# Android App Forensic Evidence Database

DESIGN DOCUMENT

sdmay19-38
Clients: NIST Center of Excellence in Forensic Sciences - CSAFE at Iowa State University
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: 12/02/2018 / Version 2.0.0

# Table of Contents

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



### List of Definitions

CSAFE - Center for Statistics and Applications in Forensic Evidence

REST API - Representational State Transfer Application Programming Interface

ETG - Electronics and Technology Group

IEEE - Institute of Electrical and Electronics Engineers

API - Application Programming Interface

ANSI - American National Standard Institute

SQL - Technology used for database management with fixed structures

NOSQL - Technology used for database management with dynamic structures

JSON - File format with key-value paired data

# 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

We will need a physical server that can support the amount of memory space we will accumulate. The various web crawlers will be running simultaneously in containers on our server. Our project will also include a web interface for investigators to use. This must be compatible with several operating systems and multiple web browsers.

## 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, and a database that will store all the metadata from these apps and a evaluation based off the evidential data from the apps. We will also have a user interface that will allow users to query the database and return information about a desired app.

We will also  be making a report for our clients based off the results of our database and APK crawler. It will  consist of the efficiency of the crawler and the structure of the database. It will also include possible improvements to be made when we hand the project off along with extensive documentation and testing.

# 2 Specifications and Analysis

## 2.1 Proposed Design

In regards to our crawler design, we decided to design each crawler with a unique skeleton for visiting each application on a given store. Although for collecting the metadata belonging to each app, it was decided to create an abstract structure. This would allow for each crawler to only implement the metadata retrieval for information on the site and still have a common method call.

For our design we decided to make each of our web crawlers flask apps, use MongoDB as our database and use CyBox as our filesystem for storing the APK files. Each crawler will be a separate service, allowing us to modify and scale the crawlers individually as needed. We decided to use MongoDB as our database because 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. Figure 1 below shows what our designed system will look like.

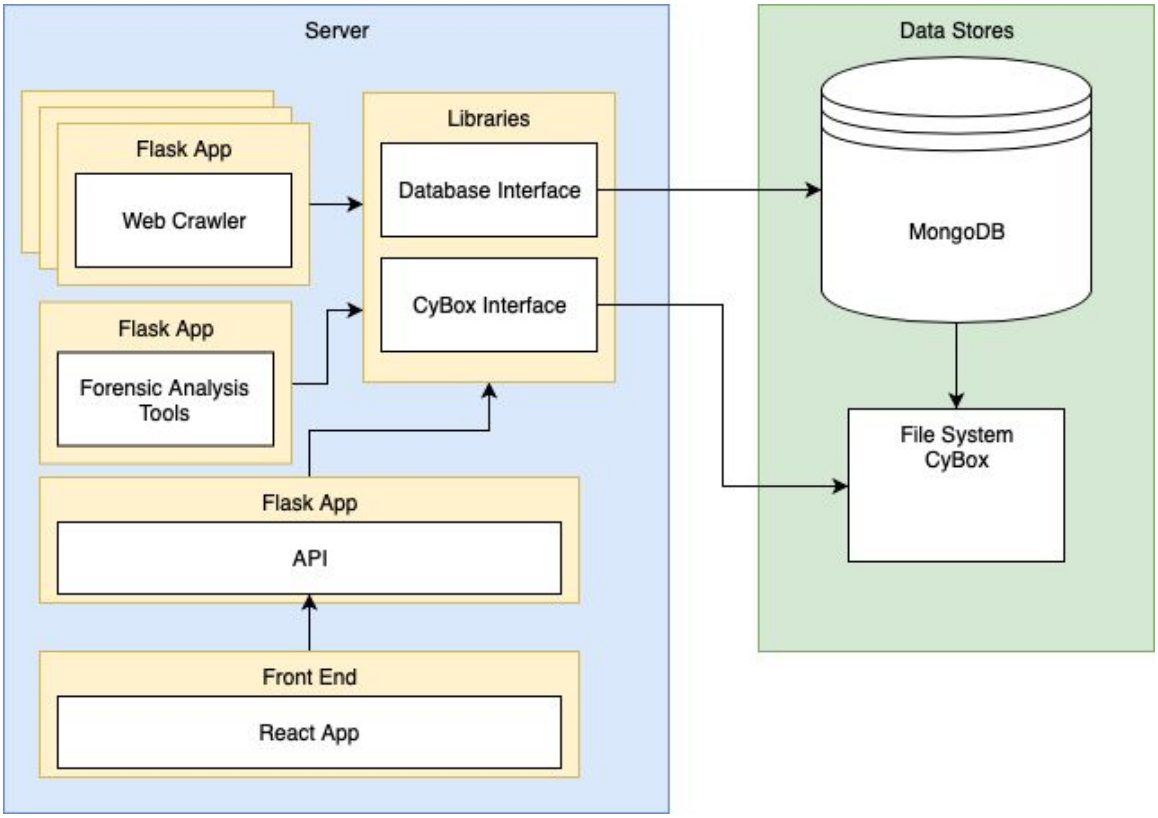


Figure 1. System Design

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  collect data on apps that have been updated. While crawling, it will upload the findings to our database and will be processed by the forensic analysis tools.

As explained above, the database will hold onto the apps metadata and the file id leading to the APK files in CyBox. 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. We are going to use the following database model to store all the data we collect into three collections: Application Store, Version, and Forensic Reports.

Database Model

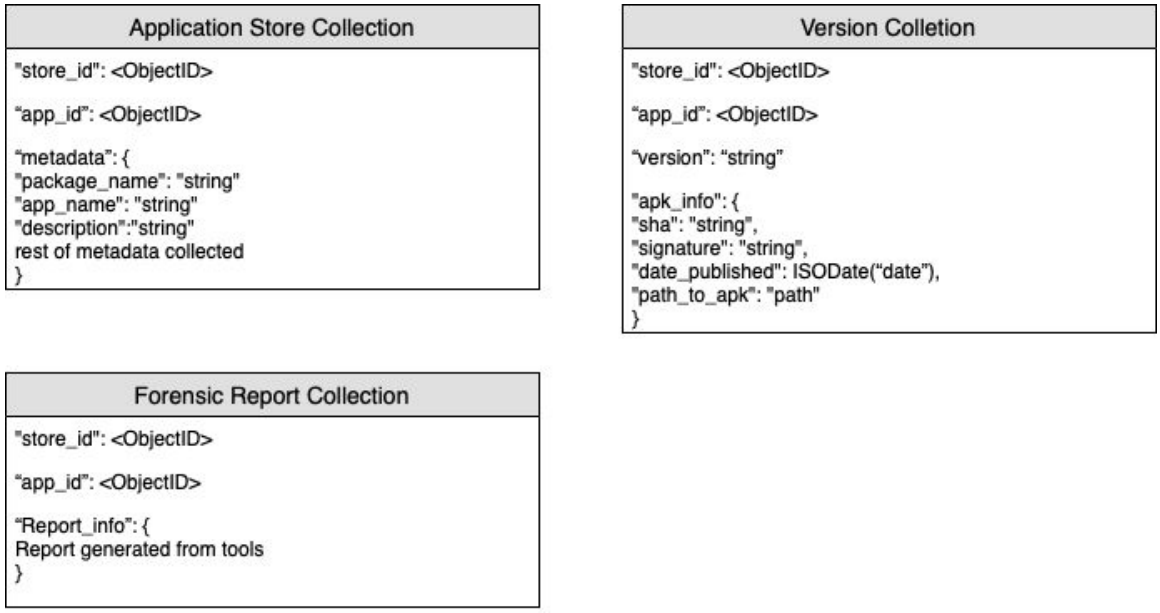


Figure 2

For building and deploying this system we plan on using docker containers for each of our services. Each service will be its own container in order to isolate each service from one another and make them more secure. Additionally, by making them all containers we can deploy each of our services faster and update them in an automated fashion. In order to ensure that all of the containers are running smoothly we need to have a container orchestrator. We have chosen Kubernetes to be our orchestrator as it allows us to scale our services easily and have health monitoring for our services.

Functional Requirements:

- For all apps on an app store.
    - APK crawler must collect APK files. In addition to collecting the metadata, the crawler will also download their current and past APK files.
    - APK crawler must store data in a database. This is self explanatory, but once finished with collecting app metadata and its APK, it will store the metadata and the location of the APK file in our file system into the database.
    - App data must be passed into forensic program. Once collected, the APK will be passed through our client's forensic program that will output additional information about where the app stores data on a user's device.
- Database must store results from forensic program. Again, self explanatory, but once data has been passed to the forensic program it must store the results outputted.

Non-Functional Requirements:

- Scalability - The system must be scalable to support all the vast amount of applications we need to download and analyze. This will be done by designing our system using microservices and using a NoSQL database.
- Availability - The system must be operating 24/7 to ensure that all versions are collected when they are updated on an app store. In addition, the service needs to be available 24/7 to support 3rd party requests at any time.
- Reliability - The system must be able to recover from an event such as a power failure. The system will be designed with fail safes and recovery functionality to ensure that the system can rebound from a negative event.
- Maintainability - The system needs to be maintainable past the day that we deliver the product. Since webpages are constantly updated and changed the crawlers also need to be updated to support those changes. This will be done by creating detailed documentation on the product and designs of how each component should behave.
- Security - The system must be secure and only allow authorized users to interact with the system. We will implement this by carefully designing our system to ensure only authorized users can access the data.
- Data Integrity - The data that we collect from the websites and the forensic tool should not be modifiable by any individual after collection. We will ensure that the data remains genuine by restricting access to the database and filesystem. We will also replicate our database to ensure that if there is a database failure we will not lose any data.

## 2.2 Design Analysis

In terms of the system design for the various web crawlers we chose to abstract the design so that when implementing more crawlers, we would have a bank of possible methods to add when applicable. This also allowed us to simplify the base of the crawler so that its

main functionality is to visit every application and the data collection would be handled elsewhere. However, from abstracting the crawlers, we couldn't design them with a structure conforming to the corresponding store, and instead had to manipulate it into a common form.

Our design for this system is focused on being able to scale and operate in the future with the amount of data that we will be collecting. We achieved this by separating the app store crawlers into separate services that we can scale individually based on the store if there have been a lot of updates on that store that week.

We chose MongoDB because not only will it suit us now but in the future as the amount of data we will need to store will only grow exponentially. Since MongoDB supports sharding we can scale very fast and efficiently to meet the demand. With different stores offering different amounts of information we need to have a flexible way of storing information. MongoDB allows us to store our data without defining all of the key value pairs in advance. In addition, the forensic analysis tools output their results in JSON format allowing for easy insertion into the database.

We chose CyBox as our storage system. CyBox gives us data replication and "infinite" storage for free as it does not cost anything to use. Having elastic storage is important to us as we will have a large amount of data to start with and will continue to grow as time goes on. Some tradeoffs that we made by going with this design is since we are using a 3rd party for data storage we suffer on speed since we need to upload them to the cloud. CyBox also limits how many files you can simultaneously upload along with having a maximum file size. However we will never go above the size limit and the amount of files we can upload at a time will only be a factor in the beginning when we need to backfill all of the information from the application stores.

By choosing to use containers for our services it allows us to scale easier, have health monitoring, and additional security layers in our system. However by containerizing all of our services we need to have a way to monitor and keep track of them. Having to deal with setting up a orchestrator will take time and resources away from the project and developing all of the crawlers.

# 3 Testing and Implementation

## 3.1 Interface Specifications

The web crawlers collect data from the app stores and then add the data to our database. They also upload APK files to our server as they will not be stored into the database.

A user will be able to access our system by using a web site. It will be built using React, and will make an HTTP request to our server, in order to query information about the desired application from our database. The server will then relay a response back to the website.

## 3.2 Hardware and software

The program we are developing is being written in Python. Python is useful to us as it allows us to multithread and has a large community for building web applications like web

crawlers.The UnitTest python library will be be used for handling the web crawlers' unit testing. Each web crawler will have sample html pages and sample data from those pages. The crawler will run on the samples and validate that the data it collects matches the sample data. This ensures that our code correctly handles pages and outputs data in the desired format.

We will not be using any hardware in our testing since our entire project is based in software.

### 3.3 Functional Testing

Unit Testing:

Web Crawlers:

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

Success condition: 100% data match

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.

Success condition: 100% of data is written correctly

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.

Success condition: 100% of data is correctly obtained within 5 seconds

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.

Success condition: 100% of files are in the expected directories

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.

Success condition: Average metadata collection takes 10 seconds, average APK file download and upload to file system takes 30 seconds

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.

Success condition: Performance from small to large scale drops by less than 15%

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.

Success condition: 95% of test users understand and can navigate the application, 90% of test users can perform desired actions correctly

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.

Success condition: 100% of data variations are written correctly into the database

### 3.5 Process

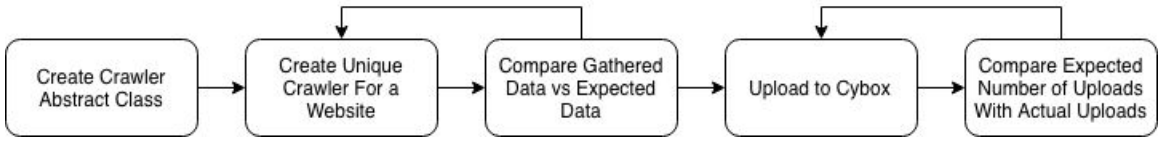


Figure 3. Process Diagram

### 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. Fortunately, the sponsors of our project, CSafe, have granted our team 30 terabytes of server space. The main reason our project requires such substantial storage are the APK files. The size of these files can range from a few megabytes to a several gigabytes. There are millions of Android applications across the various app stores.

Abstraction:

> Considering the large number of web crawlers that we will be building, it was decided early on that we would work to abstract our code as much as possible. This is done by creating utility classes which hold generic function for common web crawling issues, and interfaces to ensure all the crawlers are getting some base level of metadata. However, as we have continued to build our crawlers, we have found that abstracting the process can be very difficult given how different the various Android store's websites are set up. While abstraction can be arduous early on, implementing more crawlers will make abstraction easier as time goes on, as more cases will  be accounted for.

# 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

Python. www.python.org/. Accessed 2 Dec. 2018.

Mongo. www.mongodb.com/. Accessed 2 Dec. 2018.