Android App Forensic Evidence Database

DESIGN DOCUMENT

sdmay19-38
Clients: Neil Gong and Yong Guan
Advisers: Neil Gong and Yong Guan
Team Members/Roles:
Mitchell Kerr - Technical Lead
Connor Kocolowski - Report Manager
Emmett Kozlowski - Scribe
Matt Lawlor - Meeting Facilitator
Jacob Stair - Testing Lead

Team Website: http://sdmay19-38.sd.ece.iastate.edu

Revised: 10/12/2018 / Version 1.0.0

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

1 Introduction

1.1 Acknowledgement

Team 38's Client: NIST Center of Excellence in Forensic Sciences - CSAFE at Iowa State University

Team 38's Advisors: Profs. Yong Guan and Neil Gong

1.2 Problem and Project Statement

With technology becoming more and more integrated into the lives of everyone, digital forensics has begun to play a larger role in proving innocence or guilt of suspects. One piece of technology that is a staple of everyday life are cell-phones. Mobile app, which are located on the phones create records of all sorts of digital evidence such as GPS locations, activity timestamps, visited URLS, web history, social media contacts, etc., that is either saved on the device or on their own servers. Current digital forensic practices often involve manually combing through files, shared-preferences, and databases on the mobile device. This process is time consuming and error prone.

Our task is to create a real-world evidence database of over 7 million Android apps from roughly 40+ app stores that are globally used. We will create web crawlers to traverse the app stores collecting metadata and downloading the application. After collecting the application file, we will run it through the forensic analysis tools to collect where the application is storing the information that it gathers. Having this information stored in the database, we will then allow users to request the information about a specific application.

1.3 Operational Environment

Our project exists entirely as a software tool. As such, no specific physical conditions will need to be considered for the end product. However, our project will include a web interface to allow for use by forensic investigators. In order to achieve this, it is a requirement for the end product to work on multiple web browsers across several operating systems.

1.4 Intended Users and Uses

The primary intended end user for our database will be digital forensic investigators. There are several different scenarios in which our database will be useful to a digital investigator. One such situation might be in utilizing the location data from an

application, associated with the timestamps, to prove that a client was not in a certain place at a certain time. Investigators and their teams will be able to go to our web interface and search for all the applications which are present on the mobile device. Our software will tell them which applications contain the desired metadata.

Another possible user of our database will be academic researchers who study mobile applications. As this database will be the largest collection of Android APK files to date, researchers studying anything related to mobile applications and their contents may find use in the ability to search through all the available applications across all the third party app stores.

1.5 Assumptions and Limitations

Assumptions

- A long term hosting solution is found
- BeautifulSoup Python Library is not deprecated for future updates to the database

Limitations

- Server may not be able to handle a massive amount of simultaneous requests to the database or file system
- Project only focuses on Android mobile devices
- Crawlers may have not collected app store related information necessary for some investigations

1.6 Expected End Product and Deliverables

By April 28th, we are expecting to deliver a APK crawler that will crawl through 40+ app stores and collect app metadata and their APK files. Additionally, we will have a database that will store all the metadata from these apps and a evaluation based off the evidential data from the apps.

The APK crawler is designed to crawl through over 40+ Android app stores to collect metadata about these apps and download their APK files. It will be activated periodically to collect any new apps that are uploaded and to collect data on apps that have been updated. When finished crawling, it will upload the findings to our database to be yes processed at a later time.

As explained above, the database will hold onto the apps metadata and the file path leading to the APK files in our filesystem. After being stored, we will run the APK files through our clients forensics application to determine where these apps store their data on a user's device along with the type of data. This information will also be stored on this database.

Lastly, we will be making a report for our clients based off the results of our database and APK crawler. It will most likely consist of how well the crawler collects metadata and how efficient it is.

It will include possible improvements to be made when we hand the project off. The report will also contain information on our database architecture and its ability to scale and hold our information.

2 Specifications and Analysis

2.1 Proposed Design

For our design we decided to make each of our web crawlers flask apps, use MongoDB as our database and a filesystem for storing the APK files. Each crawler will be a separate service, allowing us to modify the crawlers individually as needed. This approach was also used so that we can add more crawlers for each app store by adding more instances of the flask app to our solution. We decided to use MongoDB as we wanted a database that could scale to meet our needs. MongoDB can scale horizontally which allows us to grow fast as we collect more and more data. We decided to go with a filesystem to store the actual APK files as they will not be accessed very often and there will potentially be large files that would be difficult to store in a DB.

Functional Requirements:

- The web crawler must download and store the APK file
- The web crawler must collect and store all the metadata
- Working web crawlers for all 40+ app stores
- All metadata must accessible from an API
- The APK files must be analyzed using the forensic tool

Non-Functional Requirements:

- The system must be able to scale to handle storing 100TB of data
- The web crawlers must be able to scan an app store in 1 week

2.2 Design Analysis

So far most of the team has been focused on researching and implementing the web crawlers for each of the app stores. We are have made significant progress on 3 app stores and are finalizing them while confirming that we are collecting all of the correct data. We are then going to focus on collecting the application data by having the web crawlers start saving information to the database. We will also start on creating our API for others to interact with the system.

3 Testing and Implementation

3.1 Interface Specifications

We are currently not using any interface to test our project. We have been manually testing our code these past weeks. We are planning on meeting up to discuss about an interface to use to test.

3.2 Hardware and software

Currently, our project has been entirely software focused. It is possible that down the line we could be using Android devices to test results gathered from the forensics tool provided to us by our clients. We would go through the phone's directories and verify that they contain the correct data listed from our forensics tool.

The software we are using is written in Python. We have been testing our software manually for the past few weeks. We check what our crawlers are printing out and make sure that they are collecting the correct info.

The usefulness of the phone will allow us to test our client's forensic program, but will give us the option to use a real device instead of a simulator. For some, the actual device is easier to use over a simulator. Python is useful to us as it allows us to multithread and has a large community for building web applications like web crawlers.

3.3 Functional Testing

Unit Testing:

Web Crawlers:

Each web crawler will have an automated unit test run against it that uses a local simplified version of its app store's site. The test site will be crawled through and the results will be compared to the correct data set.

Integration Testing:

Web Crawlers → Database:

The web crawlers must all be tested to ensure that they can communicate with the database correctly. All metadata must be written to the database and each app from each app store should have its own entry.

User API → Database:

The user API that forensic analysts will use to get information about specific apps will be tested to ensure that it communicates with the database correctly. The API must be able to query metadata and APK files from the database.

Database → File System:

The database will be tested to ensure that all file paths are unique and point to the correct files in the file system.

System Testing:

Once our entire system is established and communicating, we will run tests to make sure that apps and metadata follow the data path and are written into the database and stored in the file system correctly.

3.4 Non-Functional Testing

Performance Testing:

We need to make sure that our web crawlers are fast enough to collect all the data on the stores. We will test this by recording how long it takes to collect a certain number of apps. After we get this data we can see if the crawler is fast enough or needs to be enhanced to ensure proper performance.

Scalability Testing:

In order to test our system can scale to support the amount of data we need collect we will start with one crawler and slowly add more to the system and observe that there is no drop in performance as more load is put on the system. In addition, we also start with small amounts of requests to the API and scale that up to ensure we can support a large amount of users.

Usability Testing:

We will test our forensic analyst's application functionality with a sample set of users that have not been involved in the development process. This reveals how confusing the application is to a user that is not familiar with the system.

Compatibility Testing:

Each web crawler must work with its respective app store. However, all of these crawlers must work with the database, even with different sets of metadata. We will need to test that all variations of metadata and file types be accepted and written correctly into the database.

3.5 Process

- Explain how each method indicated in Section 2 was tested
- Flow diagram of the process if applicable (should be for most projects)

To be filled out later

3.6 Results

Data Storage:

The amount of data that needs to be stored for this project is tremendous. We have tried to use a VM provided by the school, but we ran out of storage space almost instantly. The culprits are the APK files. These range from a few megabytes to a few gigabytes. There are

millions of Android applications across the various app stores. This means that we must find a storage solution that can manage around an estimated 10-100 terabytes of data.

4 Closing Material

4.1 Conclusion

Our team plans to accomplish our goals, which consist of:

- Crawling over 40+ web-versions of android app stores
- Collecting all the data pertaining to each app (including the app itself)
- Storing all the information in a database
- Designing a way for querying the database for apps that log specified data

In order to achieve the goals we have set forth, we will delegate to each member a set of stores to implement a crawler for. We will also secure a sizable storage space for the crawler to write to, with a correlating database. Once we have both a database and a few crawlers set up, we will begin the collection phase of the project, where the crawlers run until they finish. After the data has been collected, we will create a user-friendly way to search for specific data and the apps that log it.

Our solution will be able to reduce the time taken for digital forensics. It is able to do this by utilizing the database we created. Forensic analyzers will be able to query our system by entering the name of applications of the digital device and our system will return the type and location of the information the app catalogs. This transaction of information will be much faster than combing through every file on the device.

4.2 References

https://www.python.org/

https://www.mongodb.com/

4.3 Appendices