Skyler Booth
Georgia Briskey
Marcus Mabry
Alex Trimpe
Legal and Social Informatics of Security
Sameer Patil
May 5th, 2017

Team 7 Final Project Report

Introduction

Over the course of the Spring 2017 semester, our team chose to complete the task of "Building a scraper to pull news articles related to privacy incidents (e.g., data breaches) that involve regulatory/legal components...The scraped articles would need to be checked to ensure high precision and then used to populate a database of news articles about privacy incidences".

A scraper is a tool and technique used to extract specific data from other data, or in our case, a website. With our previous experience of scraping in Python, we chose this topic because it not only applied our current Python skills, but also challenged us to extend these skills and apply them in a different context.

With the help of Python and SQL Documentation in addition to Professor Patil's advice, we were able to develop a step-by-step process to achieve our task's end goal. For this process, we developed Python code that searched through articles within the *Chicago Tribune* with keywords pertaining to security. Our Python code was able to extract articles, ensure they were involved with law and regulations, and populate them into our database. We were then able to analyze these articles to look for any trends or patterns.

From the populated articles, we were able to conclude that the density of privacy incidents involving legal/regulatory components has only increased per year since the earliest article we collected.

Approach

To approach this project, we developed a strategy made up of small steps, all with their own end goals. We first chose the languages we were going to implement, which were Python and MySQL based on our familiarity and previous experiences with the two. With MySQL decided on, we created the database in which the extracted articles would be populated into. With this basic foundation, we moved onto our Python code, which needed to include three parts to become fully functional. The first part was a program named 'Story Finder', which used our given keywords to search the *Chicago Tribune* for articles and if they matched, returned the set of articles, otherwise they were thrown out. Our second program was the "Story Checker" and its purpose was to narrow our article set by checking to see if they had legal/regulatory components to them. Story Checker is able to do this by searching the articles to find legal/regulatory keywords. Our second program was the 'Story Checker', which narrowed our article set by checking to see if they had legal/regulatory components to them by searching within these articles with given legal/regulatory keywords. The Story Checker program either added articles to a dictionary if it pertained to the law, or discarded them if it did not. Finally, our third program was the Database Entrance, which entered the articles and their information into our database.

Once the functions were combined, we ran our code on a Linux server and were able to see the returned articles. From here we began to document our process and make any changes to make for a more efficient process and ultimately successful final project.

Challenges

Throughout the project, we ran into multiple problems such as: Deciding on a source, figuring out how to access all stories, deciding on keywords to search with, advertisements and sponsored stories, Python's Re.Findall, and general coding bugs. Our first major challenge was deciding on a source, with the sheer volume of new sources out there, it was hard to narrow down one. Also while we were searching for a news source, we ran into our second issue, not being able to directly access a database where all of the news stories were stored. After we had some guidance from Professor Patil, we finally decided on the *Chicago Tribune*, which also had our second issue of not having an accessible database of the news stories it produced. We realized the search function on the homepage would return any story that the *Chicago Tribune* had produced that matched the keywords that were search, thus our second major issue had been solved but it added our next challenge, which was deciding on privacy incident keywords to search.

This was an issue because none of the articles we read for class directly discussed keywords for privacy incidents and we also didn't want so many that it would cause our program to run for hours. After we read through the database that our project was based off of, we chose 5 key phrases that they used for their search to conduct ours. After we did some initial test searches on the *Chicago Tribune* site, we discovered that the search results would include advertisement and sponsored stories. Analysis of our story URLs showed that all the stories that were sponsored had the string 'redeye' in the URL, which we concluded was the program that sponsored the stories. So we decided that any story that had 'redeye' in the URL would be dropped from the URL set.

Our next issue came with the 'Story Checker' program. It would search through the documents to see if the contained legal or regulatory keywords. One of those keywords was privacy and every story had a link to the *Chicago Tribune*'s Privacy Policy, which meant every story would pass the test if it had legal or regulatory components. To get a better understanding of this particular challenge, see figures 15 & 16. See Figures 15 & 16 to get a better understanding. Instead of searching the entire HTML page, we found that we could use Re.Findall to pull out the article body and only search through it.

Another issue we ran into was with Python's Re.Findall module. This module is meant for searching through a body of text and extracting the text between a certain starting point and ending point. For example, if you had a string "The quick brown fox jumps over the lazy dog" and your starting point was 'quick' and your ending was 'fox' Re.Findall should return ' brown '. However, this wasn't always the case. It took a long time to narrow down the starting and ending points for every single piece of data we needed to extract in order to get it to work right. Even after all this work it would still have some issues that we couldn't fix, so Re.Findall was a major headache that we had to deal with.

Our last major challenge was just general coding issues, we thought we had everything done and we would go and run the code and some error or issue would pop up. These were

mostly minor errors but all of them added up to a major challenge that kept adding on the hours that this program took to complete. However, we eventually overcame all of these issues and created a bug-free, working Incident Finder program.

Job Roles

Each group member in our team had a different role. Skyler Booth was the lead Python programmer, Alex Trimpe was the tester, Marcus Mabry was the project strategist/designer, and Georgia Briskey was the database administrator.

Presentations

As a group we had to split each section of our presentation drafts into four parts. The first two briefings we did in lecture really helped us to analyze the audience and get feedback on how each of us should improve for the final presentation. The coding was split equally with everyone helping here and there to modify and edit things when needed. Georgia specifically focused on the idea of using a scraper for the project and introducing the topic in greater detail. Marcus was focused on our strategy to accomplish our goal of scrapping the news articles and the approach we had to take by dividing it into three sections. Skyler made sure all the code worked properly and did exactly what we needed to do and he also combined the three sections together to form the final working parts. Alex had the task of finding some of the articles we scraped and making sure they contained information relevant to our topic. He also documented what issues came up so that we could fix them before the final draft. Once everyone did their parts individually, we met up to compare ideas on how to revise our work so that is flowed together. Once we did this, we combined everything together into one file for submission.

Final report

Similar as we did on the presentations, we split the final report up into equal parts and then came back together to compare and revise what we had done. Since each section was dependent on the others, we really had to communicate to get them to flow in a way it would make sense in a written report. We realized that since most of our project consisted of code, we would struggle a bit trying to do a write up to describe actions as such.

Code Overview

These next few sections will be about the goal of each program and how our code works piece by piece. It will be a higher view of how the code runs and what each specific program does in order for someone who may not be familiar with Python programing can better understand at a basic level how our algorithm works. All of the backup files run very similar to the original code with the exception that they save their results in a Pickle document in order to be ran piece by piece.

Database/CSV

The database and CSV is where we stored all of our privacy incident stories that that involved regulatory/legal components from the *Chicago Tribune*. We decided that the database/CSV should hold basic information of the stories, enough to understand what the article is about, but not too much to the point of unnecessary information. The basic information

obtained from each article included the article URL (which will allow anyone to view the entire story), the author, the publishing date, and the article description that the *Chicago Tribune* provided (which will give a basic understanding of the story). Although the prompt didn't require it, we decided to add all the stories to a comma-separated file (CSV) for better usability in the future. For example, if someone wants to analyze the data, make the data accessible offline and make it more transferable, having a CSV will make this much easier.

Story Finder

As the name of the program would suggest, the objective of Story Finder is to connect to the *Chicago Tribune* and find all of the stories that relate to privacy incidents. How does this program find the stories? As mentioned before, the *Chicago Tribune* has no way to access a database with all of their stories. To circumvent this, we used their search function. Our code took a list of keywords we developed ourselves (e.g. technology data breach, technology privacy, ETC.) that would return the most accurate results for privacy incident related articles. This program went through each of the keywords in the list and would do the following: first, it would create the base URL's for the search page results, since page 1 and page 2+ of the results had 2 different URL setups. Then the program would check what search results page it was on and create the full URL and request the page. After the page was received, it was downloaded and decoded. Story Finder then used a basic Python program Re.Findall to do its searching. The first information checked was the number of results, because we discovered that if the page following the last page of results didn't error out, it would change the entire result number to 0. For example, keywords 'technology' + 'data breach' had 34 pages of results and when the program would ask for page 35, the *Chicago Tribune* responded with a search page displaying '0 Results' rather than returning an error. If this were the case, the program would move on to the next keyword set. If this weren't the case, the program would use Re.Findall to find all the URLs of the stories in the search result. Story Finder would then add each story to list 'storyURLSet' and add one to the page counter, unless it contained the string 'redeye', which meant it was an ad/sponsored story. After each result search page was scanned with each keyword, the program returned the storyURLSet.

Story Checker

The goal of this program is to ensure the stories collected from Story Finder contained regulatory/legal components in addition to gathering the author, title, publish date, and description for each story. This program begins by going through the URLS in storyURLSet and requests the page from the *Chicago Tribune.* After each story is download and decoded, the program goes through the body of the article and checks if it contains at least one of our chosen regulatory/legal keywords (i.e. HIPPA, legal, policy, ETC.) If it does not contain one of the keywords, the story is dropped because it doesn't meet our prompt's criteria. If it does, the program then uses Re.Findall, similar to Story Finder, to find the story's author, publishing date, title, and description. If the story is missing any of these pieces, the program inserts 'None' for the missing field. After this, the program adds the story and its information to the storyURLDict dictionary. After all the URLs have been checked, the program returns the dictionary.

Database Enterance

As the program name would suggest (excuse the spelling), the goal of databaseEnterance.py is to insert all of the stories and their information into the database as well as the CSV file we created. The program starts by connecting to the database, which is currently stored in Skyler's Fall I211 server space. The program then imports the dictionary that the Story Check output. Net the program opens the Incidents CSV file. It the writes the headers to the CSV file (URL, Title, ETC.). Then the program goes through each story in the dictionary and inserts it into the SQL database as well as adding it to a new line on the CSV file. After each story is added, the program lets the user know that the story was successfully added to the database/CSV file. After all stories are done being inserted/added, the CSV file is closed and the program terminates which closes the connection to the database.

Incident Finder

The goal of this program is to allow all the other segments to run together and streamline the incident finding process. This program starts by importing the code from the other three segments, storyFinderFinal.py, storyChecker.py, and databaseEnterance.py. It then runs the Story Finder and saves the URL set that it outputs. Story Checker runs using the URL set created by Story Finder. Finally, it runs Database Enterance using the story URL dictionary found output by Story Checker. After each segment runs, it notifies the user that everything has been executed.

The Running of the Codes

In order to run our code, you will need to download a few programs. For Mac users, you will need terminal, which should already be installed, and Cyberduck. For Windows/PC users, you will need Putty and WinSCP. Terminal and Putty are SSH programs while Cyberduck and WinSCP are File Transfer Programs. If the instructions call for entering in a command, it'll be done within Putty or Terminal. If the instructions call for moving files, you will be using WinSCP or Cyberduck.

Intended Running Using Incident Finder

In order to run the code, a few things must happen. The first thing that needs to be done is creating the database on the db.soic.indiana.edu MySQL server using the code in the IncidentsTableCreationCode file, which is a simple copy and paste (See Fig. 2 and 3). The next step is editing the code within databaseEnterance.py in order to reflect the new user and password (See Fig. 4). This step can be forgone if you would like to access Skyler's MySQL server to check the finished database, see Fig. 5, 6 for the login command and the command to show the database. Finally, the 4 final versions need to be added to the same directory on the IU Linux Server (See Fig. 1). Once these steps have been completed, running the code on the Linux server should be as simple as running the command "python3 incidentFinder.py" in the same directory as the Python Files (see. Fig. 7). After this file runs, you should see the CSV in the same directory as the rest of the code files and the database should be populated and located wherever you specified in the databaseEnterance.py file. Since incidentFinder.py runs quite a lot of code and requires a lot of resources, it may error out only because it runs out of resource space. We ran it multiple times and it only happened on the initial running and as soon as it was ran a second time it worked. We think this is most likely because it cached the URLs and ran

quicker. If this error happens, we do have a second set of backup files that are meant to be ran individually in order to minimize the load.

Backup Option

In order to run the code through this backup option, a few things must happen similarly to before. First, the three backup versions need to be added to the same directory on the IU Linux Server (See Fig. 8). Next, create the database on the db.soic.indiana.edu MySQL server using the code in the IncidentsBackupTableCreationCode file (See Fig. 9 and 10). The final step is editing the code within databaseEnteranceBackup.py in order to reflect the new user and password (See Fig. 4). The last step can be forgone if you would like to access Skyler's MySQL server to check the finished database, see Fig. 5, 11 for the login command and the command to show the database. Once this is all completed, running the code on the Linux server is done in three steps. The commands must be done in the same directory as the Python Files. The first step is running the command "python3 storyFinderBackup.py" (See Fig. 12). This will run the Story Finder and output a Pickle file (the equivalent of saving a Python object such as a dictionary or in this case a set). The next step is running the command "python3 storyCheckerBackup.py" (See Fig. 13). This runs the Story Checker using the Pickled URLSet and outputs the storyURLDict as a Pickle file. Finally, run the command "python3 databaseEnteranceBackup.py" (See Fig. 14). After this runs, you should see the CSV in the same directory as the rest of the code files and the database should be populated and located wherever you specified in the databaseEnteranceBackup.py file.

If you have any issues or difficulties please see figures 17 and 18. They are instructional videos for how to run either incident finder or the backup files. If you still have issues, the best option would be to contact Skyler Booth, spbooth@umail.iu.edu, he should be able to assist you.

Connection to Class

Our findings, and in fact our project as a whole is in direct relation to the things in which we have looked at and studied during this semester of class. Articles like that which discussed the Right to be Forgotten, which was established in Europe, would be an example of an article that our web scraper would have pulled due to its legal components that surround the privacy and security objective (Hern, 2016). While this is an article that is arguing for a change in the laws, it is important and relevant due to the fact that it is a precursor for the way we inevitably wrote the laws. Similarly, looking back at articles like the Warren and Brandeis article, *The Right to Privacy*, should we use this scraper for an extended period of time, pulling articles daily and adding them into a database, we would be able to analyze the differences and similarities that privacy and security take in the law, both currently and in the past ("The Right to Privacy", 1890). This concept is even further exemplified in the older paper by Warren and Brandeis illustrates this duality for us in the modern day.

It is also a challenge in the modern day to be certain that you are complying with all necessary legal components as illustrated in the article, *Evaluating existing security and privacy requirements for legal compliance,* by Massey et. al. Having a scraper like this would allow a company to be able to analyze upcoming legal changes as well as maintain a required level of compliance while adapting to changes and updates to the laws (Massey, 2010). This is especially the case when looking at careers that deal with the healthcare sector. Because laws are always changing, and especially now, regarding healthcare, this is evermore an important consideration

for companies. This is highlighted in the paper from class, *Understanding the Drivers and Outcomes of Healthcare Organizational Privacy Responses* by Parks et. al, showing the need to continually analyze changes and adapt to new laws, areas of business, and technological innovation. This not only applies to healthcare, but also applies to the cloud and data storage across the world (Parks, 2011). The video we watched in class, *Microsoft Privacy and Compliance in the Cloud*, is a prime example of how this is something Microsoft has to dedicate a sizeable amount of energy and resources to ensuring compliance (Microsoft Privacy, 2015).

Connection to Social Context

Having a project such as this provides a way in which we are able to see how laws and new stories are an ever evolving element that can not only be extremely difficult to comply with, but also extremely tedious to restructure business and operations with each change. Just with the amount of results that we were able to pull from a single news source, it does not take much imagination to think through a scenario in which a new regulatory demand was issued but missed by a company. This could result in quite a loss for a business, both in finances and time. Having the ability to have a system that could alert a company prematurely before issues have the chance to arise could eliminate these negative side effects that currently come with change. While we currently have the program set to check websites only on manual execution, it would not require much work to re-do it to have it run every few minutes and populate the database with new articles.

It goes beyond just the professional world though, as this could be used to the study the changes made throughout time concerning new laws that deal with privacy and security. As time progresses, so does the feeling that people collectively have towards their privacy and security. Therefore, the laws always adapt to suit these newfound needs that society have collectively agreed upon. This could be plotted to be able to see the transitions that occur as new generations as well as new technologies come into existence. As these things come out, there will always be new ways in which we think about our own privacy and security and adapt them into laws. Just as seen in the article of Warren and Brandeis, some things that are critical to have for one generation are seen as less important with the development of new technologies that require the loosening of some privacies. The benefits of the new technologies outweigh the need for these privacies and securities so users are willing to give them up. This scraper could allow researchers to document the exact times in which these changes occur.

Conclusion

The results of our work was a populated database with 616 entries that contained stories for a single website. This database held articles with legal or regulatory components, which discussed privacy and security topics. We were able to successfully eliminate articles that were not relevant to the criteria, which we were looking for and were able to compile a rather large data set with which to use. By using various keywords and eliminating articles that do not contain those keywords, we are able to search for any general topics and find news stories containing or relating to these topics. This was a great way for us to expand what we have learned within the classroom as we have shown in the previous paragraphs relating our findings to articles discussed in this course. In the future, we would be able to take these and keywords and expand our search to apply for any study that could benefit from analyzing the articles and the way in

which they progress over time and relate to each other. Hopefully, what we have done will be able to be expanded on for our needs in the future and something that we can actually utilize is later studies. What we learned in class was absolutely beneficial for helping us to create this project and be able to execute it effectively and critically.

Figures

Figure 1

```
[spbooth@silo i330]$ cd final
[spbooth@silo final]$ ls
databaseEnterance.py  incidents.csv    storyFinderFinal.py
incidentFinder.py     storyChecker.py
[spbooth@silo final]$
```

Figure 2

```
CREATE TABLE incidents
(
id INT AUTO_INCREMENT,
url VARCHAR(256),
title VARCHAR(256),
author varchar(256),
postdate varchar(256),
description varchar(1000),
PRIMARY KEY (id)
)
Engine=innodb;
```

Figure 3

```
MariaDB [i211f16_spbooth]> CREATE TABLE incidents
    -> (
    -> id INT AUTO_INCREMENT,
    -> url VARCHAR(256),
    -> title VARCHAR(256),
    -> author varchar(256),
    -> postdate varchar(256),
    -> description varchar(1000),
    -> PRIMARY KEY (id)
    -> )
    -> Engine=innodb;
```

Figure 4

```
def databaseEnterance(dic):
    string = "i211f16_spbooth" #change username to yours!!!
    password = "my+sql=i211f16_spbooth" #change username to yours!!!
```

Figure 5

```
[spbooth@silo final]$ mysql -h  db.soic.indiana.edu  -u  i211f16_spbooth --pass=
my+sql=i211f16_spbooth -D i211f16_spbooth
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 657310
Server version: 5.5.52-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Figure 6

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 657337
Server version: 5.5.52-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [i211f16_spbooth]> select * from incidents;
```

Figure 7

```
[spbooth@silo final]$ pwd
/u/spbooth/i330/final
[spbooth@silo final]$ python3 incidentFinder.py
```

Figure 8

```
[spbooth@silo i330]$ cd backup
[spbooth@silo backup]$ ls
databaseEnteranceBackup.py  storyCheckerBackup.py  storyFinderBackup.py
[spbooth@silo backup]$
```

Figure 9

```
CREATE TABLE incidentsBackup
(
id INT AUTO_INCREMENT,
url VARCHAR(256),
title VARCHAR(256),
author varchar(256),
postdate varchar(256),
description varchar(1000),
PRIMARY KEY (id)
)
Engine=innodb;
```

Figure 10

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 657392
Server version: 5.5.52-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [i211f16_spbooth]> CREATE TABLE incidentsBackup
    -> (
    -> id INT AUTO_INCREMENT,
    -> url VARCHAR(256),
    -> title VARCHAR(256),
    -> author varchar(256),
    -> postdate varchar(256),
    -> description varchar(1000),
    -> PRIMARY KEY (id)
    -> )
    -> Engine=innodb;
```

Figure 11

```
MariaDB [i211f16_spbooth]> select * from incidentsBackup;
```

Figure 12

```
[spbooth@silo backup]$ python3 storyFinderBackup.py
```

Figure 13

```
[spbooth@silo backup]$ python3 storyCheckerBackup.py
```

Figure 14

```
[spbooth@silo backup]$ python3 databaseEnteranceBackup.py
```

Figure 15

Figure 16

```
<a href="http://www.tribpub.com/privacy-policy-and-your-privacy-rights/" class="trb_nls_p" target="_BLANK">Privacy Policy</a
```

Figure 17

Please see the working files for IncidentFinderInstructions.mp4

Figure 18

Please see the working files for BackupIntructions.mp4

Reference List

Hern, A. (2016, May 19). Google takes right to be forgotten battle to France's highest court. Retrieved May 04, 2017, from https://www.theguardian.com/technology/2016/may/19/google-right-to-be-forgotten-fight-france-highest-court

Massey, A.K., Otto, P.N., Hayward, L.J. et al. Requirements Eng (2010) 15: 119. doi:10.1007/s00766-009-0089-5

Microsoft Privacy and Compliance in the Cloud. (2015, January 09). Retrieved May 04, 2017, from https://www.youtube.com/watch?v=q5rwwQBTJxo

MySQL Documentation. (n.d.). Retrieved April 24, 2017, from https://dev.mysql.com/doc/

Parks, R., Chu, C., Xu, H., & Adams, L. (2011). Understanding the drivers and outcomes of healthcare organizational privacy responses.

Python 3.6.1 documentation. (n.d.). Retrieved April 24, 2017, from https://docs.python.org/3/

"The Right to Privacy". (1890, December 15). Retrieved May 04, 2017, from https://groups.csail.mit.edu/mac/classes/6.805/articles/privacy/Privacy_brand_warr2.html