Android App Forensic Evidence Database

PROJECT PLAN

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: 10/26/2018 / Version 2.0.0

Table of Contents

List of Figures

List of Tables

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

1 Introductory Material

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 Statement (2 paragraphs +)

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 apps, which are located on 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. We will then design a user interface that forensic analysts can use to query mobile apps from our database and find where the information they need is.

1.3 Operating 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 for use by investigators. In order to achieve this, it will be a requirement for the end product to work on multiple web browsers across several operating systems.

1.4 Intended Users and Intended 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 will 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 Other 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 Proposed Approach and Statement of Work

2.1 Objective of the Task

Our project aims to develop a complete database consisting of information taken from every android application from 40+ different app stores. The desired outcome from this is to allow for criminal justice investigations to reach a verdict of guilt in a time-efficient manner. This will be possible because they, will be able to query our database and get a resulting list of applications that log the type of data they are searching for. In order to make the process more efficient, we will also gather the number of downloads, ratings, and other information regarding each app. This speeds up the process because it will give an idea of apps that are likely to be a given phone.

2.2 Functional Requirements

In order to meet our goals the:

- APK crawler must collect app metadata. The crawler must collect the metadata 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 for forensic program. Again, self explanatory, but once data has been passed to the forensic program it must store the results outputted.

2.3 Constraints Considerations

For this project, constraints we have set forth consist of:

- 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 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 to have fail safes and recovery functions 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 modified by any individual after collection. We will ensure that the data will remain 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.

The backend services for our system will have API endpoints to allow for the exchange of information between each microservice. The endpoints will be designed and documented following the OpenAPI Specification. This specification is the standard for developing REST API's. For ethical issues the team will use the IEEE ethic code to determine what the best course of action is.

2.4 Previous Work And Literature

All members of our team have had plenty of experience writing software. Specifically, every member has designed and written a web crawler in Com S 311, an algorithms class. One of the projects in that class involved creating a fast and efficient Wikipedia crawler in Java. While we won't be working with Java, the knowledge still translates over well. This experience should come in handy when designing the web crawlers for the Android app stores.

Multiple team members have also had experience designing and implementing a microservice back-end system. This was done in Com S 309 and through an internship. Com S 309 entailed creating a group project from start to finish, which required some kind of back-end system. One of our team members worked with back-end systems during a summer internship, so they are leading the back-end design.

When conducting research for the project, we found an existing product that crawled through a web-version of several app stores. However, many of these existing crawlers were out of date and did not work. These crawlers were found on github at https://github.com/opengapps/apkcrawler. We decided to build our web crawlers from the ground up instead of working off of this repository, since updating and fixing the web crawlers there would not save much time in the long run and might lead to a lesser understanding of the project.

2.5 Proposed Design

As our application will need to handle upwards of hundreds of terabytes of data, careful consideration was taken as to the overall project design. In the end, we decided upon a combination of Python Flask apps, a MongoDB instance, and a filesystem. Figure 1 gives a high level view of our system design. Each web crawler will be its own flask app and will be responsible for gathering all the APK files and metadata for its respective application store. This will be done by implementing the Python library, Beautiful Soup. Beautiful Soup will allow us to target a web page's html and collect our metadata. The MongoDB will be responsible for storing all metadata about an app, the reports generated from the forensic analysis tools, and the file path to the APK files. The APK files will be stored in CyBox as we will not need to access the APK files frequently. Using the Box SDK, each crawler will upload a APK file on seperate threads to keep the program running as fast as possible. Our server will be running a program that is in charge of creating and running crawlers. Each crawler is responsible for collecting data from its store and uploading it to our database. From there, the forensics software are clients will provide us with, will pull the APK files from the database and process them to retrieve the location of the stored information. A user can access these findings through a website. They will enter in query values to find the data they need from the database. We decided on a single program to run our crawlers multiple times. We went this route as our clients stated that they would like these crawlers to be running periodically.

2.6 Technology Considerations

Before we started, we debated about which coding language would work best for our project. We talked about using Java, Python, or Javascript. We started with these three as they are all known for building web crawlers. As we discussed how we would like these web crawlers to work, we realized that we would like to implement multithreading. We then ruled Javascript out as it does not support multithreading. Java and Python both had libraries to help us with our goal. One big issue we faced was that we were not familiar with Python. However, the repository our clients should us was written in Python and could be used as a reference. Online, it was recommended that you used Python for web crawlers because it is a scripting language. We ended up deciding on Python as the referenced repository was written in Python and it seemed that there was a lot of support online using Python as a web crawler.

Another crossroad that we faced was which type of database should we use. The big question was should we go SQL or NoSQL. One of the big advantages of NoSQL databases is its ability to scale. Because we are downloading so many apps and metadata, it made sense to us that we would want a database to be able to handle this influx of data. NoSQL databases also allows us to have a unstructured schemes. This allows us to have documents and data with a variety of variables. This could be useful when dealing with 40+ app stores as they all have various amounts of metadata for their apps. SQL databases requires queries to be predetermined. This feature allows large amounts of data to be retrieved quickly and efficiently. SQL databases use long-established standard, which is being adopted by ANSI. NoSQL databases do not adhere to any clear standard. It is easier to manage SQL database systems without having to write substantial amount of code. In the end we favored the scalability of NoSQL with the unstructured schemas. We felt that this these features would best suit us.

2.7 Safety Considerations

Since our project does not actually involve any hardware development, there is little to no physical danger that we must concern ourselves, our clients, or our end users with.

The Android app data and files must be secure. One of the reasons we chose MongoDB is because it is a secure database, so we should not need to be concerned about our data getting in the wrong hands.

2.8 Task Approach

Our client has specified that we need to use web crawlers to download and store Android apps and metadata from various app stores. We have decided to use Python as the language that our crawlers will be implemented with. This is due to Python having existing web crawling libraries that make creating web crawlers much easier than with other programming languages. After deciding on Python and the BeautifulSoup library, we designed a microservice architecture that best fits the project specifications.

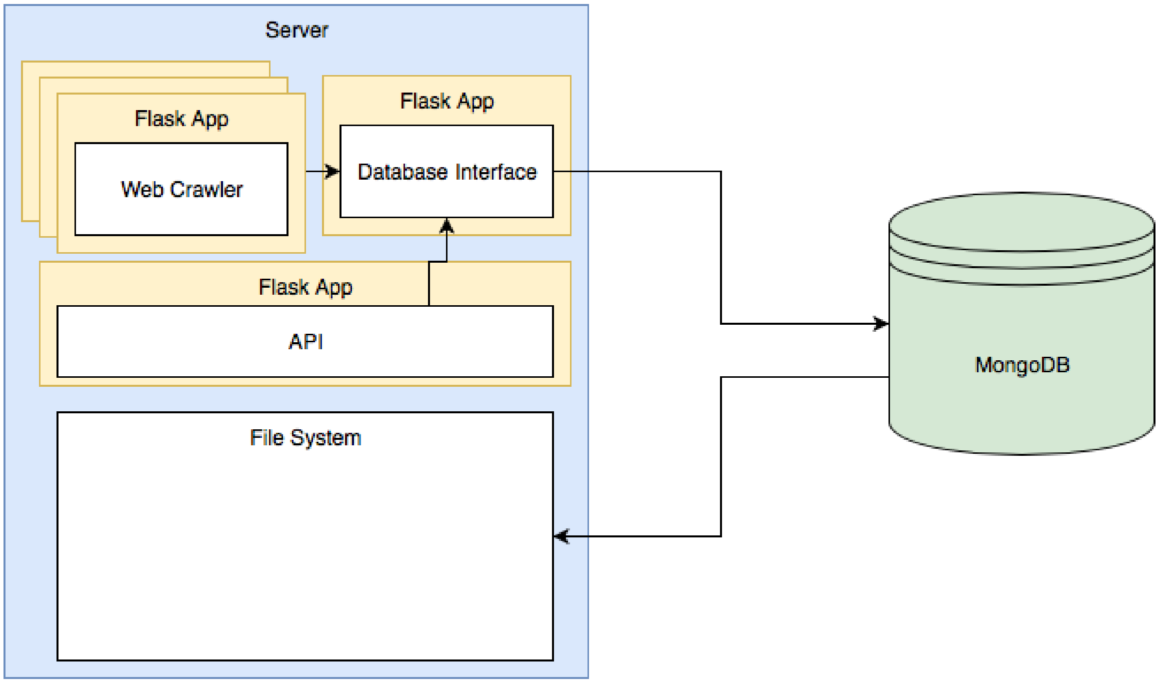


Figure 1: Proposed Architecture Diagram

We are using Flask to set up Python applications that perform the actions in the system. The web crawlers access the app stores and obtain the APK files and metadata. This is sent through the database interface application to a MongoDB database. The metadata and APK file paths are saved into a table while the APK files are saved into a file system on the server. Finally, an API is used to query the database. This would be used with a GUI that forensic analysts can use to get Android app information. Another discussed solution was to save everything within the same database, without using the filesystem. While this architecture would have made the system more encapsulating over all, storing the APK files in a filesystem will solve many of the complications of designing a schema to store APK files of varying sizes.

2.9 Possible Risks And Risk Management

Our project requires that we obtain a large amount of storage space to store all the applications that we download. We might not be able to obtain all the space needed to download all the APK files. We are looking into possible solutions that we can utilize at the scale we need.

Another resource that might be difficult in obtaining at the scale we require is computing power. Our solution will be crawling multiple app stores across millions of pages requiring a significant amount of computing power be dedicated in parsing all of the web pages.

Currently we are talking to CSAFE to provide us with the compute power that we need. In the meantime we can use a VM from ETG to test our service on a smaller scale.

We are implementing the solution in python and using MongoDB for our database. We as a team do not extensive knowledge or have worked on a large project in python. This will require us to ramp up our understanding of python. In addition, none of us have worked extensively with MongoDB before which is another piece of technology that we will have to learn.

2.10 Project Proposed Milestones and Evaluation Criteria

Some milestones we have established are:

- To have the crawler tools completed by the end of October. This will be tested by checking the data sent back from the crawler is accurate to what data we are looking to acquire.
- To have the database architecture complete by the end of September. This will be tested by the approval of our clients.
- Collect a majority of app data by December 20th. We will test this by making sure the app crawlers for each store are collecting the appropriate data and not throwing errors.
- Perform app analysis on the apps collected by April 15th. At this point we will have the app data we desire. The output of this data relies on our client's forensic app.

2.11 Project Tracking Procedures

We are using a Trello board to track what tasks have been completed and which tasks are assigned to different team members. Our team also writes weekly reports that detail all individual work that has been done that week, as well as the state of the project.

We meet with our client once a week to update them on our progress, ask questions to clarify project specifics, and discuss challenges that we are facing. This is a great way to keep communication between us consistent and is a good way to demo the project.

We also hold team meetings at least once a week to discuss deadlines, assign tasks, and make project decisions. This is also a good opportunity for the team to prepare any demos or presentations for our weekly client meetings.

2.12 Expected Results and Validation

The desired outcome is to create a APK crawler that will crawl 40+ app stores and collect their metadata and APK files. The metadata and the file paths pertaining to the location of the APK files in our filesystem will be stored in our database along with information about the APK after being run through a forensics program developed by our clients. This will allow a user to enter in a query in a website and access the information for investigation purposes.

At a high level, we will ensure that a single crawler can pull data correctly from one website. We will then make changes to suit each app store as they all store data differently. This satisfies our requirement to collect all the metadata from a single website. Our database will be designed to look for the same information as most of the metadata is common amongst the stores.

2.13 Test Plan

Web Crawlers:

All of the web crawlers must correctly obtain the APK files and metadata from their respective Android app stores.

Test: Each web crawler will be run on *n* number of apps. Each APK file and its metadata obtained must be checked against the versions on the web page for any inconsistencies.

Result: All APK files and metadata are correctly obtained from the web crawler's Android app store.

Database:

The database must store all versions of apps, as well as different metadata from different Android app stores. The APK file paths must point to valid APK files.

Test: Write dummy entries into the database with the same app names but different versions and metadata. All of the entries should be present, with no overwrites that have different metadata.

Result: The database is populated while creating multiple versions of the same app.

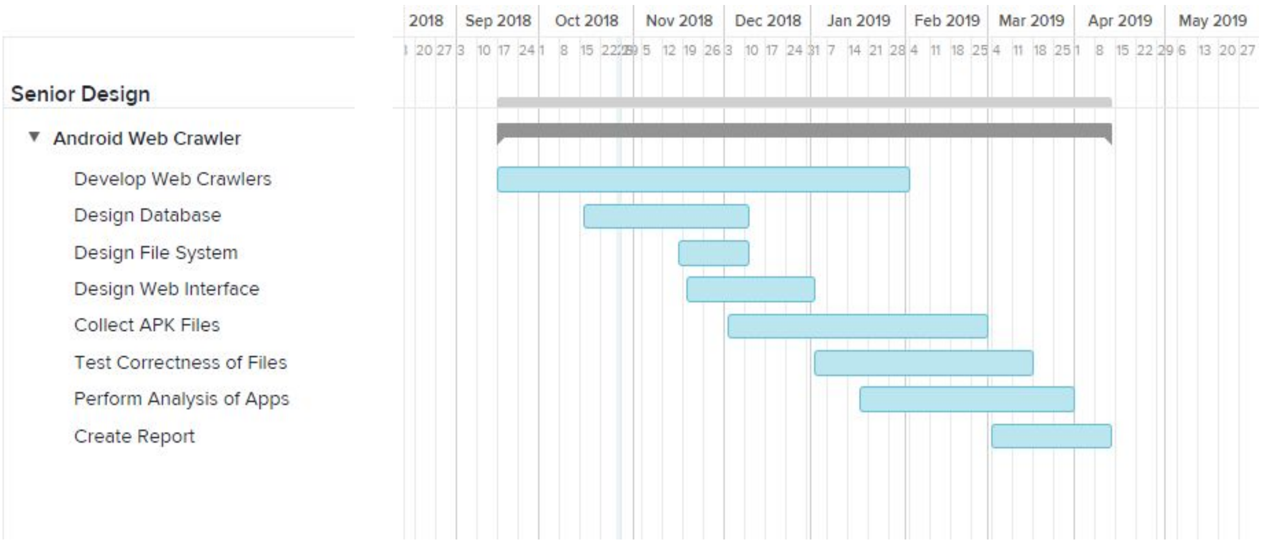
Test: Write dummy entries with APK file paths into the database and add APK files to the file system. Check that all of the file paths in the database lead to the correct APK files.

Result: All of the file paths in the database point to the correct files.

3 Project Timeline, Estimated Resources, and Challenges

3.1 Project Timeline

Our project timeline is laid out in several phases. In the first phase we researched possible technologies to use in our solution. In particular we will research how we were going to crawl the application stores, what type of database to use, and how to store all the APK files. After we decide on the technologies we were going to use we will move on to our next stage and begin implementing web crawlers for multiple stores. In addition, we will also design our database schema and interactions with CyBox. We then will begin collecting all of the information. This will be started by the end of the first semester and completed in the second semester as it will take a large amount of time to process all of this data. In the second semester we will focus on performing analysis on the APK files and



3.2 Feasibility Assessment

When looking at the desired outcomes and requirements for this project, it seems feasible. This is because we have a detailed plan as to how we are going to proceed, and when looking at the projective timeline, we have allocated enough time to maintain a buffer for any unforeseen setbacks. Additionally, we do not need to worry about cost because this will be a purely software project, and the database hosting will be on Cybox. Cybox allows free, unlimited storage to students, which will eliminate hosting costs. Some of the challenges that are likely to appear in the project are that when creating personalized crawlers for each app store, there will be html formats that are not uniform across the different stores and sometimes within the same store. Along with that, the stores do not all conform to a standard language. Another challenge will be in storing all the information, as there are 40+ app stores and google play alone has 3.8 million apps. With a sample space this large it will take up a large amount of space, which will be a challenge in acquiring somewhere to store all the data.

3.3 Personnel Effort Requirements

As seen below, Table 1 is a table of the major tasks that we need to accomplish in order to complete our project. Implementation and documentation will take up the bulk of the time, with implementation taking 730 hours alone. The total estimation for completing our project is 825 hours. We have completed most of our research goals and have begun implementing a starting crawler. With that baseline set, our work will be mimicking behavior over the multiple app stores.

Major Tasks:

Task	Description	Estimated Time
Setup VM	Acquire a VM from the ece department that we can deploy our software onto	5 Hours
Research previous app store crawlers	We will research previous versions of app store crawlers to get an idea of how to construct our implementations	20 Hours
Research app store html layout	Each app store has at least one unique layout for how they display application information	5 Hours
Design database architecture	We will need to create a database that can manage the different types of information along with dealing with the large amount of data	10 Hours
Implement database	We need to have a stable database to be able to send data to our database for storage	10 Hours
Implement backend system	We need a way to communicate with our database for pulling and pushing information	30 Hours
Implement app store crawlers	We need to develop a unique solution for each android app store in order to process every application on each store.	16 Hours per store * 45 = 720 Hours
Test Crawlers	We need to make sure each crawler works	25 Hours

	correctly	

Table 1: Major Tasks

3.4 Other Resource Requirements

The project will require external resources to maintain the team's documentation and Git instance. In addition, the team will also need a virtual machine to test the system. All these resources will be provided by the Electrical and Computer Engineering Department's Electronics and Technology group. We also will need a final spot to deploy our solution to and the digital forensic tools. These will be provided to use by CSAFE. The project also requires a large amount of storage to support downloading such a large number of applications. We will obtain the storage space to by utilizing Cybox as it allows unlimited storage.

3.5 Financial Requirements

We currently have no financial costs. The equipment required to run our system will be provided to us for free from CSAFE, the Electronics and Technology group and Cybox. In addition, all of the software that we will be using to implement our solution is free for us to use.

4 Closure Materials

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 increase the speed of forensic investigation, because our tool will provide an easy to use interface, with quick response times from our database. This will give forensic investigators to get the information they need in a time efficient manner.

4.2 References

Buse, Nicholas. “Opengapps/Apkcrawler.” *GitHub*, github.com/opengapps/apkcrawler.

4.3 Appendices

Parts of the Plan we didn't get full points on:

Format Issues	- Everyone
~~Problem Statement~~	~~- Jacob~~
~~Previous Work/Literature~~	~~- Jacob~~
~~Proposed Design/System/Solutions~~	~~- Emmett~~
~~Assessment of Proposed Solution~~	~~- Mitch~~
Validation and Acceptance Test	- Matt
~~Project Timeline~~	~~- Mitch~~
~~Challenges: Risks/Feasibility Assessment~~	~~- Connor~~
Nature of Content	- Everyone
Technical Approach	- Matt
~~Process Details~~	~~- Emmett~~
Standards	- Everyone
Conclusions	- Connor