# Forensic Analysis of Cloud Artefacts on the Google Suite

Brandon Shavers[1], Christopher Dube[2], Kanwalinderjit Gagneja[3]
[1,2,3]Florida Polytechnic University, Lakeland, FL, USA
[1]bshavers2525@floridapoly.edu, [2]cdube2269@floridapoly.edu, [3]kgagneja@floridapoly.edu

**Abstract:**
This paper presents forensic analysis on cloud artefacts by determining how and where information is stored regarding cloud-based software. This paper specifically research and work with the Google Suite collection of software, using both files kept on the system via backups, information stored in the history, and other log files stored by the browser. In this research we have also designed and developed a unique program that reads through log files for specific metadata keywords related to the Google Suite collection. The main focus of this research is to look at modern digital forensics techniques and determine how they can apply to this new medium.

**Keywords:** Cloud Artefacts, Google Drive, Google Suite, Google Docs, Digital Forensics, Cyber trails, Circumstantial Evidence, eDiscovery, Exchange Principle, Metadata.

## 1. Introduction

As technology has advanced and become prolific throughout both the residential and commercial marketplaces, a shift has occurred in the way that technology has been marketed. While software and hardware use to exist in a perpetual loop of purchase, use, and replacement as the new year's model came out, we no longer see this trend. Businesses have instead moved to supporting services in the form of infrastructure, platforms, and software as continuous, subscription-based services.

This allows a newfound freedom to both consumers and industry as they can more easily access their products and services from multiple places and even different operating systems with little to no backlash to them. This however does change the dynamic in which data and files are linked to a computer and user. As such, previously known and developed digital forensic software and analytical techniques must be adapted and changed in order to properly work in this new medium.

### A. Cloud Based Computing

The key behind the creation of these new services comes in the form of cloud computing. The idea behind cloud computing is that a user's input or data is being sent to a server hosted and run by a company for a particular purpose. For instance, a popular form is cloud storage, in which digital space is purchased to hold photos, documents or other data. The idea behind cloud-based computing is that the processor and digital environment is used instead of just the storage aspects.

### B. Software as a Service

The main topic of our research deals specifically with software being sold and used as a service. To begin these are products like Office 365 and the Google Suite that are a collection of pieces of software that are packaged and sold for use by a consumer. In the case of Office 365 it is an annual subscription that allows a user to access a myriad of their products such as Word, PowerPoint, and Excel etc. This is very similar to the Google Suite collection that contains Google Docs, Google Slides, and Google Sheets. However, Google Suites is limited in functionality and is available for anyone to use for free.

While there are multiple types of software that fall into this category; all software, platforms, and infrastructure sold as a service follow a certain criterion. These are web based, sometimes browser based, but nevertheless they always require an Internet connection. This is due to the previously mentioned core being made from Cloud Computing. Since the processors and hardware are contained on a distant server a constant connection is needed to interact with the environment.

### C. Infrastructure and Platform as a Service

While our main focus for this paper is Software as a Service, we would be remiss if we neglected to mention that both infrastructure and platforms are sold and used in similar ways to their software counterparts. Both of these types of services are more used in industry situations as the platform provides a usable OS as a service and infrastructure provides the backbones of computer architecture for use. For the purposes of this paper we have researched and worked on the Google Suites package, specifically on the Google Drive, Docs, and Slides portion of the package.

*STORAGE FORENSICS COMPARISONS*

With the use of cloud computing comes a different interaction between the computer and a user's data. Usually since software is installed and used on a user's machine, all other files and data output from it are saved and locally stored on the user's computer.

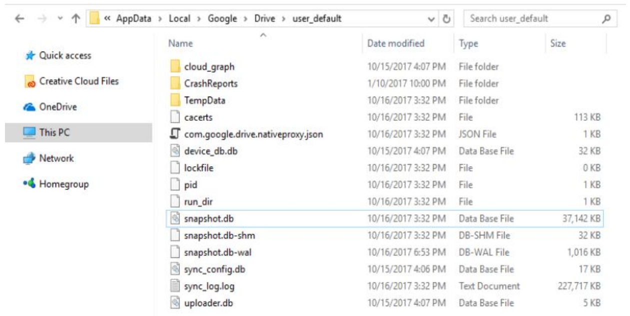
This allows for digital investigators to use software such as the Forensic Tool Kit (FTK) to find logged information and files that a malicious user may try to hide or even delete. This system works perfectly if you have access to the hardware the user is working on. However, since Software as a Service is hosted and run at an offsite location the files and access logs that are typically stored locally are also stored offsite. Because of this other method are needed to locate and find this information. For our purposes the files we want to view names and access logs for are documents and slides stored on a google drive.

### D. Digital Forensics in Relation to Services

Since the files and data are not generally stored or worked on locally normal log and registry files will not be updated with the files and work done. As such the focus must change from the logs of software to the web browsers logs. Web browsers hold information on all sites visited and used during the users' sessions. During our time working on this paper we used Firefox to test and view what data can be extracted from its files. This quickly became quite an issue as most of these files are held in weird file types such as .sqlite, .dump, .log. For the .sqlite files we used a simple database opener so that the file was properly opened and formatted to be read. The log and dump files were a bit different. At the beginning we opened and read through them in plaintext, using find and grep features from text editors and console commands, but this quickly became an issue as discussed in a later section.

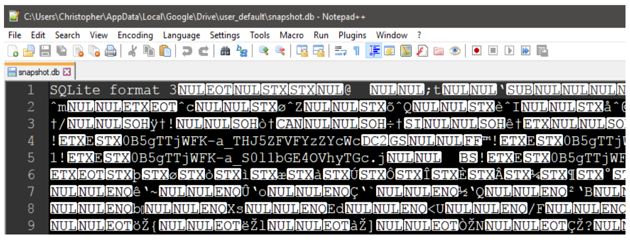
## 3. Google Drive Hidden Application Data

To begin our forensic discovery, we started by pursuing cyber trails and investigating some of the log files. At that time, we discovered many interesting files that can be utilized to determine personal information about the user. For example, their email address, which we found unencrypted within one of the log files. Most of the files we needed to examine for our forensic analysis were located within hidden system folders containing temporary application data. More specifically the folder utilized in our investigation was located at: "C:\Users\User\AppData\Local\Google\Drive\user_default" as shown in figure 1 below.



| Name | Date modified | Type | Size |
|---|---|---|---|
| cloud_graph | 10/15/2017 4:07 PM | File folder | |
| CrashReports | 1/10/2017 10:00 PM | File folder | |
| TempData | 10/16/2017 3:32 PM | File folder | |
| cacerts | 10/16/2017 3:32 PM | File | 113 KB |
| com.google.drive.nativeproxy.json | 10/16/2017 3:32 PM | JSON File | 1 KB |
| device_db.db | 10/15/2017 4:07 PM | Data Base File | 32 KB |
| lockfile | 10/16/2017 3:32 PM | File | 0 KB |
| pid | 10/16/2017 3:32 PM | File | 1 KB |
| run_dir | 10/16/2017 3:32 PM | File | 1 KB |
| snapshot.db | 10/16/2017 3:32 PM | Data Base File | 37,142 KB |
| snapshot.db-shm | 10/16/2017 3:32 PM | DB-SHM File | 32 KB |
| snapshot.db-wal | 10/16/2017 6:53 PM | DB-WAL File | 1,016 KB |
| sync_config.db | 10/15/2017 4:06 PM | Data Base File | 17 KB |
| sync_log.log | 10/16/2017 3:32 PM | Text Document | 227,717 KB |
| uploader.db | 10/15/2017 4:07 PM | Data Base File | 5 KB |

Fig. 1. Hidden Application Data Root Folder


C:\Users\Christopher\AppData\Local\Google\Drive\user_default\snapshot.db - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
snapshot.db

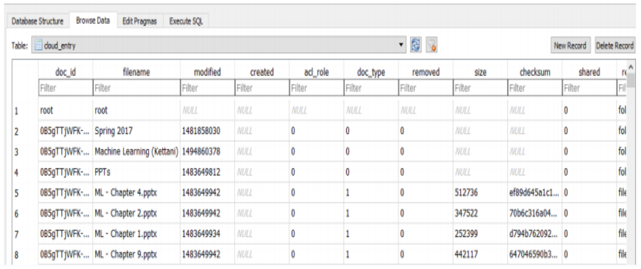
```
SQLite format 3NULEOTNULSTXSTXNUL@  NULNUL;tNULNUL`SUBNULNULNULN
^mNULNULETXEOT^cNULNULSTXø^ZNULNULSTXõ^QNULNULSTXè^INULNULSTXå^@
†/NULNULSOHÿ†!NULNULSOHò†CANNULNULSOH÷†SINULNULSOHê†ETXNULNULSO
!ETXESTX0B5gTTjWFK-a_THJ5ZFVFYzZYcWcDC2GSNULNULFF™!ETXESTX0B5gTTjW
1!ETXESTX0B5gTTjWFK-a_S01lbGE4OVhyTGc.jNULNUL  BS!ETXESTX0B5gTTjWF
ETXEOTSTXþSTXøSTXòSTXìSTXæSTXàSTXÚSTXÔSTXÎSTXÈSTXÂSTX¼STX¶STX°S
NULNULENQê'~NULNULENQÛ'oNULNULENQÇ'`NULNULENQ½'QNULNULENQ²'BNUL
NULNULENQbNULNULENQXsNULNULENQEdNULNULENQ<UNULNULENQ/FNULNULENQ
NULNULEOTöŽ{NULNULEOTëŽlNULNULEOTàŽ]NULNULEOTÒŽNNULNULEOTÇŽ?NUL
```

Fig. 2. Unreadable database file open within notepad++.

Locating these files required going into the Windows folder/file view properties, enabling hidden files/folders and navigating through the temporary application data. Upon our initial inspection, where we observed the files in question via notepad, we were unable to determine any useful information which can be seen in figure 2, which leads us to our next section.

## 4. Findings in SQL Databases

After using the SQLite browser to look inside these database files we found multiple interesting and useful files. These files basically contain a database with multiple tables containing all the information we could possibly want. This includes file names, create dates, modified times, if sharing has been turned on and so on. The first file we investigated was the snapshot.db file, more specifically, the cloud_entry table which is embedded within this file which can be seen in figure 3.



| | doc_id | filename | modified | created | acl_role | doc_type | removed | size | checksum | shared | re |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Fil |
| 1 | root | root | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 0 | fol |
| 2 | 0B5gTTjWFK-... | Spring 2017 | 1481858030 | NULL | 0 | 0 | 0 | NULL | NULL | 0 | fol |
| 3 | 0B5gTTjWFK-... | Machine Learning (Kettani) | 1494860378 | NULL | 0 | 0 | 0 | NULL | NULL | 0 | fol |
| 4 | 0B5gTTjWFK-... | PPTs | 1483649812 | NULL | 0 | 0 | 0 | NULL | NULL | 0 | fol |
| 5 | 0B5gTTjWFK-... | ML - Chapter 4.pptx | 1483649942 | NULL | 0 | 1 | 0 | 512736 | ef89d645a1c1... | 0 | file |
| 6 | 0B5gTTjWFK-... | ML - Chapter 2.pptx | 1483649942 | NULL | 0 | 1 | 0 | 347522 | 70b6c316a04... | 0 | file |
| 7 | 0B5gTTjWFK-... | ML - Chapter 1.pptx | 1483649934 | NULL | 0 | 1 | 0 | 252399 | d794b762092... | 0 | file |
| 8 | 0B5gTTjWFK-... | ML - Chapter 9.pptx | 1483649942 | NULL | 0 | 1 | 0 | 442117 | 647046590b3... | 0 | file |

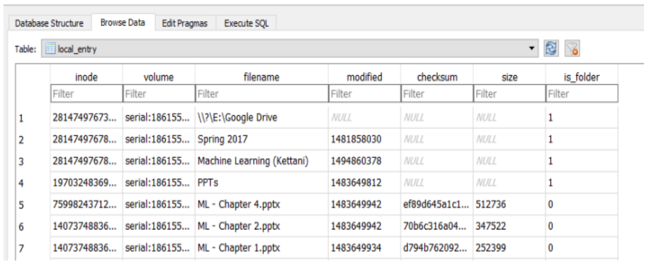
Fig. 3. Snapshot.db cloud entry table

Some of the found data related to

- File name
- Created (UNIX Timestamp)
- Modified (UNIX Timestamp)
- URL
- Checksum (MD5 hash)
- Size
- Shared

Within the same snapshot.db file we also investigated the local_entry table. Which can be seen in figure 4. This contained data relating to the

- File name
- Modified (UNIX Timestamp)
- Checksum (MD5 hash)
- Size

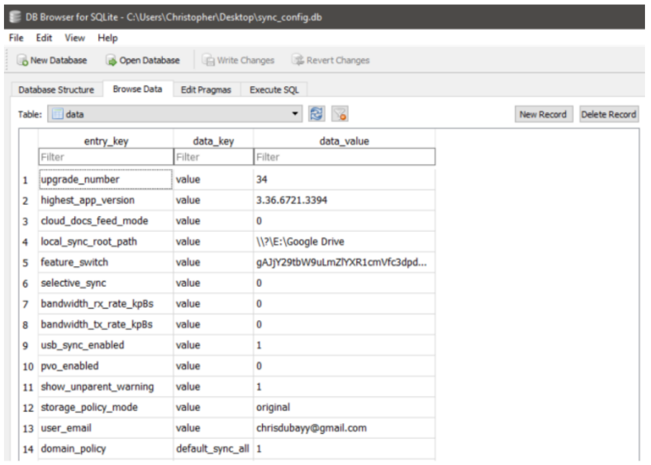


| | inode | volume | filename | modified | checksum | size | is_folder |
|---|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 28147497673... | serial:186155... | \\?\E:\Google Drive | NULL | NULL | NULL | 1 |
| 2 | 28147497678... | serial:186155... | Spring 2017 | 1481858030 | NULL | NULL | 1 |
| 3 | 28147497678... | serial:186155... | Machine Learning (Kettani) | 1494860378 | NULL | NULL | 1 |
| 4 | 19703248369... | serial:186155... | PPTs | 1483649812 | NULL | NULL | 1 |
| 5 | 75998243712... | serial:186155... | ML - Chapter 4.pptx | 1483649942 | ef89d645a1c1... | 512736 | 0 |
| 6 | 14073748836... | serial:186155... | ML - Chapter 2.pptx | 1483649942 | 70b6c316a04... | 347522 | 0 |
| 7 | 14073748836... | serial:186155... | ML - Chapter 1.pptx | 1483649934 | d794b762092... | 252399 | 0 |

Fig. 4. Snapshot.db local entry table.

We also investigated the sync_config.db file, which can be seen in figure 5. This contained data relating to the

- Client version installed
- Local Sync Root Path
- User Email



| | entry_key | data_key | data_value |
|---|---|---|---|
| | Filter | Filter | Filter |
| 1 | upgrade_number | value | 34 |
| 2 | highest_app_version | value | 3.36.6721.3394 |
| 3 | cloud_docs_feed_mode | value | 0 |
| 4 | local_sync_root_path | value | \\?\E:\Google Drive |
| 5 | feature_switch | value | gAJjY29tbW9uLmZlYXR1cmVfc3dpd... |
| 6 | selective_sync | value | 0 |
| 7 | bandwidth_rx_rate_kpBs | value | 0 |
| 8 | bandwidth_tx_rate_kpBs | value | 0 |
| 9 | usb_sync_enabled | value | 1 |
| 10 | pvo_enabled | value | 0 |
| 11 | show_unparent_warning | value | 1 |
| 12 | storage_policy_mode | value | original |
| 13 | user_email | value | chrisdubayy@gmail.com |
| 14 | domain_policy | default_sync_all | 1 |

Fig. 5. Sync_config.db data table.

With the information mentioned above, an investigator would have everything they would need, bar the actual files themselves, to conduct a thorough investigation and have admissible circumstantial evidence in a court of law.

During our investigation, we also browsed over the Sync_log.log file. In this file we found lots of useful information such as file names, dates, modified, sync sessions, file created, file saved, file deleted etc. In one particular log file, there were approximately 650,000 lines of text. Because of the detail of this information we decided it would be best to find a way to parse through the information.

### E. Log Examination Dilemma

While the sqlite file were easily opened and read in the sqlite browsing program, we did not have as much luck with the log files. To start the files like places.sqlite contain all of the history information available from the user, since the browser was implemented. As such the files are very large and cumbersome to work with, especially with just a basic find command at your disposal. In addition the files contain unique characters that cannot be read by any text editors. These unique characters in fact cause most of the problems with reading and parsing through the files. Since they are not contained in normal ASCII conventions most programs crash upon getting to them, and others can have mixed results, as the characters mess with how the find feature is run. These files however contain extremely useful information to an investigator, which is why we devised our own program to circumnavigate these issues.

## 5. Log Parser

In order to expedite our forensic analysis of the large log files we decided to design a file parser that takes the static file names we have found and grabs out only pertinent information from them. This would be things such as the doc names, if they are shared and a few other choice items. We decided to implement this with Java, as we wanted maximum cross platform compatibility. Now this may not seem to relate to the information we gained from the database, however there's one particular useful thing held in the log files but not the .db files. The log files can contain the URL link that shares a document to many people. As such if the files are listed in the database as shared it is possible to parse through the logs to find a link to gain access to the file.

In addition to the possibility of finding links to a user's files, there is also the possibility to find what files have been worked on and edited with the correct keywords. The personal three that we find to be the most useful are usp=sharing as this is used to end google drive files that have been shared. docs.google and drive.google are two others that work well in showing what files have been accessed and used by the user. Our desire in making this parser was to create a simple to use and effective alternative to the grep command, which is why we elected to make it console based, and not in a GUI. In addition, by including tracking on the number of instances a keyword is used, the program itself can also output the number of times a particular URL was accessed. While this would not seem pertinent to start multiple visits to an illicit or illegal site can show intent in a court of law.

**6. Log Parser Code Explanation**

Below is the code devised and written up for the log parser. The design is made to read through and output the data and links that the user desires.

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
```

The additional functions used to design and execute this program are as follows:

```
   Scanner Keyboard = new Scanner(System.in);
        System.out.print("Please enter The file to use: ");
                 String fileIn = Keyboard.next();
```

Opens up a scanner function that reads in user input. The user is prompted for the file that data should be parsed from

```
    System.out.print("Please enter The text to look for: ");
                 String word = Keyboard.next();
```

Prompts the user to input a keyword or phrase they are looking for.

```
    String historyHolder = "";
     int [] indexHolder;
     int count = 0;
     String line;
     String[] data;
     int arrayCounter;
```

Defines variable for use throughout the program.

```
      try {
        BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream(fileIn),"UTF-8"));
```

Begins a try statement to bring in the data from the user-inputted file. The distinction of UTF-8 was made as the encoding method as it strips the unknown characters previously causing crashes with other text editors and programs.

```
      while ((line = br.readLine()) != null) {
                 historyHolder =
historyHolder.concat(line);
                  }
```

Begins to bring in the data from the file. Places the data in a large string variable called historyHolder that is used to parse out the data from later on.

```
br.close();
} catch (FileNotFoundException e) { e.printStackTrace(); }
catch (IOException e) { e.printStackTrace();
             }
```

Closes the reader and sets up error catching in the case that an error is found. This is done to catch the program if the file a user inputted is not located or if a problem occurs when reading the file.

```
        while (!word.equals("Stop")){
                  count = 0;
```

Starts a while loop that is setup to allow multiple keywords to be tested before closing out of the program. Count is set back to 0 preventing issues with doubling up on the count size.

```
     for (int i = -1; (i = historyHolder.indexOf(word, i + 1))
        !=    -1; i++) {
                  count++;
             }
```

Retrieves a count of all instances that the keyword appears in the selected file. This is done as a precursor to actually grabbing the necessary data as well as to be outputted later on.

```
        indexHolder = new int[count];
        int arrayCounter = 0;
```

Sets up the holder of the index number of our keyword in an array fashion, and sets up a counter to run through the array in the next step.

```
for (int i = -1; (i = historyHolder.indexOf(word, i + 1))
 != -1; i++) {
            indexHolder[arrayCounter] = i;
            arrayCounter++;
    }
```

These lines of code are the backbone of our data retrieval effort. This works as a for loop that finds the instances of the data we want to retrieve and increments our counter by one to keep track of how many instances of our keyword we have found. In addition this takes in the positional data for said keywords. This is vital for the upcoming steps.

```
data = new String[arrayCounter];
int begin;
int end;
char x;
```

We assign the data array the number of instances it will need to take in, gotten from our previous set, and sets up three variables for use in grabbing the data we are to retrieve.

```
for (int y = 0; y < arrayCounter; y++){
            begin = indexHolder[y];
            end = indexHolder[y];
            x = historyHolder.charAt(begin);
```

Set up to run through all of the instances of the keyword and place the corresponding indexes in beginning and ending variables so that they can be used to find the beginning and end of the URL.

```
while(x != ' %'){
        begin--;
        x = historyHolder.charAt(begin);
         }
```

This while loop is designed to continually run until a percentage sign is found in the String containing all of the history information. This is designed to find the first character in the URL as all of the URLs that have been found precede the http request with a percentage mark. Since the parser is used to find which URL's have been accessed by the user this has been determined to be the best way to find its beginning.

```
x = historyHolder.charAt(end);
while(x != ' '){
    end++;
    x = historyHolder.charAt(end);
   }
```

This statement is exactly the same as the previous except the while statement is in reverse, searching for upcoming blank spaces that signify the end of a URL.

```
data[y] = historyHolder.substring(begin, end);
     System.out.println(data[y]);
```

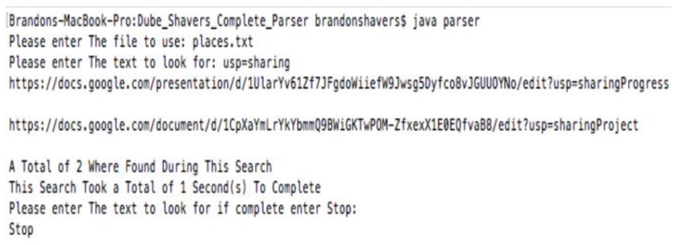
The found URLs are then moved to a separate array for storage and printing to screen, along with the total number of instances found.

```
    System.out.println("Please enter The text to look
for
    if complete enter Stop: ");
        word = Keyboard.next();
     }
     Keyboard.close();
```

Finally, we check with the user to start another search, or they can decline with Stop. This is designed this way as previously mentioned the file's data takes around ten minutes to be properly placed into useable storage so it is much easier to run multiple searches once it is loaded than one per run of the program.

```
long startTime = System.currentTimeMillis();
long endTime = System.currentTimeMillis();
long totalTime = endTime - startTime;
```

The addition of time is done as quality of life updates. They were done to help relay how much total time it took to find a user's query. The code output is shown in figure 6.



```
Brandons-MacBook-Pro:Dube_Shavers_Complete_Parser brandonshavers$ java parser
Please enter The file to use: places.txt
Please enter The text to look for: usp=sharing
https://docs.google.com/presentation/d/1UlarYv61Zf7JFgdoWiiefW9Jwsg5Dyfco8vJGUUOYNo/edit?usp=sharingProgress

https://docs.google.com/document/d/1CpXaYmLrYkYbmmQ9BWiGKTwPOM-ZfxexX1E0EQfvaB8/edit?usp=sharingProject

A Total of 2 Where Found During This Search
This Search Took a Total of 1 Second(s) To Complete
Please enter The text to look for if complete enter Stop:
Stop
```

Fig. 6. Parser Program Output

## 7. Differences Between Parsers

The reasoning behind the need of a newly constructed parser was due to the fact that current parsers did not do exactly what we were looking for. We attempted grep commands as well as normal find commands on the plain text versions of some of our files. These where done in normal applications such as TextEdit and notepad. This, however, always looped back to the same problem with special characters. With the discovery of the file sharing links being valid for months after their initial creation we came to the conclusion that this would be a necessary step in this research. This is due to not only having a chance to access and see what some of the files contained but also to have access to how many times they accessed them. While there are a variety of tools made as specific log parsers, they too were designed around the SQL counterparts of the log files, which for our circumstances of finding exact URL's would not work.

In addition, even the few that could relay this information, neglected to show the number of instances it was accessed. While this is the case, a normal log parser that works through SQL works within a fraction of the time. This is mainly due to the file format not having to be changed in order to read in this style of file. A normal SQL file is formatted and designed compactly so that a designated log reader can read it. In our case the history file has its own special characters that can be more easily read by a browser. Because of this it removes the ease of readability with outside programs and for this instance, what we are trying to do. This being said our design is a streamlined version, specifically designed to find URL's and how many times they have been accessed, which does not have a large user base outside of digital forensics.

One does not really understand the scope in which browsers and software log you're every action and move until one jump head deep into it. The files and data we parsed through ended up being quite large and mostly unusable at the start. The interesting part was the gems we gathered from the rough lumps of coal we began with. Things like the sharing links being usable months after creation was completely unthinkable to us, and it was extremely gratifying to be able to find and parse through the data to find new ways to approach problems. While we began thinking we would find file names and account information at most, we happily call this a success with accomplishing this and much more.

## 8. Conclusion

While digital forensics is an ever-changing field, as is the field of Computer Science in general, our methods of data finding must be just as flexible. Since cloud computing removes a majority of the locally hosted information on files accessed and used, we are required to think outside of normal terms and find unique ways to retrieve the necessary information. Things such as browser log files, backup folders and user information that is stored from browsers or the applications link to the server all become viable and useful files in finding this information. In addition, workarounds and forward thinking are necessary to continually move forward in this field. Problems such as logs being virtually unusable is a hurdle that has to be crossed, a perspective that needs to be changed so that new useful insight can be gained.

## REFERENCES

[1] Chung, Hyunji, et al. "Digital Forensic Investigation of Cloud Storage Services." Digital Investigation, vol. 9, no. 2, 2012, pp. 81–95., 18 Sept. 2017.

[2] Price, Rob. "Google Drive Now Hosts More than 2 Trillion Files." Business Insider, Business Insider, 6 May 2017, www.businessinsider.com/2-trillion-files-google-drive-exec-prabhakar-raghavan-2017-5.

[3] "First Character of the Reading from the Text File:" Stack Overflow: stackoverflow.com/questions/17405165/first-character-of-the-reading-from-the-text-file-%C3%AF.

[4] K. Kaur and X. Xiaojiang Du and K. Nygard, "Enhanced routing in Heterogeneous Sensor Networks", IEEE Computation World'09, pp. 569-574, Athens, Greece, Nov. 15-20, 2009.

[5] Lauren Evanoff, Nicole Hatch, Gagneja K.K., "Home Network Security: Beginner vs Advanced", ICWN, Las Vegas, USA, July 27-30, 2015.

[6] Gagneja K.K. and Nygard K., "Heuristic Clustering with Secured Routing in Heterogeneous Sensor Networks", IEEE SECON, New Orleans, USA, pages 51-58, June 24-26, 2013.

[7] Gagneja K.K., "Knowing the Ransomware and Building Defense Against it - Specific to HealthCare Institutes", IEEE MobiSecServ, Miami, USA, pp. 1-5, Feb. 11-12, 2017.

[8] Gagneja K.K., "Secure Communication Scheme for Wireless Sensor Networks to maintain Anonymity", IEEE ICNC, Anaheim, California, USA, pp. 1142-1147, Feb. 16-19, 2015

[9] Gagneja K.K., "Pairwise Post Deployment Key Management Scheme for Heterogeneous Sensor Networks", 13th IEEE WoWMoM 2012, San Francisco, California, USA, pages 1-2, June 25-28, 2012.

[10] Gagneja K.K., “Global Perspective of Security Breaches in Facebook”, FECS, Las Vegas, USA, July 21-24, 2014.

[11] Gagneja K.K., "Pairwise Key Distribution Scheme for Two-Tier Sensor Networks", IEEE ICNC, Honolulu, Hawaii, USA, pp 1081-1086, Feb. 3-6, 2014.

[12] Gagneja K., Nygard K., "Energy Efficient Approach with Integrated Key Management Scheme for Wireless Sensor Networks", ACM MOBIHOC, Bangalore, India, pp 13-18, July 29, 2013.

[13] Gagneja K.K., Nygard K., "A QoS based Heuristics for Clustering in Two-Tier Sensor Networks", IEEE FedCSIS 2012, Wroclaw, Poland, pages 779-784, Sept. 9-12, 2012.

[14] K. K. Gagneja, K. E. Nygard and N. Singh, "Tabu-Voronoi Clustering Heuristics with Key Management Scheme for Heterogeneous Sensor Networks", IEEE ICUFN 2012, Phuket, Thailand, pages 46-51, July 4-6, 2012.

[15] Gagneja K.K., Nygard K., "Key Management Scheme for Routing in Clustered Heterogeneous Sensor Networks", IEEE NTMS 2012, Security Track, Istanbul, Turkey, pp. 1-5, 7-10 May 2012.

[16] Runia Max, Gagneja K.K., “Raspberry Pi Webserver”, ESA, Las Vegas, USA, July 27-30, 2015.

[17] A. S. Gagneja and K. K. Gagneja, "Incident Response through Behavioral Science: An Industrial Approach," 2015 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, 2015, pp. 36-41.

[18] Tirado E., Turpin B., Beltz C., Roshon P., Judge R., Gagneja K., “A New Distributed Brute-Force Password Cracking Technique”, Future Network Systems and Security, FNSS Communications in Computer and Information Science, vol. 878, pp 117-127, 2018

[19] Caleb Riggs, Tanner Douglas and Kanwal Gagneja, "Image Mapping through Metadata," Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC), Shanghai, China, 2018, pp. 1-8.

[20] Keely Hill, Gagneja K.K., “Concept network design for a young Mars science station and Trans-planetary communication", IEEE MobiSecServ 2018, Miami, FL, USA, Feb. 24-25, 2018.

[21] Javier Campos, Slater Colteryahn, Gagneja Kanwal, “IPv6 transmission over BLE Using Raspberry PI 3", International Conference on Computing, Networking and Communications, Wireless Networks (ICNC'18 WN), March 2018, pp. 200-204.

[22] Gagneja K., Jaimes L.G., “Computational Security and the Economics of Password Hacking”, Future Network Systems and Security. FNSS 2017. Communications in Computer and Information Science, vol. 759, pp. 30-40, Springer, 2017.

[23] Gagneja K.K. Ranganathan P., Boughosn S., Loree P. and Nygard K., "Limiting Transmit Power of Antennas in Heterogeneous Sensor Networks", IEEE EIT2012, IUPUI Indianapolis, IN, USA, pages 1-4, May 6-8, 2012.