboard. It provides an overall summary of what mobile device information is stored within the Splunk index. The pie chart shows the data percentage of each mobile device, to assist the investigator in prioritizing which device to analyze.

##### Contacts Dashboard



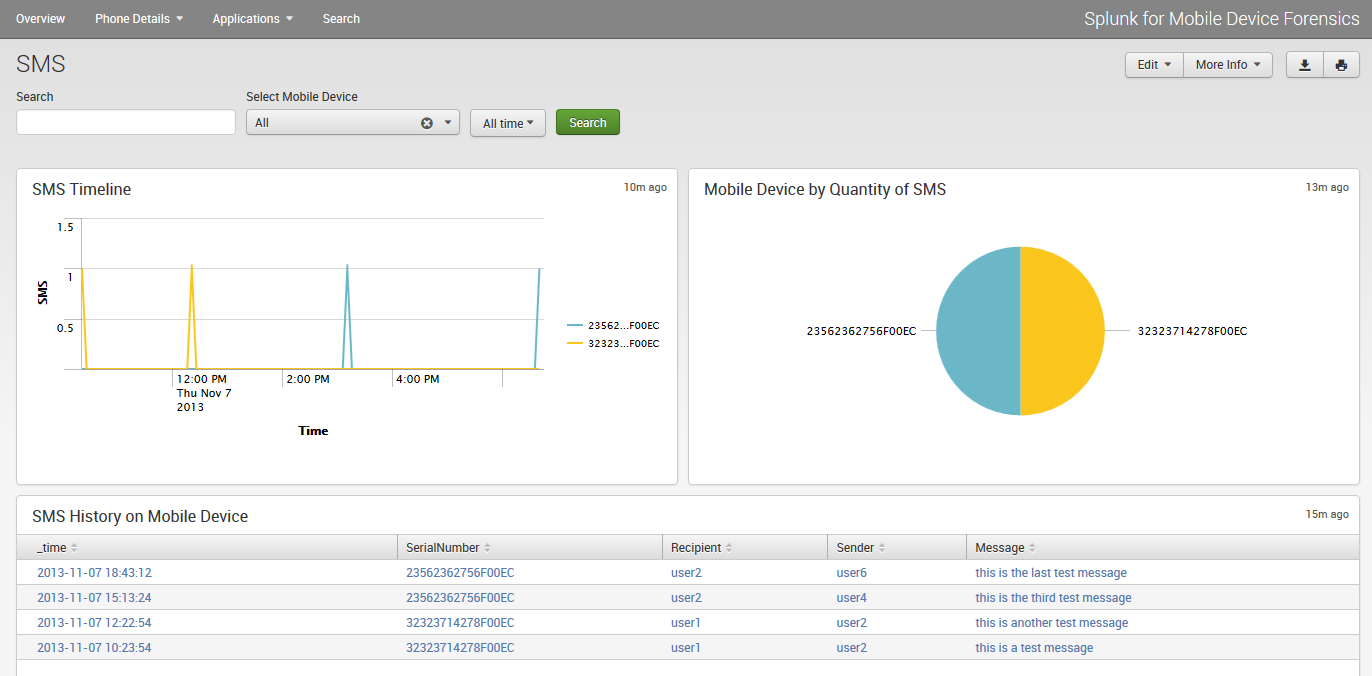
This is the Contacts Dashboard. It displays all contact that have been extracted from the mobile devices. It will display the Emails, Phone Numbers and other details of each contact. The dashboard allows the investigator to search for specific keywords, narrow down to a specific mobile device and zoom into the time range of when the contact was added to the mobile device.

##### Calls Dashboard



This is the Calls Dashboard. It displays all the incoming and outgoing calls made on the mobile device. The charts display the timeline of all the calls that were made, the volume of calls that were extracted from each device and the call logs sorted from latest to oldest. The dashboard allows for keyword search, specific mobile device data and narrow down to specific time range.

##### SMS History Dashboard



This is the SMS History Dashboard. It displays the SMS messages extracted from the mobile device. Similar to the Calls Dashboard, it will display the timeline of the SMS messages based on its timestamp, the volume of SMS messages per mobile device and the SMS messages ordered from the latest to oldest. The dashboard allows for keyword search, specific mobile device data and narrow down to specific time range.

## 5.3 Implementation Update 2

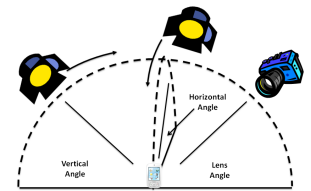
#### Python Script for Android (Caleb)

An important aspect of Android forensic is to actually gain access to the device itself. Without access to the device, android forensic cannot be done due to several permission rights. The techniques used can either be logical or physical. However, we will focus more on the logical techniques, which mostly involve gaining access to the file system. Below are several methods that can be done to gain access to the file system.

Smudge Attacks on Android

Touch Screens are an increasingly common feature on smartphones where size and user interface advantages accrue from consolidating multiple hardware components (Keyboard, number pad, etc) into a single software definable user interface. Oily residues, or smudges, on the touch screen surface, are one side effect of touches from which frequently used patterns such as a graphical password might be inferred. A note that this is not specific to only Android device but most touch screen smart devices which password can be deduced from the smudges found on the touch screen.

This attack depends on the fact that smudges are left behind by the user’s fingers due to repeated swiping across same locations. The pattern lock or any pass code is something the user will have to swipe every time that he want to use his mobile device. Smudges would be the heaviest across the same locations and hence using proper lighting and high-resolution pictures we can deduce the code as shown below. So during examining any device, forensic analyst usually take care to avoid hand contact with the screen to check for the smudge attack.



USB-debugging is enabled

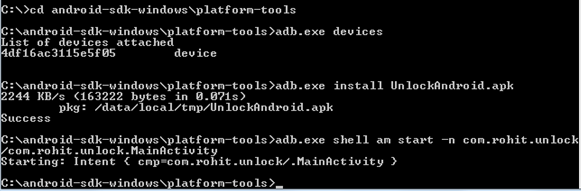
If USB-debugging in the Android device is enabled, then bypassing the lock code can be done easily with the use of ADB (Android Debugging Bridge). ADB is primarily a command line tool that communicates with the android device. ADB is bundled with the Android platform tools. In simple terms, this is what happens when ADB is used:

* An ADB daemon runs as a background process on each Android device
* When Android SDK is installed on the analyst machine, a client is run. The client can be invoked from shell by giving an ADB command
* A server is also run in the background to communicate between the client and ADB daemon running on the Android device.

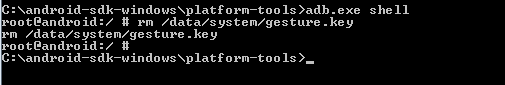
Any of the below methods can be used to take advantage of USB-debugging to bypass the lock screen:

* Using a specially created apk file

An apk file have been specially created to bypass the lock screen called UnlockAndroid.apk.

* + By connecting the device to the machine where the Android SDK (including platform tools etc.) is installed
  + Open command prompt and type cd C:\android-sdk-windows\platform-tools>adb.exe devices. The device must be identified by the ADB if everything is going fine. If the device is not identified, installing the relevant drivers for the device will rectify the problem.
  + Copy the UnlockAndroid.apk file into C:\android-sdk-windows\platform-tools directory.
  + In the command prompt, type C:\android-sdk-windows\platform-tools>adb.exe install UnlockAndroid.apk. Observe that the application is installed on the device.
  + To start the application, simply type C:\android-sdk-windows\platform-tools>adb.exe shell am start -n com.rohit.unlock /com.rohit.unlock.MainActivity
  + Observe that the screen lock is bypassed now and access to all the applications and folder in the mobile device is obtained. Below is a screenshot of the process.  
    
* Deleting the gesture.key file

If the Android device is using the pattern lock and it’s a rooted device, then the process below can be tried to bypass the lock screen.

* + Connect the device to the machine where Android SDK (including platform tools etc.) is installed.
  + Open command prompt and type cd C:\android-sdk-windows\platform-tools>adb.exe devices
  + The device will be identified by the ADB if everything is going fine.
  + Connect to ADB shell by typing adb.exe shell
  + The terminal appears giving you access to shell. Now type rm /data/system/gesture.key. This is the file where the pattern is stored.
  + Restart the phone and you will still observe that the device is asking for the pattern. You can draw any random pattern and unlock the device. Below is the screenshot of the process.  
    
* Updating the SQLite file

If the Android device is rooted, then by updating the SQLite files you can bypass the screen lock.

* + Connect the device to the machine where Android SDK (including platform tools etc.) is installed.
  + Open command prompt and type cd C:\android-sdk-windows\platform-tools>adb.exe devices
  + The device will be identified by the ADB if everything is going fine.
  + Connect to ADB shell by typing adb.exe shell
  + Type the following to bypass the lock screen:

|  |
| --- |
| cd /data/data/com.android.providers.settings/databases  sqlite settings.db  update system set value=0 where name=’lock\_pattern\_autolock’;  update system set value=0 where name=’lockscreen.lockedoutpermenantly’; |

* Cracking the PIN in Android

All the methods above involve bypassing the lock screen and completely deleting or disabling the lock screen. But what if you actually want to know the actual PIN so that you can lock/unlock at any time? In android, the PIN that user enters is stored in /data/system/password.key. As you might expect, this key is not stored in plain text for security reasons. It is first salted using a random string, then the SHA1-hashsum and MD5-hashsum are calculated and concatenated and then the final result is stored. Seems complicated, but it can be easily shown with the code below

|  |
| --- |
| [plain]  public byte[] passwordToHash(String password){  if (password == null){  return null;  }  String algo = null;  byte[] hashed = null;  try {  byte[] saltedPassword = (password + getSalt()).getBytes();  byte[] sha1 = MessageDigest.getInstance(algo = "SHA-1").digest(saltedPassword); byte[] md5 = MessageDigest.getInstance(algo = "MD5").digest(saltedPassword); hashed = (toHex(sha1) + toHex(md5)).getBytes();  }  catch (NoSuchAlgorithmException e){  Log.w(TAG, "Failed to encode string because of missing algorithm: " + algo);  }  return hashed; }  [/plain] |

Since the hash is salted, it is not possible to use a regular dictionary attack to get the original text.

* + Pull out the salt using adb. Salt is stored in the ‘secure’ table from /data/data/com.android.providers.settings/databases/settings.db
  + Get the password : sha1+md5: (40+32) (stored at /data/system/password.key)
  + Once you have the md5 and the salt, you can brute force using the tools available in market (Ex. hashcat) to get password.

There are no difficulties cracking or bypassing this kind of protection on an Android device, the only real obstacle is that we cannot directly access the /data/system folder and gesture.key file except when we are dealing with a rooted device. For this project, I personally used the method of removing the gesture.key file to bypass the lock screen. This is because in most cases, whereby when the device is already in the analyst custody, the lock files are valueless from a forensic point of view.

More complicated techniques could be used if the Android device is not rooted. A few examples of these are a physical dump of the memory chip and the use of some special hardware tools like Riff-Box and JIG-adapter.

Data Extraction on Android

Data Extraction is the process of retrieving data out of data sources for further data processing or data storage. Our project is performing data extraction on both the Android mobile device and the iPad and send to Splunk to process and analyze the retrieved data from the devices. For data extraction on Android, extract the data of all the files on the system or only those relevant files that you are interested in. But for any form of extraction, it’s important that the device is **unlocked or USB-debugging is previously enabled**.

There are many ways to extract data from an android device, ranging from creating a clone of the entire operating system to installing an application into the device and obtaining the relevant information using the ContentProviders which android used to share information along the various applications. A notable thing to mention is that extracting information using an application is very easy as there are much support and documentation provided by both the android operating system developers and the various developers from the community.

However, installing an application into the suspect’ device to extract information goes against our policy due to a possibility that the culprit might have already installed a **special application** in his device that is able to wipe all his **sensitive information** upon installation of any application.

Therefore, we shall look into extracting information through the ADB itself. There are many different exploits available for different android devices for different versions. It is important to note that one exploit is unable to be used for **all** android devices due to the different variants of android created by the various mobile device manufacturers.

For the current device I am working on, I notice that the device is already a CyanogenMod Google Nexus S device. Therefore a simple adb root command will allow me to obtain the root access for this device. For now, this will be enough for the exploit phase for this project as in the future, we will add various more exploits to work on a wider range of android devices.

Since we managed to obtain the temporary root of the device, we can obtain the various important databases to analyze on. For data extraction on Android, we have managed to obtain the Device Information previously, but now the current progress have allowed us to extract more than that which will be shown and explained below. An in-depth analysis of these databases have to be done due to a lack of documentation and support which shall also be explained below

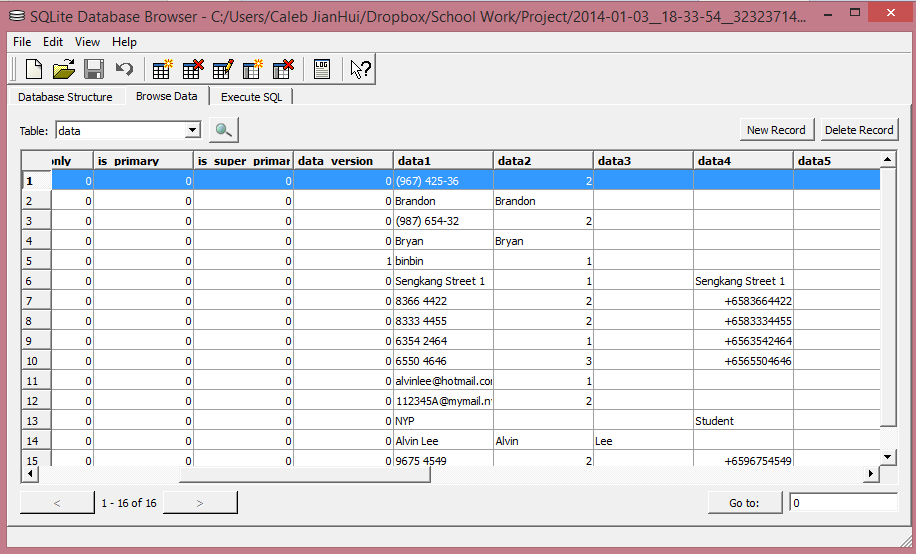
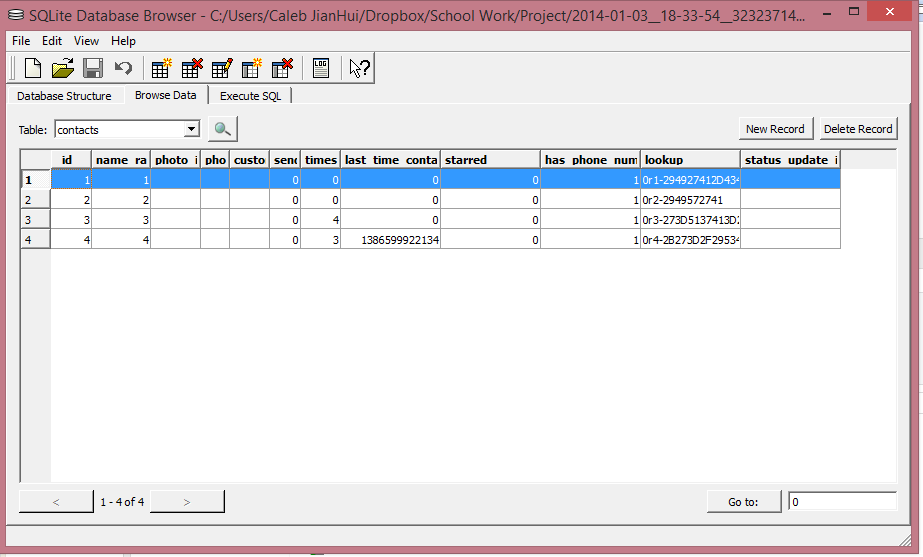
1. **Contacts**: The list of contacts found in the address book of the android device at the point of extraction.

Android device store their contact list at either contacts.db or contacts2.db depending on the version of the android operating system.

This file can be found in /data/data/com.android.providers.contacts/databases/ and extracted out from the device with a simple adb pull command. The structure of the database file is as shown below



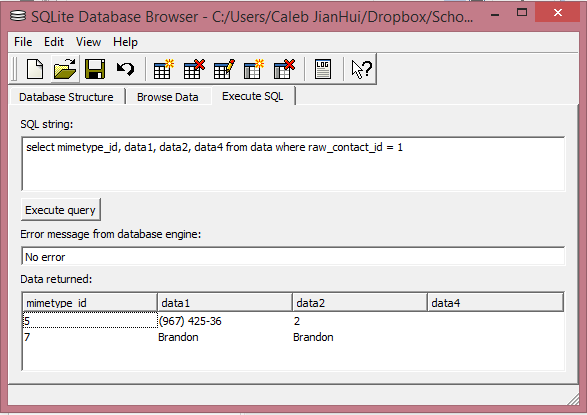
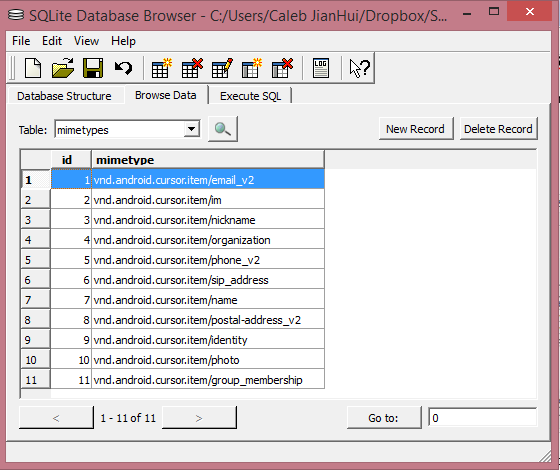
As you would have noticed, the structure of the contacts database is messy and require an in-depth analysis of the various tables in order to accurately obtain the crucial information for analysis. An example of this would be that the contacts table do not store any contacts information whereas the data table actually store several of the contacts information as shown below.

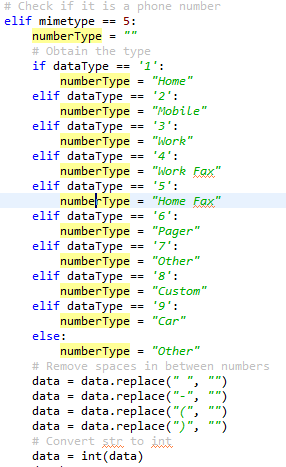
SQL query on this table have to be done in order to obtain the contact list from this device itself. A snippet of my code is shown below



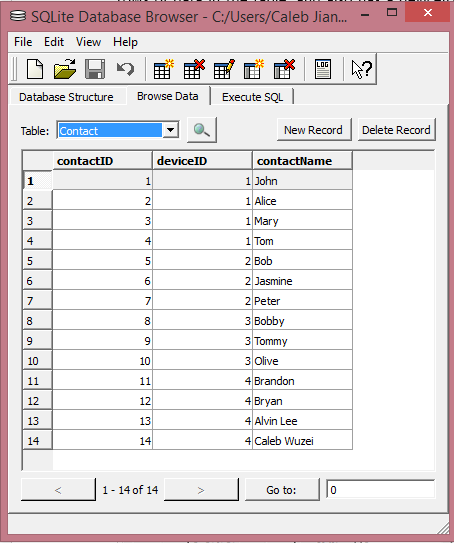
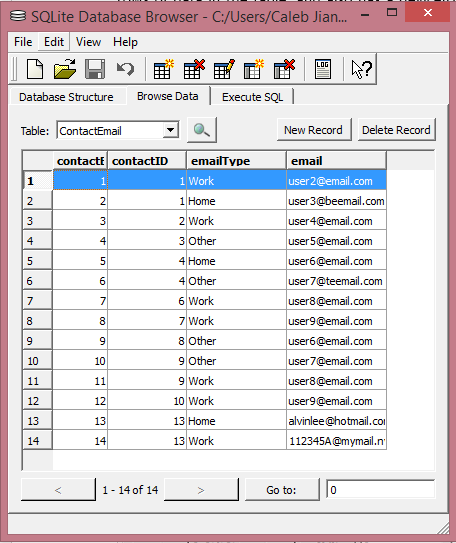
As shown above, I first query for the total amount of contacts from the contacts table. If the amount of contacts is more than zero, I go ahead to get the id and display name from the raw\_contacts table. For each id, I further query for an in-depth contact information. The mimetype\_id indicate what does the following data refer to. An example is shown below together with the list of what does the mimetype\_id represent from the mimetypes table.

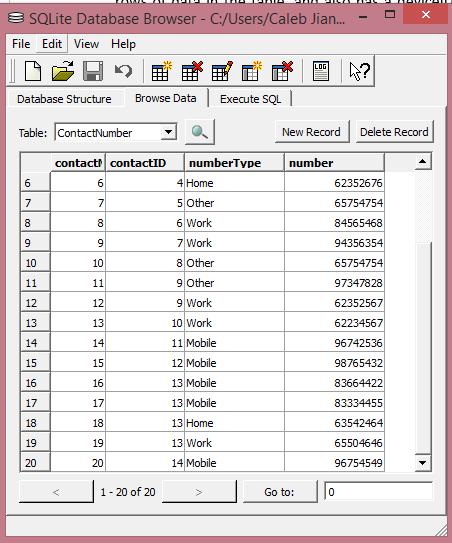
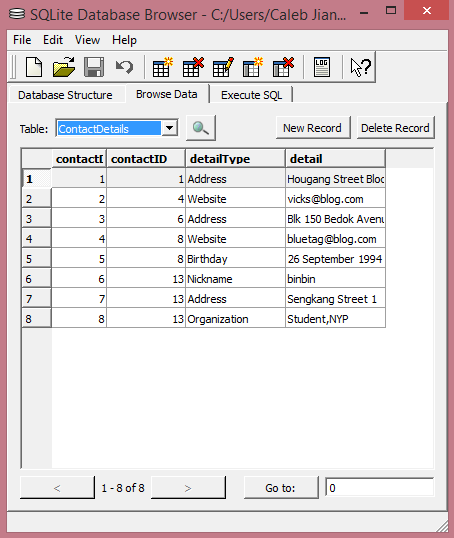
 

The data2 column will further explain what type of data it is, due to a lack of documentation, I have to do trial and error in order to ascertain the data type it is referring to. For example, for mimetype\_id of 5 which is referring to phone numbers, a data2 data of 2 refer to “Mobile” while 1 refers to “Home”. A snippet of my code is shown below to show how the logic works for the data2 (datatype) column



Our script will then add the various data into their relevant respective tables (**Contact**, **ContactEmail**, **ContactNumber**, **ContactDetails**) of **our integrated database** based on their mimetype id which Splunk will then collect information from. An example is shown below whereby our integrated database have received the information in their various tables from our python script for android.

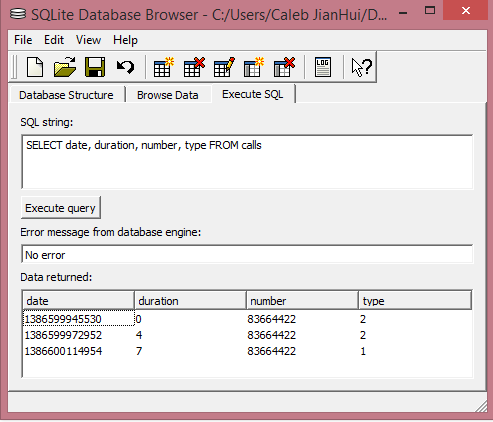
 

The contactID belongs to the **Contact** table and is a foreign key to the other three tables. The Contact table also contain a deviceID which is a foreign key to the **Device** table, indicating that this contact originate from a certain device in the **Device** table. The data of these tables are all in a human readable format for the user to easily understand the data.

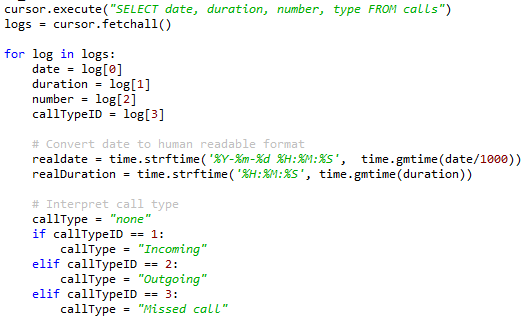
1. **CallLogs**: The list of call logs found on the android device at the point of extraction.

For call logs, android device store their call logs at either contacts.db or contacts2.db depending on the version of the android operating system which is similar to contact list as shown above.

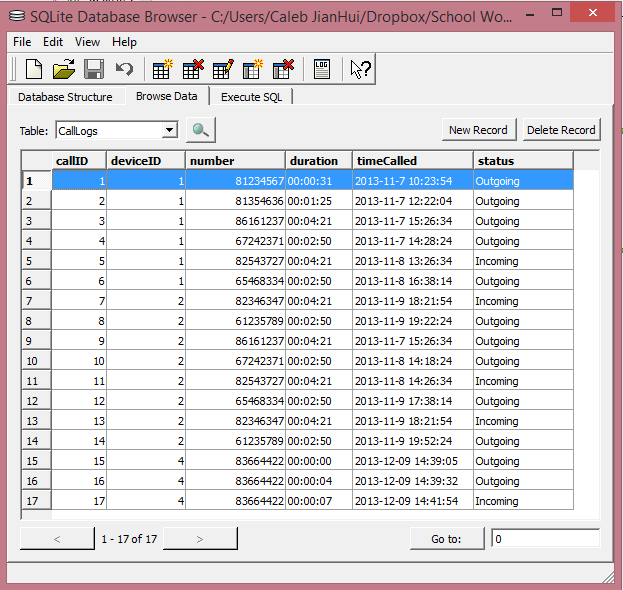
This file can be found in /data/data/com.android.providers.contacts/databases/ and extracted out from the device with a simple adb pull command. An SQL Query can be easily done to obtain the various information for the call logs as shown below.



After much analysis and testing, we are able to determine that the type refers to whether it is an incoming call or outgoing call. In addition, the date value is in milliseconds since epoch time and the duration is also in seconds. Our script will automatically convert it to a human readable format as shown below



After converting it to a human readable format, we will then export these data into **our integrated database** as shown below

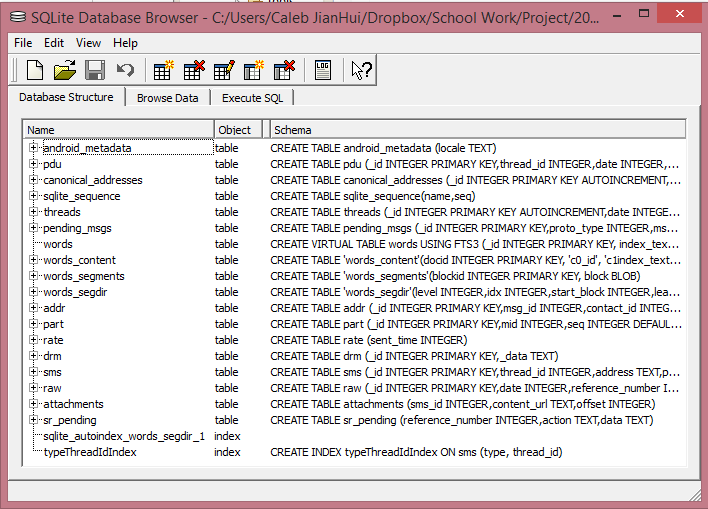


The CallLogs table also contain a deviceID which is a foreign key to the **Device** table, indicating that this call log originate from a certain device in the **Device** table. The number indicate the opposite party in a call. The duration, timeCalled and status are in a human-readable format to allow the user to understand the data easily.

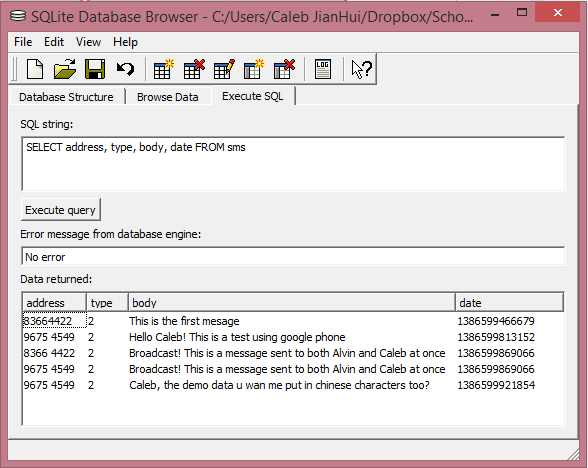
1. **SMSHistory**: The list of SMS found on the android device at the point of extraction.

For SMS, android device store their SMS list at mmsms.db

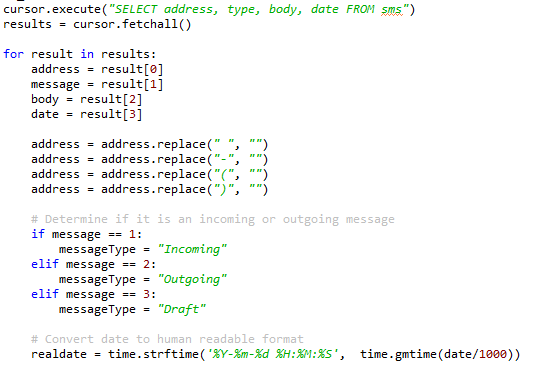
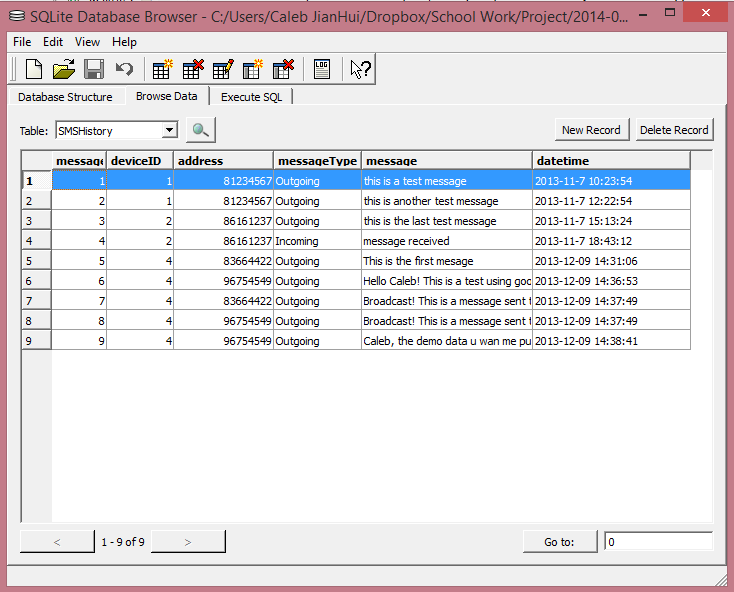
This file can be found in /data/data/com.android.providers.telephony/databases/ and extracted out from the device with a simple adb pull command. The structure of the database file is as shown below

After analysis of the database, the important tables that we are looking into are the threads and SMS tables. The thread table as shown above only contain the list of conversation threads which android probably used to arrange their messages. Therefore we will focus more on the SMS table. A simple SQL Query can be easily done to obtain only the important information for the list of SMS as shown below.



The address refers to the opposite party number regardless if it is an incoming or an outgoing message. Android does not seem to store the owner’s number when it comes to SMS. The type here refers to if it is an incoming or an outgoing or a draft message. The date is in milliseconds. Therefore our script will automatically convert all these information into a human readable format as shown on the bottom left. After formatting these data, our script will then export these data into **our integrated database** as shown on the bottom right.

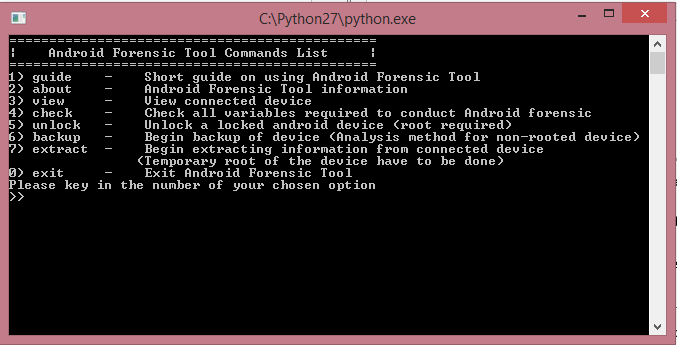
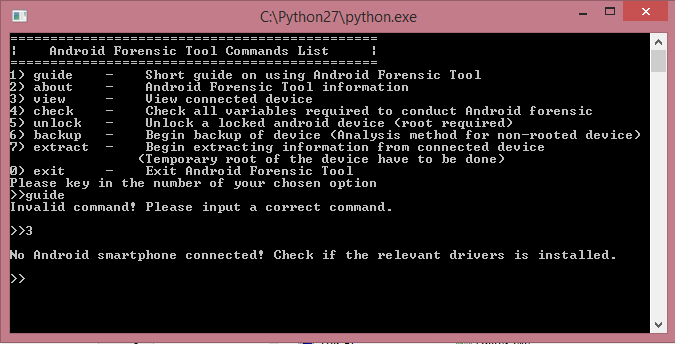
The SMSHistory table also contain a deviceID which is a foreign key to the **Device** table, indicating that this SMS originate from a certain device in the **Device** table. The address indicate the opposite party numbers in a SMS. The messageType, message and dateTime are in a human-readable format to allow the user to understand the data easily.

##### Project Enhancements

Other than performing data extraction on Android and researching of the relevant ways to obtain access to the android system, there were also several project enhancement done to improve the python script during this phase of the project.

1. **Ease of usage of Python script**

Being a python script aiming to allow automation of the forensic process for Android devices, we will have to constantly improve our python script to make it as easy to use as possible. This prompt us to create a simple menu for the user to view upon launching of our script as shown below

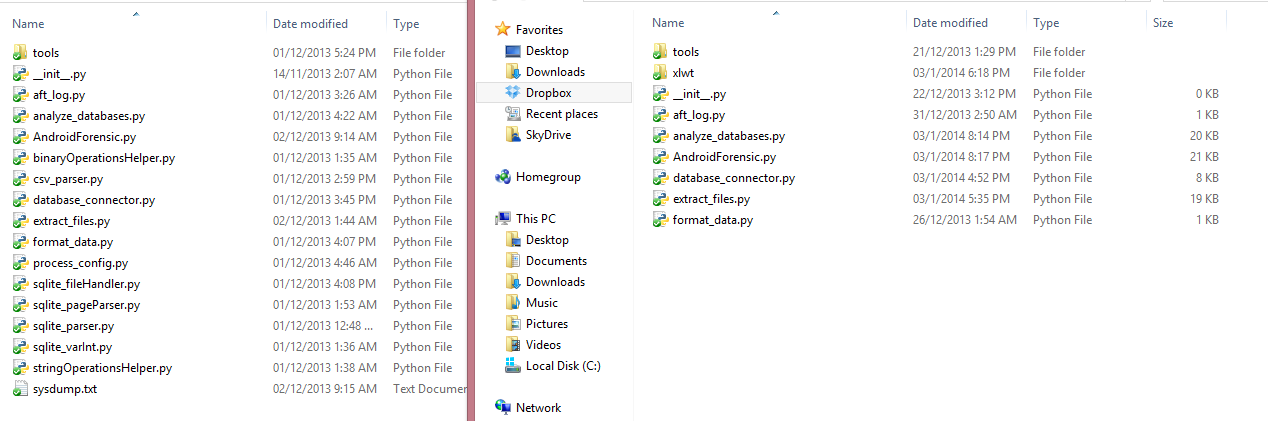
User now have to simply type the number of their choice in order to use this python script. This is shown below whereby the user no longer have to type ‘guide’ in order to use the program while a simple ‘3’ can view the connected device

1. **Segregation of task**

You might not have notice, but the backup process is no longer part of the extraction process. This is because backup might take up a lot of time if the analyst choose to allow the temporary rooting of the device itself. With temporary root of the device, more important information can be obtained from the device itself as shown above.

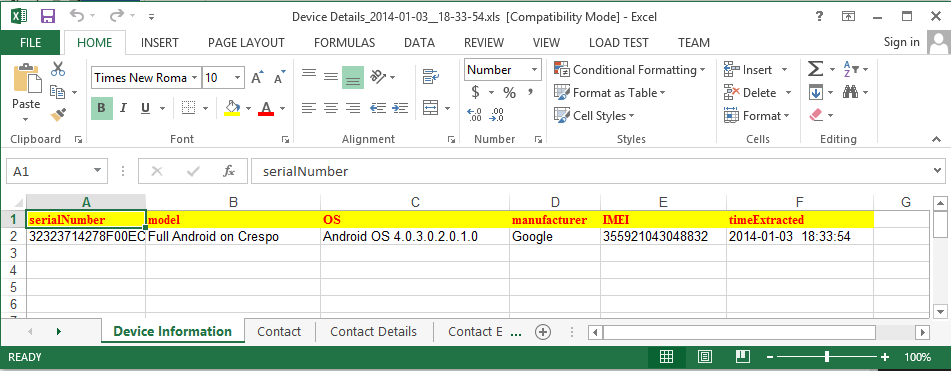
1. **Decrease of size of python script**

Although not a highly significant issue, our python script for Android have been rewritten to make it more compact and organized. This results in a smaller memory size of the entire python script as shown below where the many files previously have been slim down to only a few scripts. This is possible due to the python script no longer manually read the various databases and instead utilize the inbuilt database reader.



1. **Export to XLS**

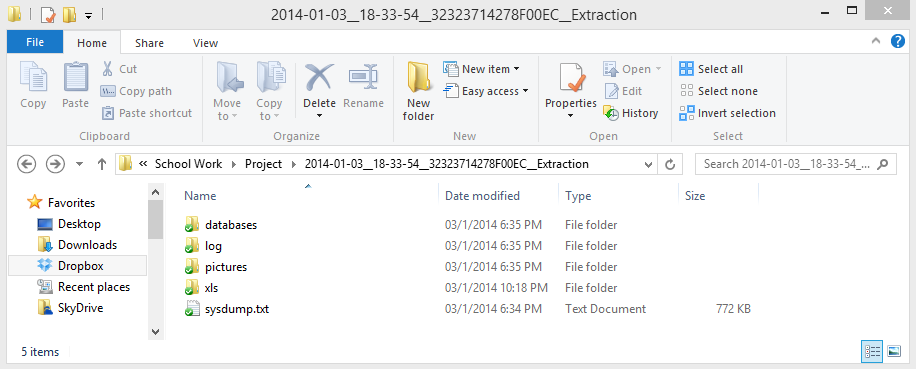
Export to XLS is a project enhancement feature allows a backup reference to the data that have been extracted out from the device. This allows the analyst to keep track of the records and data being extracted from the python script and exported into the integrated database. After the extraction process, our python script will generate an XLS file filename “Device Details\_” with the date and time concatenated behind. An example of the xls file is shown below



We will only show information related to this examination therefore showing only the important information that the user would like to see. The various information are kept in their pages to allow easy navigation between the various data without looking through multiple xls file.

1. **Better management of files for an examination**

Files used for an examination are better categorized and placed into their respective examination folder. An example is shown below



The databases folder will contain all the databases extracted from this device for this examination. The log folder will contain the logs for this examination as mentioned at the earlier phase of the project. The pictures folder will contain all the pictures found in this mobile device. The xls folder will contain the xls file created as mentioned above. All of this allows the analyst to obtain any information he need from this examination easily.

#### Python Script for IPad (Alvin)

##### Smudge Attack on iPad

A smudge attack is an attack that will recognize and find out the password pattern/PIN of a touchscreen mobile devices (such as iPad) by capturing the fingerprint left on the screen. Oily residues left by tapping fingers on a touchscreen may hence breach user privacy. Touch screens are an increasingly common feature on personal computing devices, especially smartphones, mobile devices. Oily residues, or smudges, on the touch screen surface, are one side effect of touches from which frequently used patterns such as a graphical password might be inferred.

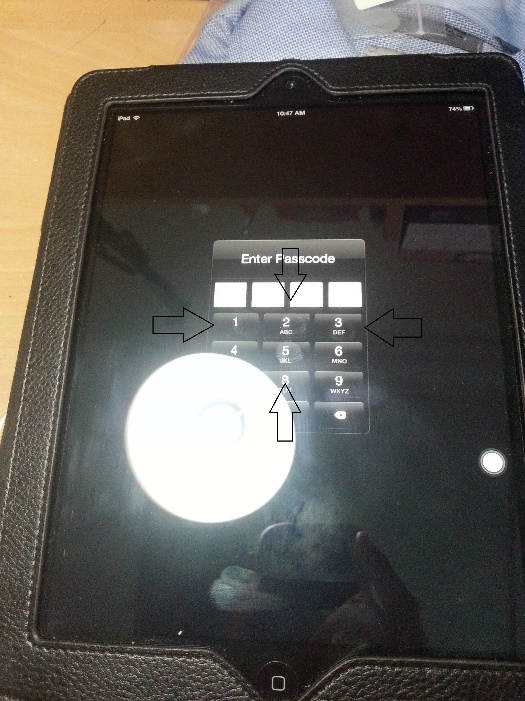
***Experiment & Preserving Fingerprints***

Once the evidence iPad was acquired, first step before any extraction or analysis of the iPad is to perform the smudge attack on the iPad. After the experiment, the password characters for login will be clearly disclosed. The touchscreen will not be able to work if the user was wearing gloves, it will require the user to use their fingers, hence oily residues, or smudges will be left on the touch screen surface, so as to allow the analyst to perform smudge attack on the iPad.

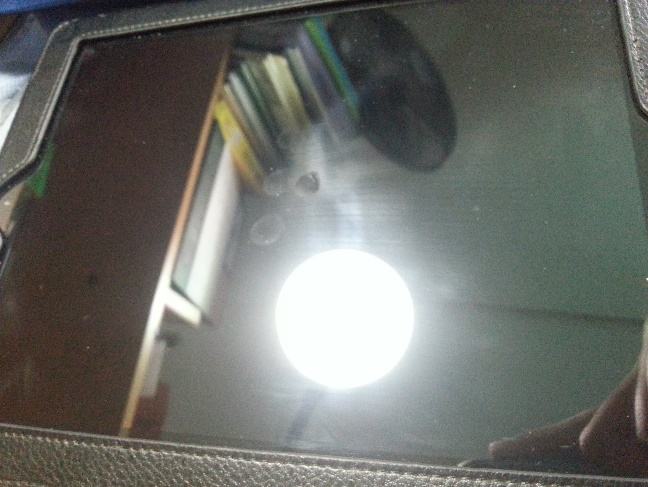
**\*Note:** All the following images on this section was taken by the camera of Samsung Galaxy S3.

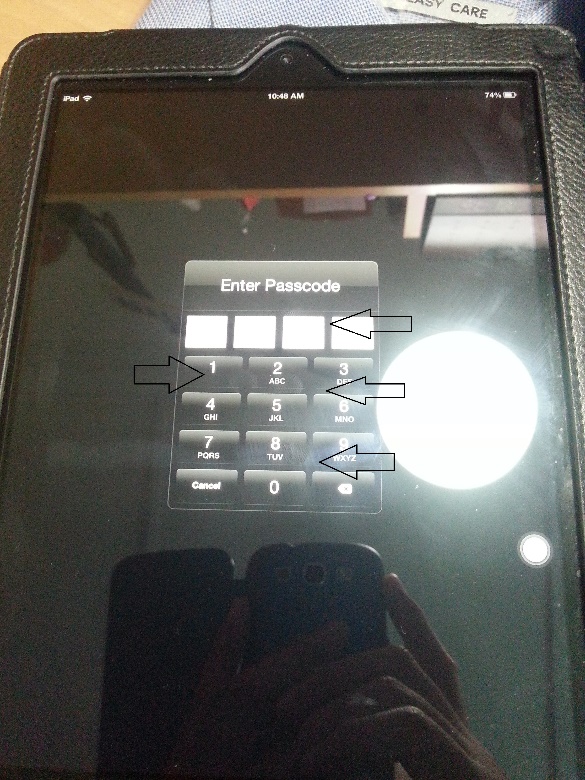
In the image taken above on the left, it is clearly that no visible fingerprints were identified on the iPad. Why is this so? Here we’re going to introduce an important element when conducting a smudge attack: Lightning. With the aid of lightning, the fingerprints will be shown onto the iPad clearly as seen from the image taken above on the right. As a result, 4 visible PIN Fingerprints were discovered.

Comparing both the images on both the left and right side above, once the iPad was prompt to enter the passcode to unlock the device, it is clearly shown that the 4 visible PIN Fingerprints matched nicely with the PIN numbers **[1, 2, 3, and 5].** As there are 4 different numbers, there will be 24 different permutations. This will be elaborate further on the ***Probability*** section below.

The image taken above on the left shows that the camera was moved to a different angle while the lightning remains the same. As you can see, there are no fingerprints found. However, on the image taken above on the right, the lightning was adjusted to a different angle too, hence, there were visible PIN Fingerprints discovered. This conclude that the lightning angles as well as the camera angles are also important elements when conducting smudge attack and preserving the fingerprints. Both the lightning angles and camera angles are also dependent on each other.

What about in this case as shown in the image taken above on the left? Does the 4 visible PIN Fingerprints match nicely with the PIN numbers **[2, 4, 5, and 8]?** However, no. This can be seen on the image taken above on the right, the 4 visible PIN Fingerprints don’t match nicely with the PIN numbers.

Once the iPad orientation was turned 90 degrees to the left as seen on the image taken above on the left, the 4 visible PIN Fingerprints matched nicely with the PIN numbers **[1, 2, 3, and 5]** as seen on the image taken above on the right. This shows that the user he previous logged in his credentials in this orientation. Of course, there are also 180 degrees and 270 degrees orientation to be talk about.

***Probability***

Once the password characters for login are clearly disclosed, all the analyst have to do now is to brute-force the iPad passcode with all the permutations available (maximum: 24) from the visible PIN Fingerprint(s). For iPad, after the password characters for login are clearly disclosed, what would then be the probability of getting the correct 4-number PIN in first try?

|  |  |  |
| --- | --- | --- |
| No. of visible PIN Fingerprint(s) | No. of Permutation(s) | Probability (%) |
| 1 | 1 | 100% |
| 2 | 14 | 7.14% |
| 3 | 78 | 1.28% |
| 4 | 252 | 0.4% |

The table above shows that different number of visible PIN Fingerprint(s) found in the device will determine the number of permutation(s) available, and hence, difference in the probability of getting the correct PIN in the first try. The number of visible PIN Fingerprint(s) can also be known as the number of different digits found on the device.

**\*Note:** The number of permutation does not include repeating numbers (Example: 1111, 2222, etc.), except for 1 visible PIN Fingerprint.

***Limitations***

**#1:** User enters a wrong PIN for the first time, example PIN numbers **[1, 2, 3, and 4],** and then re-enter the correct PIN numbers **[1, 2, 3, and 5]**. This will hence resulting in more than one visible PIN Fingerprints, which will cause the number of permutations to increase and hence, increase the difficultly and probability of getting the correct PIN.

|  |  |  |
| --- | --- | --- |
| No. of visible PIN Fingerprint(s) | No. of Permutation(s) | Probability (%) |
| 5 | 620 | 0.16% |
| 6 | 1290 | 0.08% |
| 7 | 2394 | 0.04% |
| 8 | 4088 | 0.02% |
| 9 | 6552 | 0.015% |
| 10 | 9990 | 0.01% |

The table above shows what will be the number of permutation(s) and the probability of getting the correct PIN in the first try if there are more than 4 visible PIN Fingerprints found when conducting the smudge attack on the iPad.

**\*Note:** The number of permutation does not include repeating numbers (Example: 1111, 2222, etc.), except for 1 visible PIN Fingerprint.

**#2:** The iPad does not only have 4-PIN password input to access the device, it also has an option for the user to input a string of alpha-numeric symbol characters to improve the security of their device. However, the 4-PIN password input is common among iPad users as it is strong enough to be brute-force and not so complicated, easier for users to remember, and convenience for the user to login at a faster rate as compared to the string input password.

**#3:** If the iPad user is cautious enough, it is also possible that the user will wipe off the smudges each time he/she enters the 4-PIN inputs to avoid smudge attack to be conducted once their iPad device was captured.

**#4:** After the user logins onto his/her iPad, he/she continues to use it, example playing fruit ninja, which will create many more other non-related smudges. Hence, it will destroy the fingerprints for login credentials fully, partially, or create more non-related fingerprints which disguise as the login credentials, hence distract the analyst to perform smudge attack more accurately.

***Other methods?***

Dusting is the process to use the finger-print powder to dust the touch screen surface to reveal fingerprints, and use a camera (the higher resolution the better) to photograph fingerprints. After that will then sharpen the fingerprints in an image via various image processing techniques and design effective algorithms to automatically map fingerprints to a keypad in order to infer tapped passwords. The following video link below shows the dusting process on iPad:

<http://www.youtube.com/watch?v=6jS6KroER3Y>

The image below shows that the password characters for login are clearly disclosed after the dusting process as shown in the video link above:



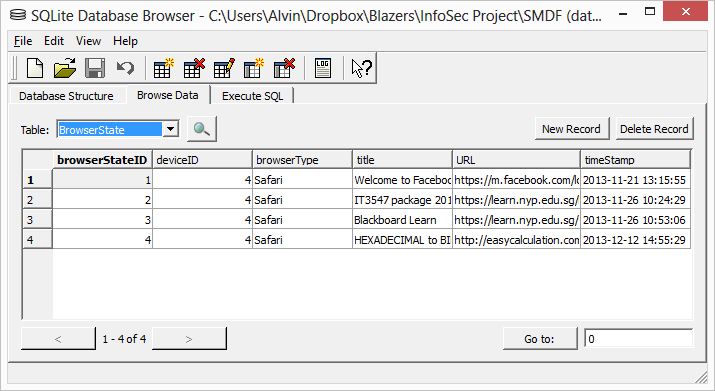
##### Data Extraction on iPad

Data Extraction is the process of retrieving data out of data sources for further data processing or data storage. Our project is performing data extraction on both the Android mobile device and the iPad and send to Splunk to process and analyze the retrieved data from the devices. For data extraction on iPad, our project has successfully extracted out 2 of the categories, Device Information and Browser History (Safari) which were done previously on the first Implementation Update of the project. The current Implementation Update of the project has successfully extracted out another 9 categories of data, resulting in a total of 11 main categories of data extracted out from the iPad itself. The 9 categories of data extracted out in this phase of project are:

1. **BrowserState:** Ii The currently opened tab(s) in the browser (Safari) at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **BrowserState** table are:

1. Type of the browser, example Safari (***browserType*** Column)
2. Title of the webpage (***title*** Column)
3. URL (***URL*** Column)
4. Last Visited Date time (***timeStamp*** Column)

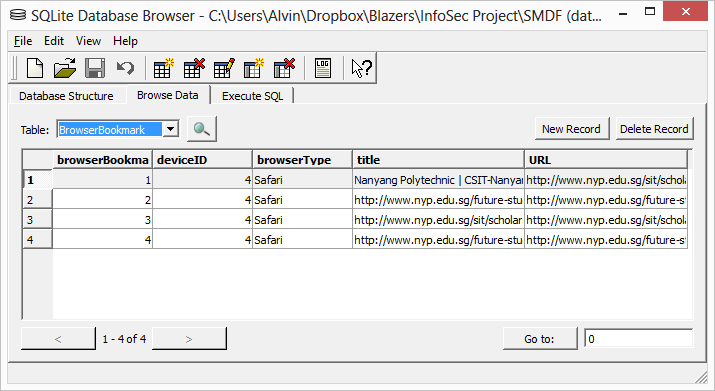


The **BrowserState** table also has a browserStateID column which is the ID of each individual rows of data in the table, and also has a deviceID which is foreign key to the **Device** table, indicating whether this row of data belongs to which device in the **Device** table.

1. **BrowserBookmark:** The list of favourites/bookmarks found in the browser (Safari) at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **SafariBookmark** table are:

1. Type of the browser, example Safari (***browserType*** Column)
2. Title of the webpage (***title*** Column)
3. URL (***URL*** Column)

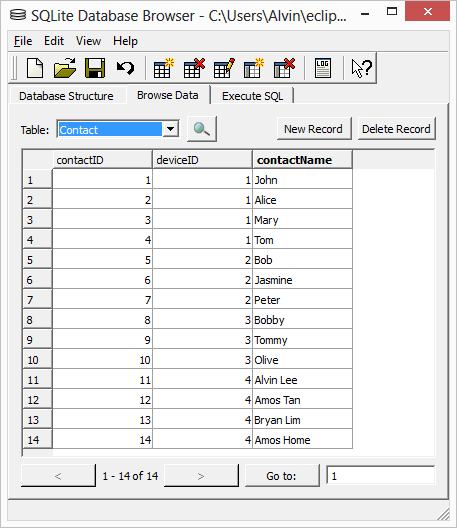


The **BrowserBookmark** table also has a browserBookmarkID column which is the ID of each individual rows of data in the table, and also has a deviceID which is foreign key to the **Device** table, indicating whether this row of data belongs to which device in the **Device** table.

1. **Contact:** The list of contacts found in the address book of the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **Contact** table are:

1. Full Name of the contact (***contactName*** Column)

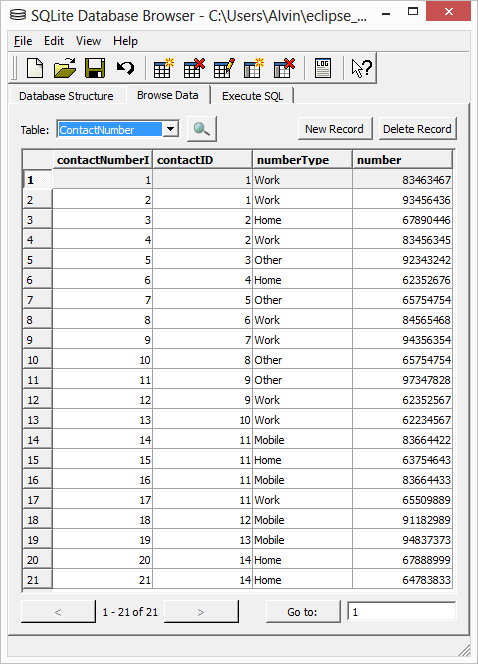


The **Contact** table also has a contactID column which is the ID of each individual rows of data in the table. This contactID will be the foreign keys for three of the tables (ContactNumber, ContactEmail, and ConactDetails). The **Contact** table also has a deviceID which is foreign key to the **Device** table, indicating whether this row of data belongs to which device in the **Device** table.

1. **ContactNumber:** The list of contact numbers contains in each of the contacts found in the address book of the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **ContactNumber** table are:

1. Type of the number, example Work, Home, etc. (***numberType*** Column)
2. Number of the contact (***number*** Column)

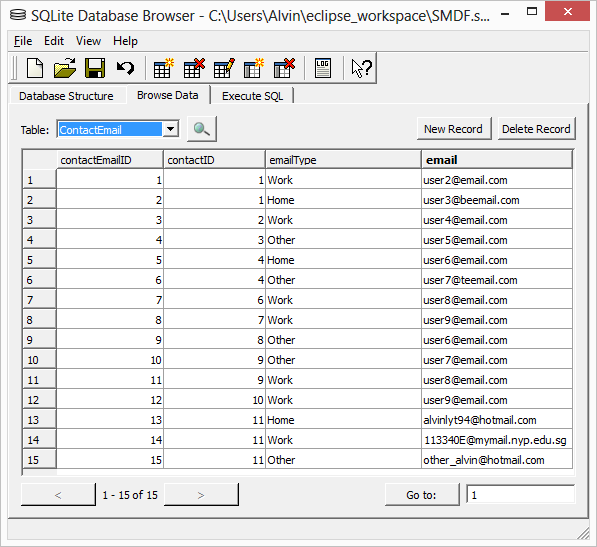


The **ContactNumber** table also has a contactNumberID column which is the ID of each individual rows of data in the table, and also has a contactID which is foreign key to the **Contact** table, indicating whether this row of data belongs to which contact in the **Contact** table.

1. **ContactEmail:** The list of contact emails contains in each of the contacts found in the address book of the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **ContactEmail** table are:

1. Type of the email, example Work, Home, etc. (***emailType*** Column)
2. Email of the contact (***email*** Column)

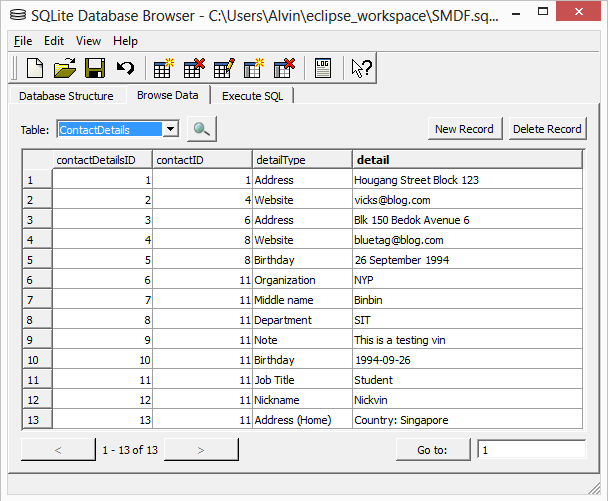


The **ContactEmail** table also has a contactEmailID column which is the ID of each individual rows of data in the table, and also has a contactID which is foreign key to the **Contact** table, indicating whether this row of data belongs to which contact in the **Contact** table.

1. **ContactDetails:** The list of other contact details contains in each of the contacts found in the address book of the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **ContactDetails** table are:

1. Type of the email, example Address, Birthday, etc. (***detailType*** Column)
2. Detail for each type of details of the contact (***detail*** Column)

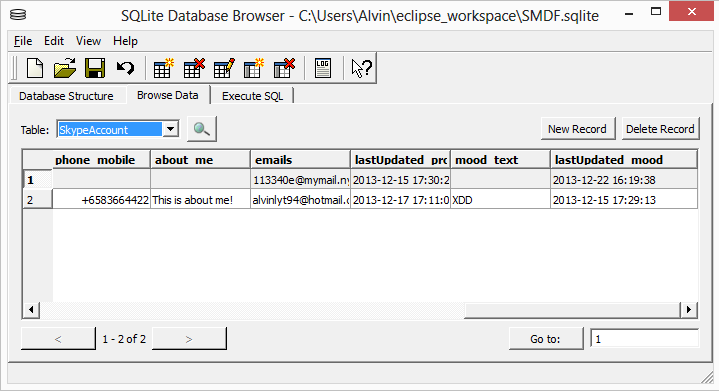
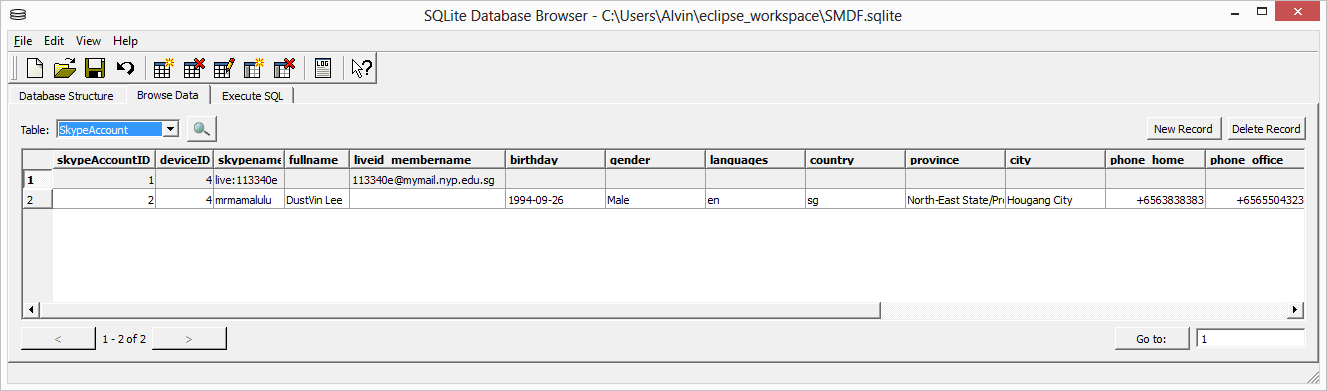


The **ContactDetails** table also has a contactDetailsID column which is the ID of each individual rows of data in the table, and also has a contactID which is foreign key to the **Contact** table, indicating whether this row of data belongs to which contact in the **Contact** table.

1. **SkypeAccount:** The list Skype accounts that had been successfully logged in before on the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **SkypeAccount** table are:

1. Skype name of the Skype Account (***skypename*** Column)
2. Full name of the Skype Account (***fullname*** Column)
3. Microsoft Live ID of the Skype Account (***liveid\_membername*** Column)
4. Birthday of the Skype Account (***birthday*** Column)
5. Gender of the Skype Account (***gender*** Column)
6. Languages of the Skype Account (***languages*** Column)
7. Country of the Skype Account (***country*** Column)
8. Province of the Skype Account (***province*** Column)
9. City of the Skype Account (***city*** Column)
10. Home Phone number of the Skype Account (***phone\_home*** Column)
11. Office Phone number of the Skype Account (***phone\_office*** Column)
12. Mobile Phone number of the Skype Account (***phone\_mobile*** Column)
13. Personal detail more about the Skype Account (***about\_me*** Column)
14. Emails of the Skype Account (***emails*** Column)
15. Last Updated Date Time of the Skype Account’s Profile (***lastUpdated\_profile*** Column)
16. Personal Mood detail of the Skype Account (***mood\_text*** Column)
17. Last Updated Date Time of the Skype Account’s Mood (***lastUpdated\_mood*** Column)

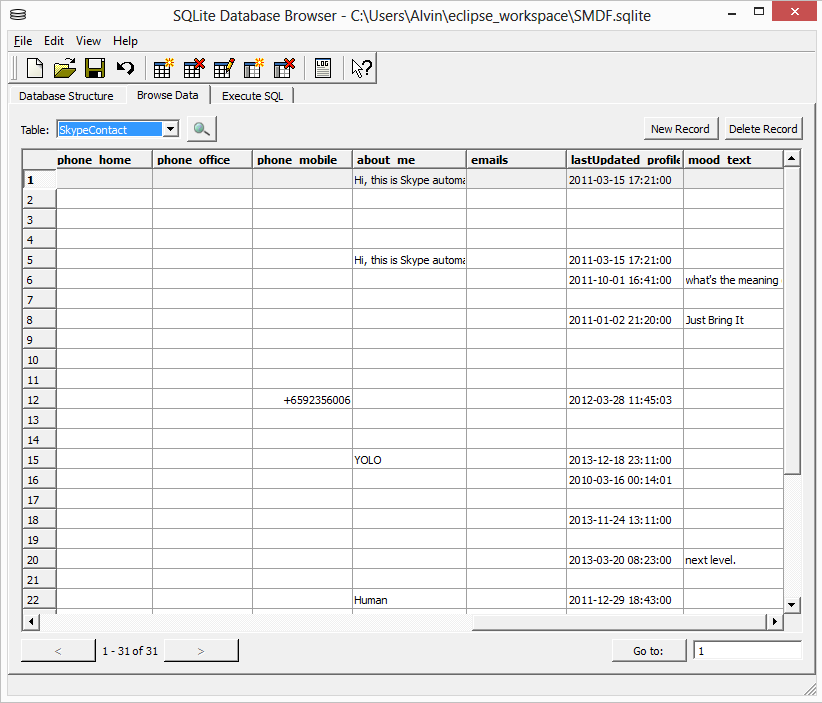
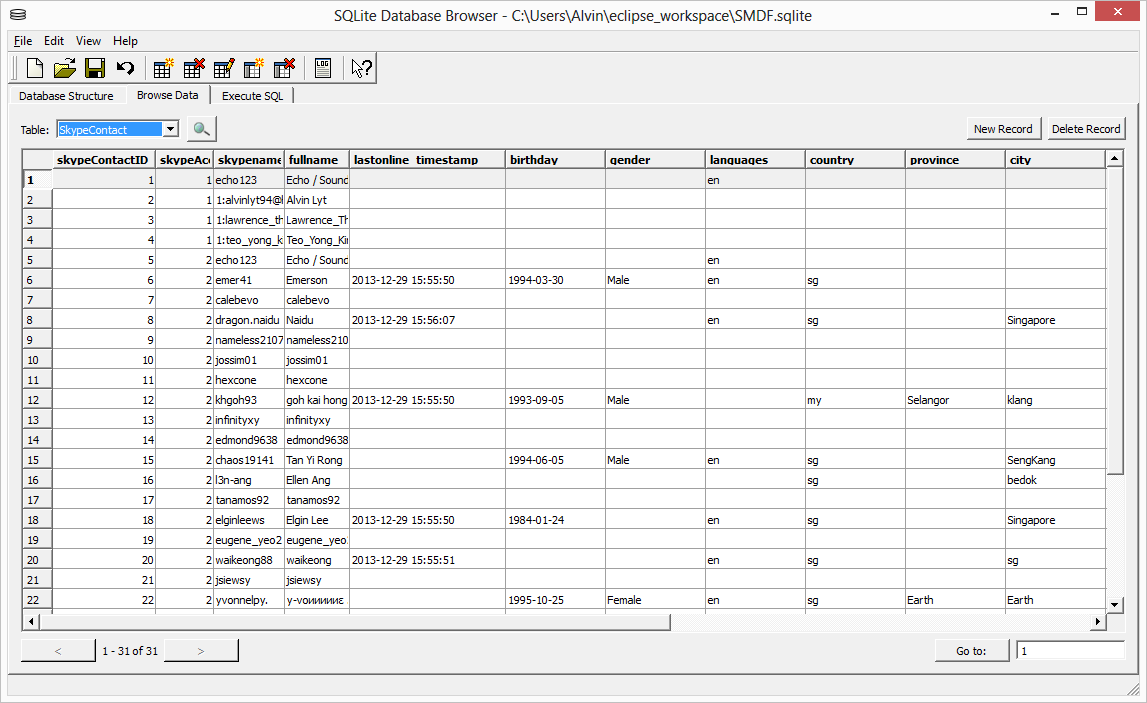


The **SkypeAccount** table also has a skypeAccountID column which is the ID of each individual rows of data in the table. This skypeAccountID will be the foreign keys for two of the tables (SkypeContact and SkypeConversation). The **SkypeAccount** table also has a deviceID which is foreign key to the **Device** table, indicating whether this row of data belongs to which device in the **Device** table.

1. **SkypeContact:** The list of Skype Contacts contain in each of the Skype accounts that had been successfully logged in before on the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **SkypeContact** table are:

1. Skype name of the Skype Contact (***skypename*** Column)
2. Full name of the Skype Contact (***fullname*** Column)
3. Last Online Date Time of the Skype Contact (***lastonline\_timestamp*** Column)
4. Birthday of the Skype Contact (***birthday*** Column)
5. Gender of the Skype Contact (***gender*** Column)
6. Languages of the Skype Contact (***languages*** Column)
7. Country of the Skype Contact (***country*** Column)
8. Province of the Skype Contact (***province*** Column)
9. City of the Skype Contact (***city*** Column)
10. Home Phone number of the Skype Contact (***phone\_home*** Column)
11. Office Phone number of the Skype Contact (***phone\_office*** Column)
12. Mobile Phone number of the Skype Contact (***phone\_mobile*** Column)
13. Personal detail more about the Skype Contact (***about\_me*** Column)
14. Emails of the Skype Contact (***emails*** Column)
15. Last Updated Date Time of the Skype Contact’s Profile (***lastUpdated\_profile*** Column)
16. Personal Mood detail of the Skype Contact (***mood\_text*** Column)

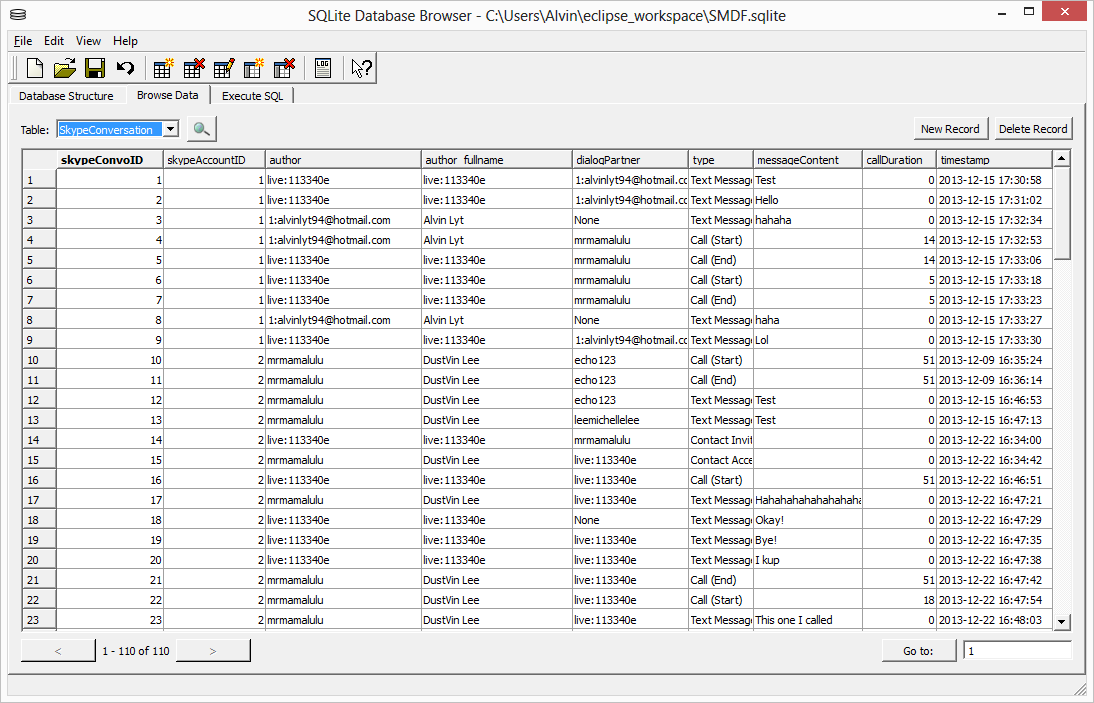


The **SkypeContact** table also has a skypeContactID column which is the ID of each individual rows of data in the table, and also has a skypeAccountID which is foreign key to the **SkypeAccount** table, indicating whether this row of data belongs to which Skype account in the **SkypeAccount** table.

1. **SkypeConversation:** The list of Skype Conversations contain in each of the Skype accounts that had been successfully logged in before on the iPad at the point when the iTunes backup are performed.

The list of data that can be extracted out from the **SkypeConversation** table are:

1. Skype name of the author who initiate the Skype Conversation (***author*** Column)
2. Full name of the author who initiate the Skype Conversation (***author\_fullname*** Column)
3. Dialog Partner of the Skype Account in the Skype Conversation (***dialogPartner*** Column)
4. Type of Skype Conversation, examples Text Message, Call (Start), Contact Invitation, etc. (***type*** Column)
5. Content of the message from the author in the Skype Conversation (***messageContent*** Column)
6. Call Duration between the Skype Account and the dialog partner (***callDuration*** Column)
7. Date Time of the events in the Skype Conversation (***timestamp*** Column)



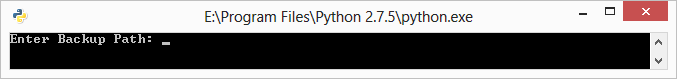
The **SkypeConversation** table also has a skypeConvoID column which is the ID of each individual rows of data in the table, and also has a skypeAccountID which is foreign key to the **SkypeAccount** table, indicating whether this row of data belongs to which Skype account in the **SkypeAccount** table.

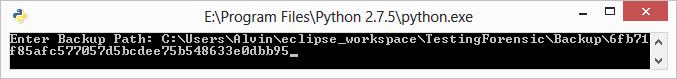
##### Project Enhancements

Besides performing data extraction on iPad and researching of smudge attack on iPad, there are also some of the project enhancements that were done in this phase of the project.

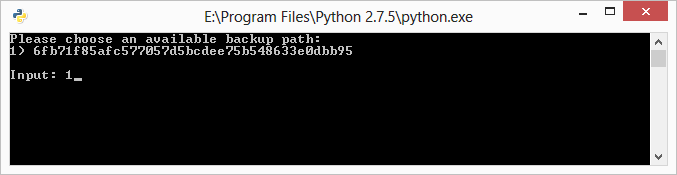
1. **Ease usage on selecting backup path**

Previously before the enhancement of this feature, the analyst have to copy and paste the backup path onto the scripts each time the analyst wanted to do analysis of the backup. However, it is not very convenient and user-friendly to do it such way. Sometimes the clipboard will be lost or overwritten with something else, hence pose more inconvenience for the analyst to re-copy the backup path again onto the clipboard.





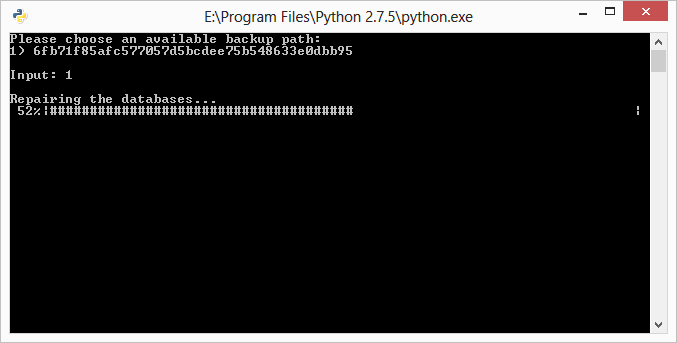
The pictures above shows the before (top) the analyst paste the backup path onto the console, and the after (bottom) the analyst paste the backup path. It also does not look professional in this way. Hence, enhancement for this is a must. The image below shows after the enhancement of this feature, where the analyst just have to select the option of the available backup path. Hence, no more copy-paste of backup path, more user-friendly, and more convenience for the analyst.



In order for the backup path option to be shown in the list in the console, the analyst will have to do a one-time copy of the original backup path from the iTunes backup path (*C:\Users\%User%\AppData\Roaming\Apple Computer\MobileSync\Backup\6fb71f85afc577057d5bcdee75b548633e0dbb95*) into the *Backup* folder in the current working directory. The python scripts will then read the available backup path in the *Backup* folders and list out for the analyst to select.

1. **Progress Bar**

Without the progress bar, analyst will not have any idea when will the process completes and unable to estimate the duration of the progress. Hence, it is not very convenience for the analyst. Now after the implementation of the progress bar, it allows the analyst to be able to keep track on the progress of the process and estimate the duration till it was done. The image below shows the progress bar in command line view:

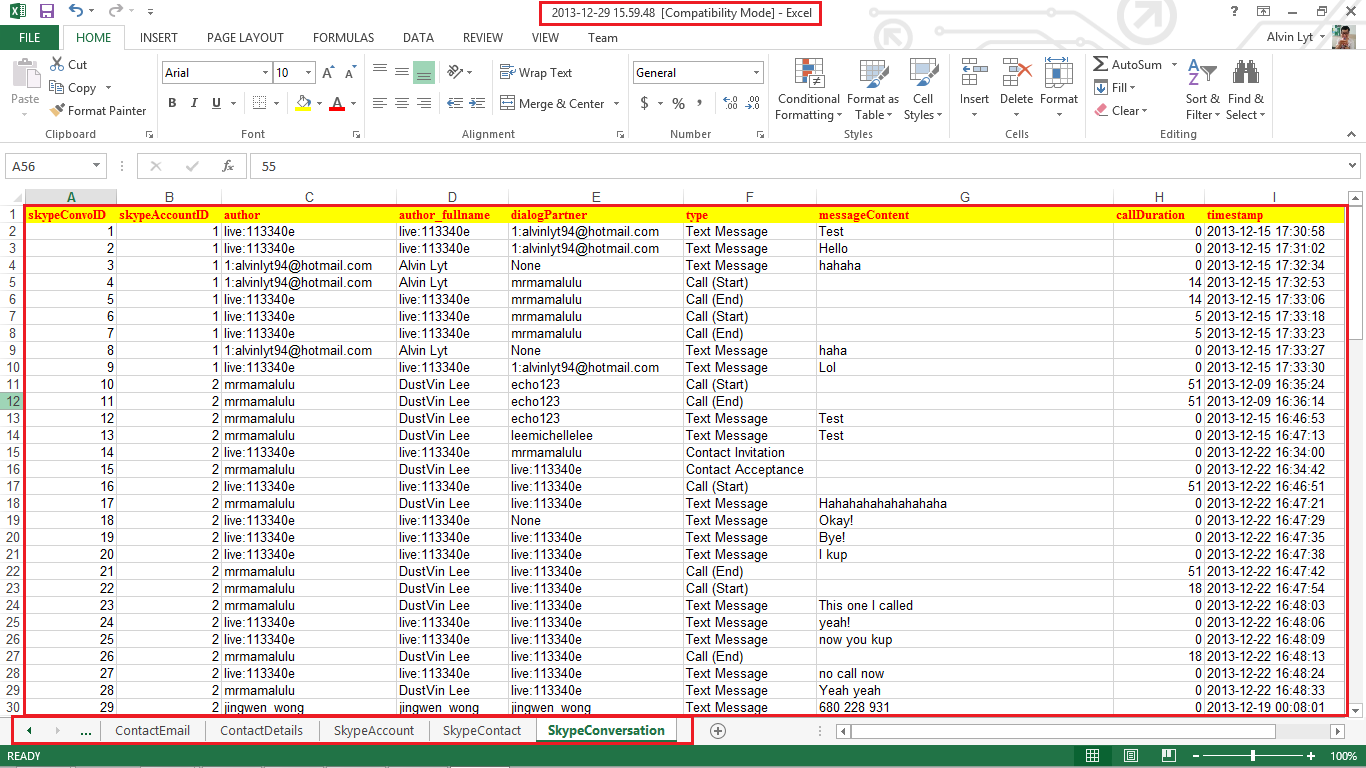
****

The image below shows the progress bar in the console view of Eclipse:

****

1. **Export to XLS**

Export to XLS is a project enhancement features which serve as a backup reference of the data being exported out into SQLite database. The exported XLS file will allow the analyst to keep track of the records and data being extracted out from the python scripts. After the extraction process, the python script will generate an XLS file with filename of the date time of the extraction onto the current working directory. The following is the example of what will be contained inside the XLS file:

****

The image above shows the content of the XLS file exported out after the extraction process. On the top of the image indicates the filename of the XLS file, which is the date time in this particular format (*YYYY-mm-DD HH:MM:SS*). In the middle section, it shows the data rows extracted out from the SQLite database with columns on the very first row (yellow foreground and red color text). Below shows all the sheets generated and each sheet indicate each table from the SQLite database. However, not all tables in the SQLite database were created, only required tables related to iPad data extraction will be exported out into the XLS file.

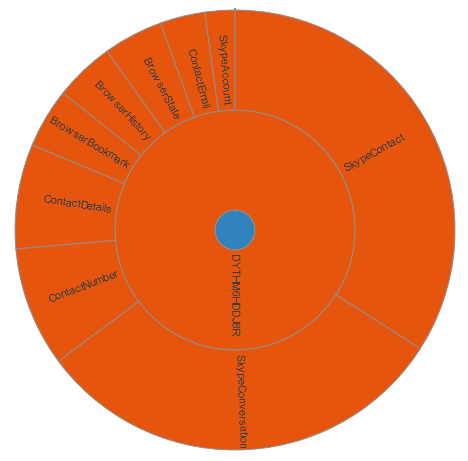
#### Splunk (Joshua)

##### Overview Dashboard



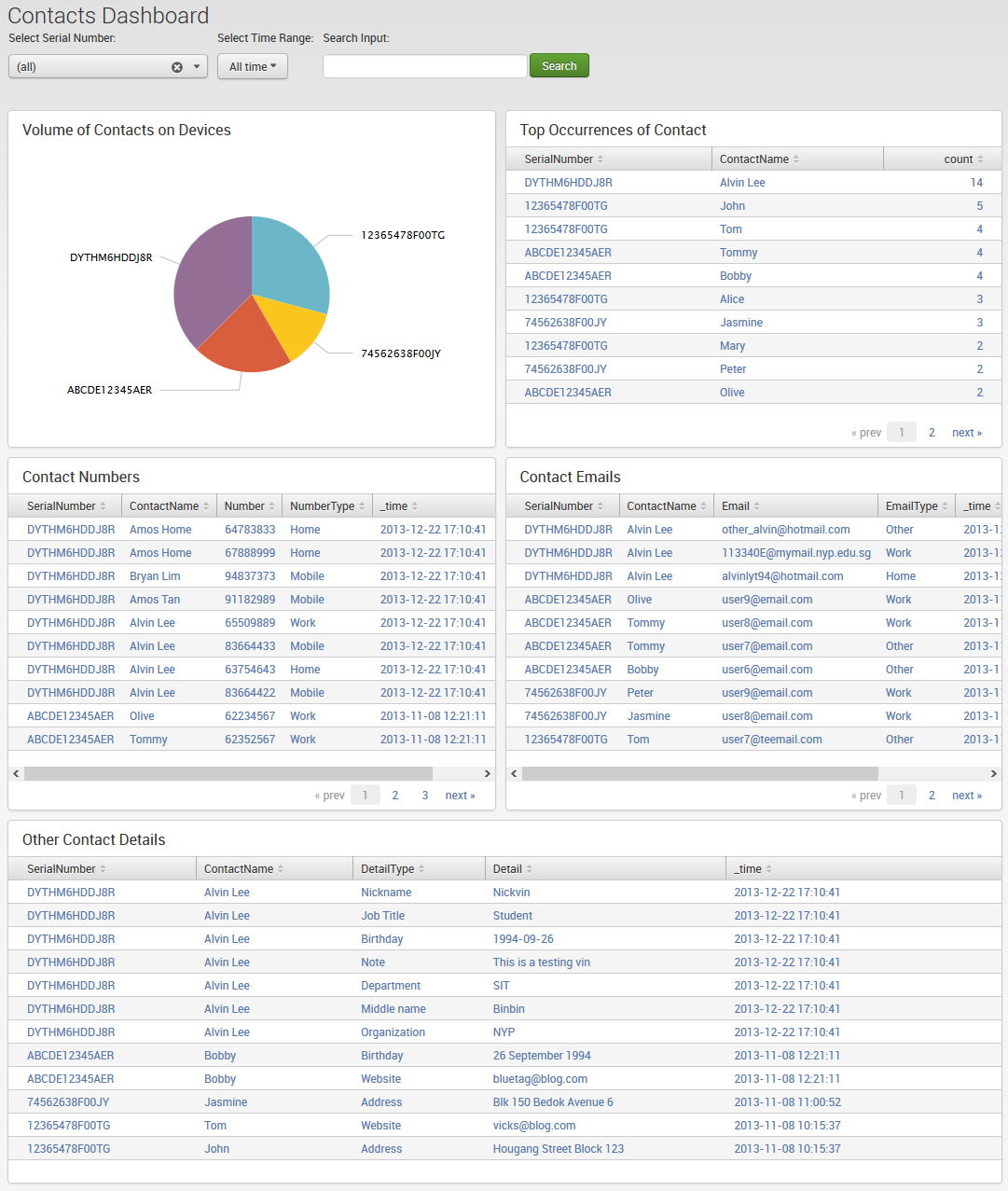
This is the updated Overview Dashboard. I have replaced the pie chart with a D3 sunburst chart. The previous pie chart only showed the amount of events for each device.

The D3 sunburst chart is able to expand on this functionality and provide another layer of representation; for each device it shows the volume of each source type.

This allows the analyst to easily comprehend the amount of data extracted per device and how much data is extracted per source type.

The sunburst chart has an interactive element as well, if the analyst selects a portion in the chart, it will zoom into that portion and show the leaf nodes in greater detail. The sunburst chart on the right shows the result of zooming into a mobile device.

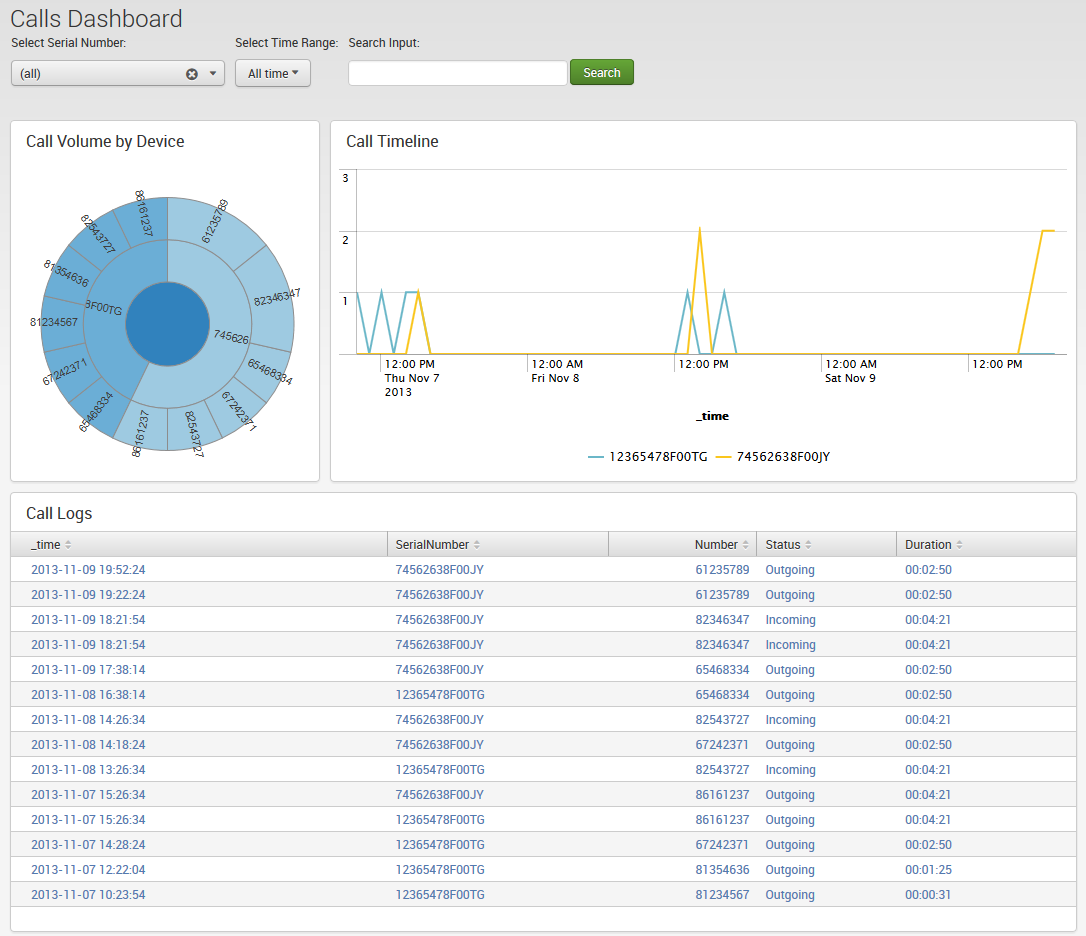
##### Contacts Dashboard



This is the updated Contacts Dashboard.

I have added two new charts at the top, a pie chart showing the volume of contacts per device and a table chart showing the contact that has the most events.

##### Calls Dashboard



This is the calls dashboard.

The sunburst chart at the top left shows the volume of calls to a number on each mobile device.

The time line chart on the right shows the amount of calls made during a period of time.

The table chart on the bottom shows each call made on a device, sorted from latest to oldest calls.

##### Browser Habits Dashboard

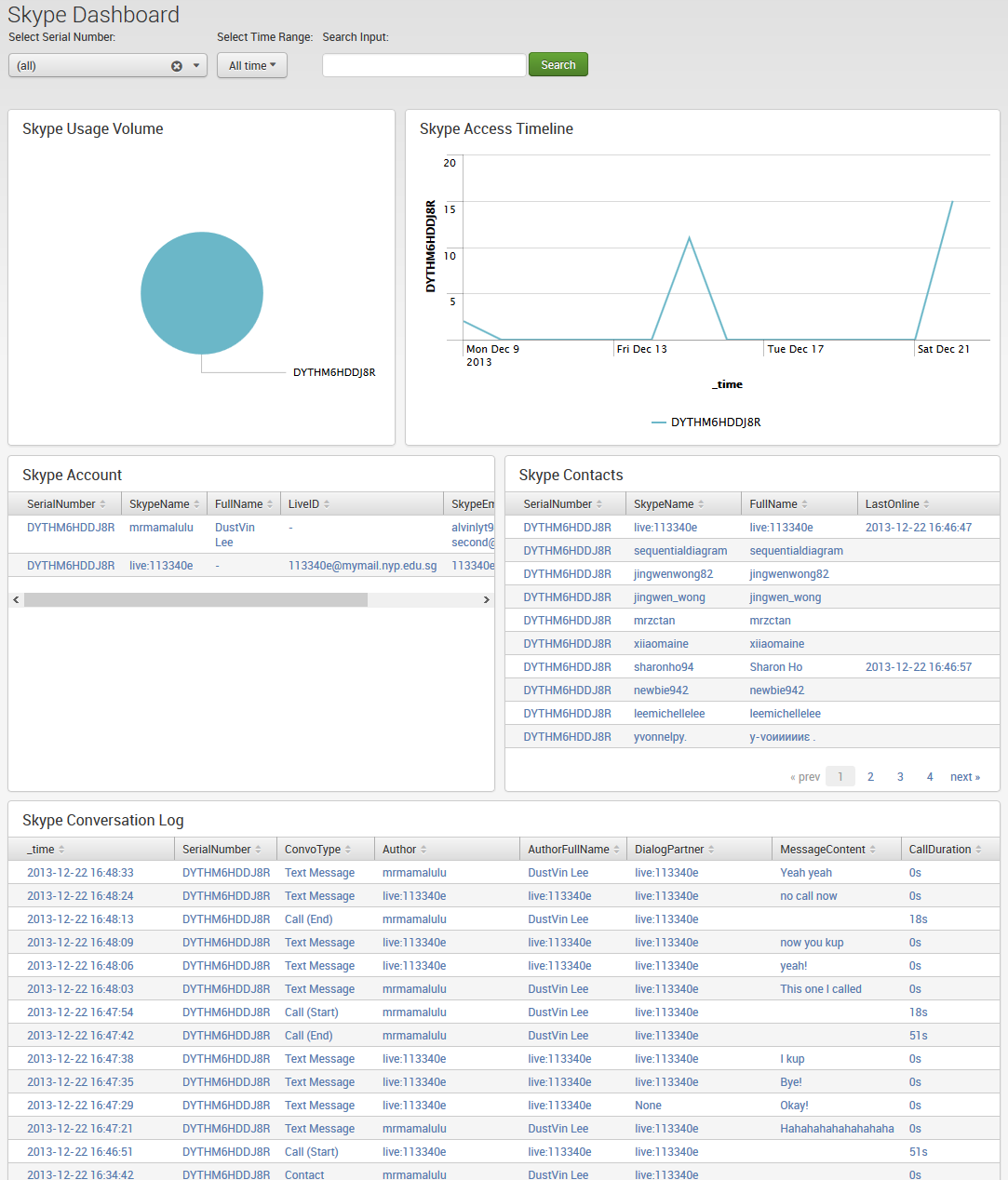


This is the Browser Habits dashboard.

The sunburst chart at the top left shows the volume of web access activity per device. The timeline chart on the top right shows the web activity count per device during a period of time.

The second row shows the most visited domains and the URLs that were last visited on the device. The third row shows the browser tabs that were opened at the point of data extraction. The fourth row shows the websites that were bookmarked on the device.

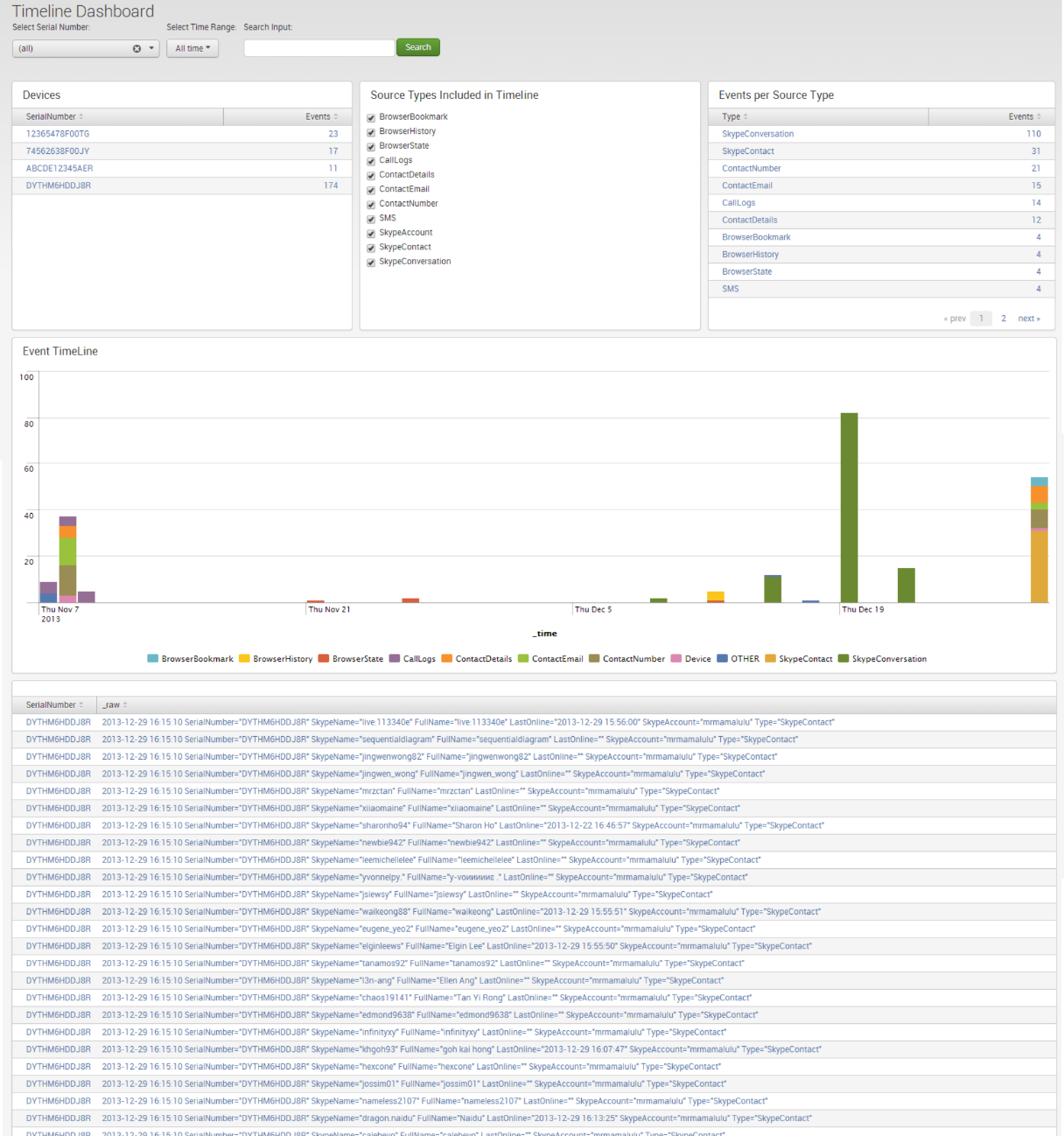
##### Skype Dashboard



This is the Skype Dashboard.

The first row has a pie chart showing the volume of Skype events on a device, a timeline chart shows the usage activity during a period of time. The second row shows the Skype accounts belonging to the owner of the device and the Skype contacts extracted from the device. The third row shows the detailed Skype Conversation Log between the device and the dialog partner.

##### Timeline Dashboard

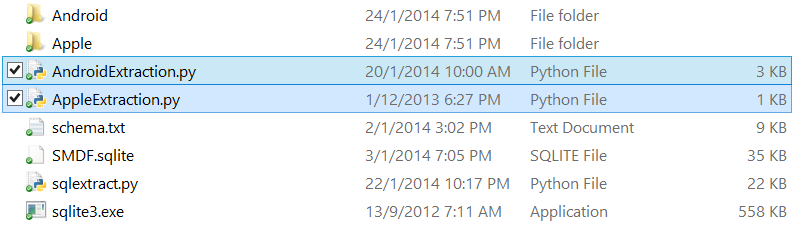


This is the Timeline Analysis dashboard.

On the left and right on first row of charts is the amount of events per device and amount of events per source type. In the middle, the user is presented a check list of source types, by default all source types are included in the timeline. If the user wishes to remove one or more source types from the timeline, all he has to do is to uncheck the source types and the timeline analysis will automatically remove it from the charts below.

The stacked view shows the volume of events that each source type occupies. Hovering over a source type legend will bring to focus all occurrences of that source type.

## 5.4 Final System



The user first has to stop the Splunk server and transfer the app into the apps folder.

Once the user transfers the **SplunkforMobileDeviceForensics** app into the Splunk apps folder, he can proceed on to the extraction process.

We have simplified the process so that the user only has to execute either the *AndroidExtraction.py* or *AppleExtraction.py* python script - depending on which platform the user wishes to extract from. The scripts used for Android and Apple extraction are located in their respective folders. This is done because we want the user to immediately see the extraction options that are available and not confuse them by requiring them to dive into the Android or Apple folders.

After extraction is completed, the user starts the Splunk Server and allow a moment to read the SMDF.sqlite database.

The user can access the app by opening the web browser and navigating to **localhost:8000** page.   
After logging in, the user will select the SplunkforMobileDeviceForensics app and the user will be redirected to the Overview page. The user can then proceed to the various dashboards to further analyse the data that has been extracted from the device.

# 6. Test Plan and Test Report (Indicate individual contributions)

## 6.1 Test Setup

#### Caleb’s

1. Connect the android device which you would like to do examination on
2. Ensure the relevant drivers for that android device is already installed in the machine
3. Enter the number corresponding to the options available which will appear upon launching of the script
4. Use test case 3 to verify if the device is successfully connected

#### Alvin’s

1. Skip Step 2 to Step 4 if performing Test Case 1a
2. Perform iTunes Backup onto the plugged iPad in the workstation
3. Go to this directory: *C:\Users\Alvin\AppData\Roaming\Apple Computer\MobileSync\Backup*
4. Copy the iTunes Backup Path into the *Backup* folder in the working directory: C:\Users\Alvin\eclipse\_workspace\TestingForensic\Backup
5. In the current working directory: *C:\Users\Alvin\eclipse\_workspace\TestingForensic* , run the ReadBackup.py file and a command prompt window will pop up

#### Joshua’s

##### Fresh install

1. Stop Splunk instance
2. Copy SplunkforMobileDeviceForensics folder into the apps folder of Splunk.
3. Run Android/Apple extraction scripts
4. Start Splunk instance
5. Log into Splunk through the web browser
6. Select the Splunk for Mobile Device Forensics app
7. Navigate the app through the Navigation bar at the top

## 6.2 Test Cases

#### Caleb’s

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Running the script | | |
| **Description:** | The analyst run the script to proceed on for the extraction process. | | |
| **Primary Actors:** | Analyst | **Executed By:** | Caleb |
| **Preconditions:** | Nil | **Date Executed:** | 3-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 9:30pm |
| **Setup:** | *Tools* folder must exist in the working directory for Test Case 1 to Pass | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Analyst run the script | * User ensures that the device is connected successfully * User double click on the python script * Commands list for the python script is displayed to the user * User types 7 and press enter to run the device | Connecting to device….  ====== Forensic Examination Information ======  Begin extraction of data, please wait...  .  . | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Analyst run the script | No device is connected | Error! No Android smartphone connected!  Check if the relevant drivers is installed  Terminating Android Forensic Tool… | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Selecting an option in the menu | | |
| **Description:** | Allow analyst to select an option available in the menu shown to the user upon launch of python script | | |
| **Primary Actors:** | Analyst | **Executed By:** | Caleb |
| **Preconditions:** | Nil | **Date Executed:** | 3-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 9:40 pm |
| **Setup:** | *Tools* folder must exist in the working directory for Test Case 2 to Pass | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Selecting an option | Input: “7” | Connecting to device….  ====== Forensic Examination Information ======  Begin extraction of data, please wait...  .  . | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Selecting an option | Input: “this is a string” | Invalid command! Please input a correct command.  >>> | **Pass** |
| 1b. Selecting an option | Input: “9” (Commands are up to 0-8) | Invalid command! Please input a correct command.  >>> | **Pass** |
| 1c. Selecting an option | Input: “0” (0 is a command to exit the python script) | Script terminates | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Ensure device is connected | | |
| **Description:** | Allow analyst to verify if the android device is successfully connected | | |
| **Primary Actors:** | Analyst | **Executed By:** | Caleb |
| **Preconditions:** | Nil | **Date Executed:** | 3-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 9:28 pm |
| **Setup:** | *Tools* folder must exist in the working directory for Test Case 2 to Pass | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Selecting an option | Input: “4” (Relevant drivers is successfully installed) | Checking all required conditions  =====================  Connected device: …. | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Selecting an option | Input: “4” (no drivers is installed) | Checking all required conditions  =====================  Error! No Android smartphone connected! Check if the relevant drivers is installed  >>> | **Pass** |

#### Alvin’s

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Running the script | | |
| **Description:** | The analyst run the script to proceed on for the extraction process. | | |
| **Primary Actors:** | Analyst | **Executed By:** | Alvin |
| **Preconditions:** | Nil | **Date Executed:** | 21-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 11:14pm |
| **Setup:** | *Backup* folder must exist in the working directory for Test Case 1 to Pass | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Analyst run the script | There are 1 iTunes Backup Path in the *Backup* folder in the current working directory | Please choose an available backup path:  0) Enter backup path  1) 6fb71f85afc577057d5bcdee75b548633e0dbb95  Input: | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Analyst run the script | No iTunes Backup Path in the *Backup* folder in the current working directory | No available backup path in the following directory: C:\...\TestingForensic\Backup  Enter backup path: | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Selecting an iTunes Backup Path | | |
| **Description:** | Allow analyst to select an iTunes Backup Path from the *Backup* folder in the current working directory. Once a valid Backup Path was requested, the script will start extraction process of the Backup folder. | | |
| **Primary Actors:** | Analyst | **Executed By:** | Alvin |
| **Preconditions:** | Nil | **Date Executed:** | 21-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 11:14pm |
| **Setup:** | *Backup* folder must exist in the working directory | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Selecting an iTunes Backup Path | Input: “1” (assuming there are at least 1 iTunes Backup Path found) | “Decoding Backup Path...”  …  Extraction Process will be started. | **Pass** |
| 2. Selecting an iTunes Backup Path | Input: “0” (assuming there are at least 1 iTunes Backup Path found) | Enter backup path: | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Selecting an iTunes Backup Path | Input: “this is a string” | Invalid Input! Please re-enter!  Input: | **Pass** |
| 1b. Selecting an iTunes Backup Path | Input: “2” (while there are only 1 iTunes Backup Path found) | Index out of range! Please re-enter!  Input: | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Input an iTunes Backup Path | | |
| **Description:** | Allow analyst to input an iTunes Backup Path manually. Once a valid Backup Path was requested, the script will start extraction process of the Backup folder. | | |
| **Primary Actors:** | Analyst | **Executed By:** | Alvin |
| **Preconditions:** | Nil | **Date Executed:** | 21-1-2014 |
| **Version/Build #:** | Version 1 | **Time Executed:** | 11:14pm |
| **Setup:** | *Backup* folder must exist in the working directory | | |
| **Configurations:** | Windows OS, Python 2.7.5 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Input an iTunes Backup Path | Input: “C:\...\TestingForensic\Backup\6fb71f85afc577057d5bcdee75b548633e0dbb95” (assuming this backup path was a valid backup path) | “Decoding Backup Path...”  …  Extraction Process will be started. | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1a. Input an iTunes Backup Path | Input: “this is a string” | Invalid iTunes backup folder or backup path not found! | **Pass** |

#### Joshua’s

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | Installing Splunk for Mobile Device Forensics app | | |
| **Description:** | Installing Splunk app | | |
| **Primary Actors:** | Analyst | **Executed By:** | Joshua |
| **Preconditions:** | Splunk 6 must be installed | **Date Executed:** | 3-1-2014 |
| **Version/Build #:** | 0.4 | **Time Executed:** | 10pm |
| **Setup:** | Complete installation of Splunk 6 | | |
| **Configurations:** | Windows OS, Python 2.7.5, Splunk 6 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-Main | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Analyst transfers the app into the apps folder of Splunk | 1. Stop Splunk 2. Transfer a clean app into Splunk\etc\apps 3. Start Splunk | App appears in the application list of Splunk main page | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| SMDF-1 | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Analyst reinstalls the app and clears data | 1. Stop Splunk 2. Delete old app in Splunk\etc\apps 3. Transfer a clean app into Splunk\etc\apps 4. Execute “splunk clean eventdata -index smdf” 5. Start Splunk | App is restored back to default and data within the index is cleared | **Pass** |

## 6.3 Summary of Test Results and Corrective Actions

Test Case results done internally by project members have gone successfully. Input from supervisor on increasing the user experience by giving more options to the user has been included, e.g. increasing ease of access for the Apple Extraction scripts.

We have actively structured the user experience to be as simple as possible. We developed the application in such a way that the choices presented to the user is clear and understandable. This involved incorporating automation processes to the extraction functions, in such a way that the user is only required to complete a few steps. Splunk was a natural choice for data analytics as it is both powerful and easy to use, synergizing well with our application concept.

We await to conduct the User Acceptance Test by the external team. After which we will take into account user feedback and implement corrective actions.

## 6.4 Counter Test Cases (SOCKS)

#### Norman Part

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-1 Get Data | | |
| **Description:** | Client device retrieves relevant data to be sent to server. Server receives and generate files. Process transparent to end user. | | |
| **Primary Actors:** | User | **Executed By:** | Joshua |
| **Preconditions:** | Application have to be installed, service running | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:10pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-1 Get Data | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Service executes every 30 seconds | Data is sent over to server | Server creates respective files. Process transparent to users | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-2 All Chat/One-to-One Chat | | |
| **Description:** | User sends a text message. All clients should receive and see it on their app, under “Everyone” | | |
| **Primary Actors:** | User | **Executed By:** | Joshua |
| **Preconditions:** | Application have to be installed, service running | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:20pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-2 All Chat | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. User types in message in textbox and click ‘Send’ | Data is received by all clients and displayed on the app | All clients see the message | **Pass** |

#### Kian Lon Part

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-3 Get Device Location | | |
| **Description:** | Get the last known location of the device | | |
| **Primary Actors:** | User | **Executed By:** | Caleb |
| **Preconditions:** | Application have to be installed | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:25pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-3 Get Device Location | Procedure / Inputs | Expected Results | Pass/Fail |
| Click on get location button | 1. Start the application 2. Click on get Device location buttons | The coordinate of the device will be shown | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-4 Device Lockdown Prototype | | |
| **Description:** | Remotely lock the device | | |
| **Primary Actors:** | User | **Executed By:** | Caleb |
| **Preconditions:** | Application have to be installed  Administrative rights need to be enabled | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:35pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-4 Device Lockdown Prototype | Procedure / Inputs | Expected Results | Pass/Fail |
| Lock the device with administrative rights | 1. Start the application 2. Click on device remote lock down button | Device will be locked and user need to unlock with password | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-5 Get Photo | | |
| **Description:** | Get a photo through the front camera from the device | | |
| **Primary Actors:** | Admin | **Executed By:** | Caleb |
| **Preconditions:** | Admin need to login to the website  Application have to be installed | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:35pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-5 Get Photo | Procedure / Inputs | Expected Results | Pass/Fail |
| Click on get photo button | 1. Click on a device in all device page 2. Click on get Photo button | The photo will be shown | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-6 Splunk android application | | |
| **Description:** |  | | |
| **Primary Actors:** | Admin | **Executed By:** | Caleb |
| **Preconditions:** | Splunk have to be installed  Administrative rights need to be enabled | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:45pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-6 Splunk android application | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Admin access to the Splunk login portal | Username: admin  Password: Pa$$w0rd | Dashboard of charts with data retrieved from user device will be shown | **Pass** |
| 1. Splunk Device event bar chart | Click on the one of the device event bar chart | Dashboard of charts of that specific device will be shown | **Pass** |

#### Soh Jason Part

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-7 Text-based command communication link | | |
| **Description:** | The commands that are sent through the communications link between the client phone and the server | | |
| **Primary Actors:** | Admin | **Executed By:** | Alvin Lee |
| **Preconditions:** | Application have to be installed  Service must be running  Server must be running | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 5:50pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-7 Text-Based communication link | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. Root Command execution | !runcmd df  Where df is the command to be run as root. | Text after the !runcmd is executed as a command under root and its results are returned to the application | **Pass** |
| 1. Ping command | !ping google.com  where google.com is the targeted host. | If the host is alive, “success” will be returned to the application. Otherwise, “failed” will be returned to the application. | **Pass** |
| 1. List applications detail and status | !applistdetail | The list of applications and their state “running” or “stopped” will be returned to the applicaiton | **Pass** |
| 1. Launch application | !applaunch com.android.settings | The targeted application will be launched | **Pass** |
| 1. Stop application | !servicekill  com.android.settings | The targeted application will be stopped and deleted from running and cached memory. | **Pass** |
| 1. Shutdown phone | !shutdown | The phone will be shut down. Under emulation, the emulator will be frozen. | **Pass** |
| 1. File Transfer | File is converted to BASE64 and split into strings of 8000 characters long.  @Server fileContent%filename%BASE64string  @Server !fileStop%filename%BASE64string | !fileContent will save the string to memory.  After!filestop is received, the associated fileContent will be combined together and a file will be created on the server. | **Pass** |
| 1. Authentication | !Authenticate user,password | The server will return to the application in the form of “true,employees,user”  First item will be success of the authentication.  The subsequent items are the groups the user is in. | **Pass** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:** | TC-UC-8 Windows Active Directory Authentication and App restriction | | |
| **Description:** | Link Windows Active Directory credentials and obtain information about the user. | | |
| **Primary Actors:** | User | **Executed By:** | Alvin Lee |
| **Preconditions:** | Application have to be installed  Server must be running  Windows Server must be running | **Date Executed:** | 5-2-2014 |
| **Version/Build #:** | 2.1.3 | **Time Executed:** | 6:00pm |
| **Setup:** | - | | |
| **Configurations:** | - | | |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-UC-8 Windows Active Directory Authentication and App restriction | Procedure / Inputs | Expected Results | Pass/Fail |
| 1. User authentication | User enters his username and password on the phone | Applications that are restricted in the user’s group will be disabled on the phone. The user cannot launch them. | **Pass** |
| 1. Admin authentication | Admin enters his username and password on the phone | All applications will be enabled. | **Pass** |

# 7. Conclusion

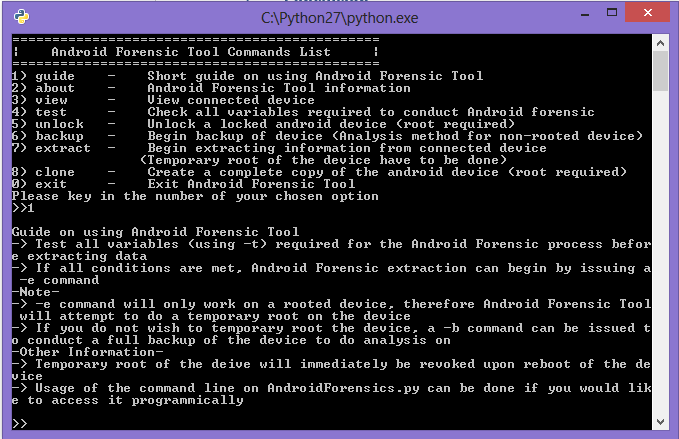
## 7.1 Accomplishments

Caleb

I have successfully created a lightweight python script which aids the analyst in extracting important information from the android device. The entire python script is only 9Mb and with the use of python script, it allows cross-platform usage of this script.

A notable feature in our python script is that everything is done through the Android Debugging Bridge (adb). We choose not to develop an application to extract the data as installing an application in the android mobile device might tamper with the data in the android device. Therefore we do not install any application to decrease the percentage of tampering the data in the android device.

As this script is mostly automated with little interaction with the user, I have also implemented a guide to aid new users in using our python script as shown below.



In addition, there is also an unlock function which allows unlocking of a locked android device to allow interaction with the actual device. However, for now it will require root privilege in the android device first.

There are 3 main extraction functions available for the Android Forensic Tool currently, namely “extraction through backup”, “Normal Extraction method” and “Clone” the entire android device itself.

For extraction through the backup method, it involves using the default android device backup mechanism, this is the safest method to extract information from the device as close to no data are tempered in this way. This method also does not require root privilege for the android device. However, the data that are extracted through this method are limited as several databases are unable to be extracted due to the lack of privileges.

For the normal extraction method, a system memory dump of the device is first taken as every processes and statistic in the android device are recorded down in the system dump. Important databases such as contact databases are recorded and stored in the “databases” folder. Several databases are also analyse and sent to the centralised database for the splunk application to take in. However, this will require the Android Forensic Tool to exploit the device in order to obtain the root privilege.

These are the data that can be analyse from the databases collected:

* Device Information
* Application List
* Contact
  + Contact Details
  + Contact Email
  + Contact Number
* Call Logs
* Browser
  + Browser History
  + Browser Bookmark
  + Browser Search
* SMS
  + SMS History
* Skype
  + Skype Account
  + Skype Contact
  + Skype Conversation
* Accounts

For “Clone”, the entire android device is copied down into a directory for the analyst to view any files that are inside the android device.

Alvin

I had managed to be able to extract out many relevant information from an iOS device – iPad using the iTunes Backup. The extracted data will then be stored inside the sqlite3 centralised database, then the data will be sent to Splunk to start analysing the extracted data.

The following are a list of data that can be extracted out with the aid of iTunes Backup:

* Device Information
* Application List
* Contact
  + Contact Details
  + Contact Email
  + Contact Number
* iMessage
* Browser
  + Browser History
  + Browser Bookmark
  + Browser State
* SMS
  + SMS History
* Skype
  + Skype Account
  + Skype Contact
  + Skype Conversation

Apart from that, I had also done the exporting of the extracted data out from the sqlite3 database into another new .XLS file. This .XLS file can act as a backup or reference for the analyst to keep track and log down what data was being extracted from what device at what time.

Additionally, I also did the enhancement of my iOS Forensic Tool. I improved the iTunes backup input from the analyst from having to copy and paste the backup string path, into selecting a backup path from a folder directory, until a combination of both to allow the analyst to have more ways to input the iTunes backup path whichever suits the analyst. I also did a progress bar onto the system to allow the analyst to keep track on the progress and estimate when will the code finishes the progress.

Furthermore, I also had a better understanding after researching on the Smudge Attack of Touchscreen devices such as the iPad and Android devices.

Joshua

* Splunk 6.0 Django Web Framework
  + Created SplunkforMobileDeviceForensics app
  + Used Python, HTML5, CSS and javascript to build framework and dashboards.
  + Created a python script that Splunk will use to extract the latest events from the database. Events that have been extracted will be filtered out.
  + Completed the following dashboards
    - Overview
    - Device
      * Calls
      * Contacts
      * SMS
    - Applications
      * Application list
      * Skype
      * Browser Habits
    - Timeline

## 7.2 Evaluation and Future Enhancements

Caleb

The following are the future enhancement for Android Forensic Tool:

* Unlocking the device without root privilege
* Analyse more databases so that more data can be obtained from the android device
* Make the Android Forensic Tool more lightweight
* Ensure that the Android Forensic Tool is cross-platform
* Provide more guides to allow new users to understand the tool
* Allow the use of Android Forensic Tool for a wide range of Android devices and versions
* A GUI version of the Android Forensic Tool to enhance usability and functionality
* Minimizing the number of dependencies of the scripts

Alvin

The following are the future enhancement for iOS Forensic Tool:

* Able to perform extraction of any iOS versions (including iOS7)
* Able to connect to iTunes and backup automatically using python scripts
* Analyse more databases so that more data can be obtained from the iOS device
* Make the iOS Forensic Tool more lightweight
* Ensure that the iOS Forensic Tool is cross-platform
* Provide more guides to allow new users to understand the tool
* A GUI version of the iOS Forensic Tool to enhance usability and functionality
* Minimizing the number of dependencies of the scripts

Joshua

Future enhancements for the Splunk App largely depends on what new data can be extracted. Dashboards that correlate different source types can be considered. Emails would be the next logical step if we had more time, emails on mobile devices is the norm now, it contains a wealth of information on the target device.

The next step would be to ensure that the dashboards are as friendly and easy to understand as possible. This would go in line with our goal of anybody being able to take advantage of our application.

## 7.3 Lessons Learnt

Caleb

After developing Android Forensic Tool, I found out on how much data can be actually obtained from the device itself. The information in an android device tells so much on a user’s personality and interest. These information can also be used to prove a person’s crime or innocence.

I actually learned to use an application to wipe all data left behind by several applications, this is so that personal identifiable data is not stored in the device. This is to prevent unwanted users to obtain sensitive data from my android device.

In addition, it is advisable to constantly update the android device version whenever it is safe, disable “USB Debugging” and not to root the device itself. This will make it harder to extract data from the device itself. Constantly check for loopholes by using vulnerability scanning applications in the android device and patch them to reduce or prevent the chance of extracting data from the android device.

Throughout this period, I have also learn a new programming language which is python, and increase my understanding on Android further more. Last semester, I work on the Android Application side. However for this project, I learn more of the Android structure and also on the adb portion.

Alvin

From this project, I had gain a lot of knowledge and experiences in the forensic field of study. I understands that the project should be done by extracting the data without actually installing any application into it, as it will hence tamper the evidence itself, which violates our case study. I got to understand more into the Smudge Attack of the iPad and even other devices, such as Android. I also did research on many different ways to accomplish my project requirements

From the technical point of view, this project gave me the opportunity to learn a new programming language – python, which further broaden my technical knowledge and skills. I also have more understanding towards iOS devices and using iTunes backup to perform extraction of the data. I learnt on using sqlite3 as the centralized database for our project to connect between the Splunk and the data extracted out from the iOS device and the Android device respectively. I also learn on how to export the data out from the sqlite3 databases onto a new .XLS file. Furthermore, I’m able to create a program that is easy for the analyst to use.

From the social point of view, this project allows me and my teammates to work together as a team and gaining bond with each other. It also gives us the opportunity to learn new stuffs and strengthen skills from one another, exchanging of knowledge and learn together as a team.

Joshua

When we did the project planning, we decided that we were going to take an iterative and incremental approach to developing the project. We took lessons from the Scrum agile software development framework.

A key concept of Scrum is maximizing the team’s ability to deliver quickly and respond to changing requirements. I feel this applies to our project very well because mobile forensics without tampering the device is a new territory for us. We weren’t sure what we could or could not extract. When Caleb or Alvin would discover a new piece of source type to extract, we would add that to the requirements list, as such we had to rapidly adapt to the new changes. But even with a increasingly bigger requirements list each week, our end goal still remains unchanging.

My future projects would definitely use a lighter version of Scrum for managing the application development. It is a superior framework to the waterfall method and allows for more flexibility in responding to changing requirements.