Final Project Report

Team 1 - SDDEC19-01

Team Members: Alec Lones, Jeremiah Brusegaard, Mark Schwartz, Nolan Kim

Faculty Advisor and Client: Benjamin Blakely

Team Website: http://sddec19-01.sd.ece.iastate.edu/

1. Revised Project Design

1.1 Problem Statement

Currently, there is no good way to be notified about every data breach that occurs. Since companies can release breach reports on any medium, including Twitter posts, blog posts, or forum threads, important data breaches can easily fly under the radar. This can be incredibly dangerous for companies because they need to stay up to date with the latest data breaches. Not staying up to date can lead to potential vulnerabilities in their own environments not being addressed. In addition, it is important for these data breaches to be stored for security teams to reference later.

1.2 Proposed Solution

The purpose of this project is to serve as an early warning for CSO's (Chief Security Officer) on breach reports that may affect their company . We plan to do this by implementing a web scraper to traverse the internet and identify data breach reports using machine learning. Our scraper will then store the breach reports in a database for future reference. With this information, the CSO will remain informed on current security threats to their organization.

1.3 Operating Environment

Our machine learning model is stored in a file formatted for Linux machines, and therefore the scraper must be run on a Linux machine with Python 3 installed. The machine must also be able to run a MongoDB database, which the scraper will use to store breach reports. This environment will require that we have constant uptime or at least near constant uptime to allow for uninterrupted web crawling.
In order to make the installation process easier, we have created a virtual machine with all the necessary libraries and programs installed. Therefore, a computer with a VirtualBox installation running virtual machine which meets our minimum specifications, listed below, is required for the best results.

1.4 Minimum Virtual Machine Specifications

OS: Linux 64-bit

Processor: 4 Core CPU

Memory: 16 GB RAM

Network: Constant Internet connection

Storage: 20 GB available space

1.5 Intended Uses and Users

The user for this tool is a CSO or a cybersecurity analyst, who is assumed to have advanced technical capability and the desire to keep up with new breach reports. With our tool, users will be able to see new breach reports and learn about potential security breaches that could have an effect on their company, and react accordingly. The purpose of this tool is to gather, consolidate, and report information, not act on it. Once the user receives information about a breach, they will need to decide on the next course of action themselves.

1.6 Assumptions and Limitations

1.6.1 Assumptions

- Uninterrupted internet connection
- Constant power supply
- Will not get blacklisted from too many DNS requests (Request limit is implemented to avoid this)
- Analyst training the machine learning model will be fully competent in what a breach report is or is not
- There won't be resource limitations for crawling - hardware, etc.

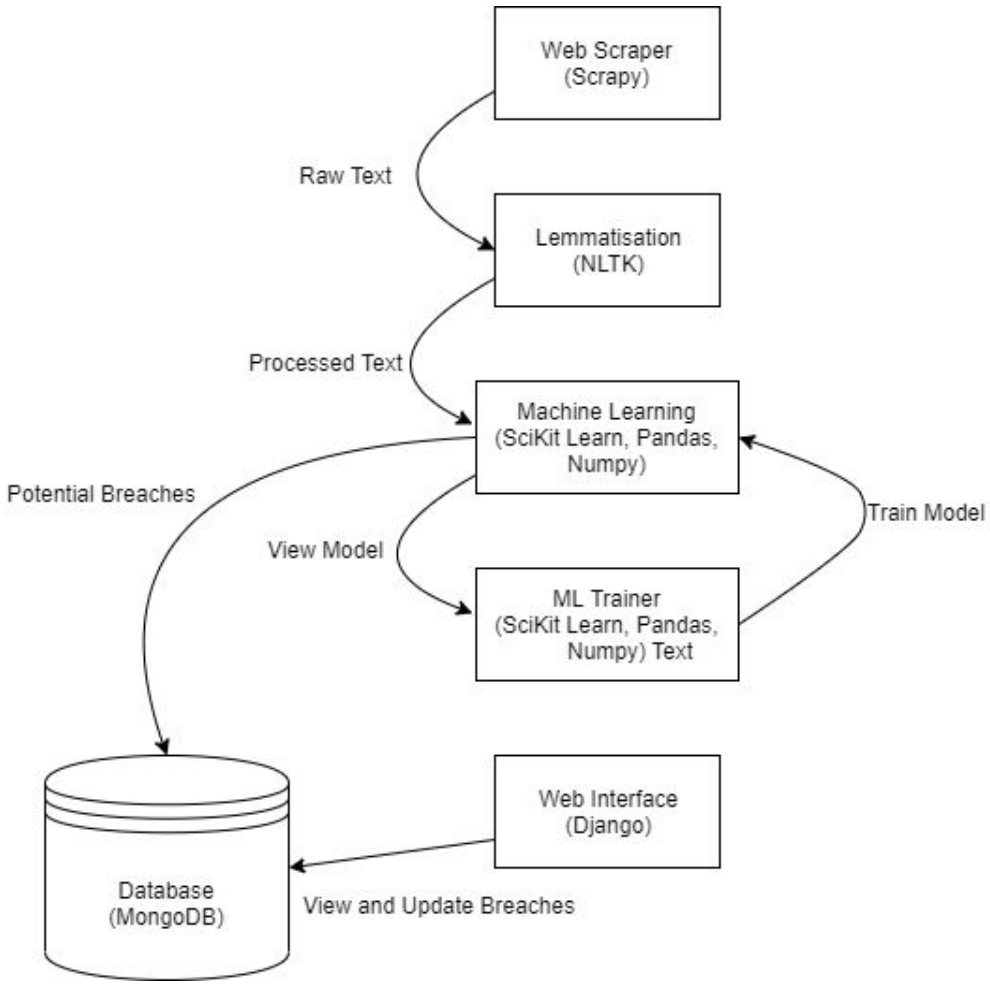
1.6.2 Limitations

- Breach reports are limited to English
- The project should take no more than 1 year
- Reduced crawling speed to avoid getting blacklisted
- Budget was $0 for project development restricting hardware and software resources for development

1.7 Relevant Standards

- PEP8 Style Guide Standard
    - Promotes readability and consistency in code.
- IEEE 829
    - Format for test documents.
- IEEE 1008
    - For creating comprehensive test cases.

1.8 High Level Block Diagram



2. Implementation Details

2.1 Functional Requirements

- System must scrape the internet
    - Data will only be pulled from the internet
    - Scraper should not be limited to a set list of sites
- System must save pages that contain data breach reports
    - Pages flagged as data breach reports will be saved into the database
    - Pages verified by an analyst will be marked as so in the database
- System must allow for training of the Machine Learning model
    - System comes with a model but allows training of a new one
- System must support displaying of aggregate data
    - Web interface will display data and allow the analyst to verify results
- Machine learning can evaluate a webpage and get feedback from supervisor
- Breach reports are stored in a database as a link to the website they came from
- Front end UI should display new breach report

2.2 Non-Functional Requirements

- Can be run on a Linux machine with Python3 and a network connection
- Can run without interruptions to crawl multiple websites without crashing
- Must crawl at minimum 30 pages per min

2.3 Modules

2.3.1 Web Scraper

For the web scraper, Scrapy was used. Scrapy is a library for Python which has a lot of web scraping features that were useful for this project. Scrapy is used to scrape the entire internet starting with a given url.
There is an issue with how Scrapy handles following links involving recursion causing it to crash due to memory consumption after extended periods of use. To mitigate this when running the scraper the user must put in a starting url for the crawler so that it is easy to restart and choose a new starting point for the crawler.
The web scraper works by getting a start url and pulling the html from that page. Then the body text from the html is sent to the lemmatizer for processing.

2.3.2 Lemmatizer

For our lemmatizer we used the library NLTK (Natural Language ToolKit) which has the english language corpus to recognize words and lemmatize them appropriately.
When a block of text is passed to the lemmatization function it first tags every word with its given part of speech which allows the lemmatizer to know how the word needs to be processed. After that it returns the same block of text with all the processed words. The processed text is sent to the vectorizer for further processing.

2.3.3 Vectorizer

The vectorizer was created with Sci-Kit Learn. A count vectorizer is used because it is the most memory efficient option available while also allowing the user to see the most commonly vectorized words.
There was also testing done to see if using N-grams would enhance the project however it didn’t help in performance. The reason to use N-grams would be for making sure context is retained in the reports found however it seems the model performed best just look at the word count.
The count vectorizer will take the lemmatized words as input and create vectors of words that the machine learning model understands.

2.3.4 Machine Learning

For the machine learning portion of the project SciKit Learn is used as well as some other libraries. The best model found is the Random Forest Classifier.
At this time the model is using 100 classification trees with 13,400 features. The best parameters were found using randomized grid search and cross validation (see testing section). Once the classification has been made the model sends the data to be stored in the database.

2.3.5 Database

MongoDB is used due to the flexibility that it allows. Since no relational data is stored, a standard SQL database was not appropriate. Currently the following fields are being stored:

- BaseURL
    - Domain that was scraped
    - Stored to look at statistics on a per domain basis
- FullURL
    - Entire URL string that the scraper was directed to
    - Used for manual review of reports
- ScrapyRedirectURL
    - URL that Scrapy got redirected to when scraping the page
    - Used for manual review of reports
- FullText
    - Unprocessed text that got scraped
    - Used for troubleshooting lemmatizer
    - Could be removed in order to reduce storage space
- LemmatizedText
    - Processed text that was lemmatized
    - Used for troubleshooting lemmatizer
    - Could be removed in order to reduce storage space
- Breach
    - Boolean for whether or not a link is a breach report
- Confirmed
    - Boolean for if a human has confirmed that the Breach value is correct

4. Testing Process and Results

4.1 Testing the Machine Learning Model

To test the machine learning model there was different techniques used such as cross validation and maximizing the Matthews Correlation Coefficient. By using cross validation it ensures that the model is not overfitting. Then using Matthews Correlation Coefficient we can get a more balanced accuracy scoring metric since our data is more weighted to non breach reports.

4.2 Testing the Model on Training Data

A large portion of testing done involved finding a satisfactory machine learning model. Because we used Random Grid Search, this process involved generating a model with random parameters and running it over a set of test data.

To quickly separate bad models from good models, we looked at the model's Matthew's correlation coefficient and its confusion matrix, and use those figures to get an idea about how many false positives and false negatives were being generated. We favored models that minimize the amount of false negatives, as it can be dangerous to miss a breach report. Minimizing false positives is a bonus, but not as important. Note all this testing was on data supplied by the client for testing. However, it appears this was a very good sample of data for training our model based on the results we found.

4.3 Results

The best model we found achieved a Matthews correlation coefficient of .95 on test data. This is very good because it means our model is performing well in classification even with the data being weighted with more non breach reports than breach reports.

4.4 Testing Promising Models on the Internet

Once we found a model that achieves good results for our set of test data, we connected it to the scraper and let it run on pages scraped from the internet. We then sort through the classified web pages by hand to determine whether the machine learning model works properly when fed random web pages.

4.5 Results

By using this method, we found a model that both performed well on test data and in the wild.

4.6 Scraper Testing

Testing the scraper involved making sure the scraper could pull necessary information from websites, as well as scrape without stopping. In order to test these things, we let the scraper run on its own until it crashed or failed to pull data from a page.

4.7 Results for Scraper Testing

We found that the scraper had trouble scraping data from dynamic web pages. In addition, we found that Scrapy's memory usage increased as it kept running, causing it to eventually crash after very long periods of continuous scraping. Because both of these issues involved problems that were caused by design faults in the libraries we were using, we decided that they were out of scope for this project.

4.8 Database Testing

The Database Interface was tested using some of the seed data supplied by the client. Each method was run to check for code coverage and that the database was storing information properly and accurately. Additionally relevant errors were handled by the interface. Lastly, the Add command was rigorously tested to see if the threading could handle hundreds of Add commands simultaneously.

4.9 Results for Database Testing

All tests passed successfully. The database will be fit to run this program with a high volume of requests coming in. It also successfully pulls and stores data quickly.

5. Concluding Material

5.1 Conclusion

In conclusion, it was found that it is possible to use machine learning to accurately and quickly find new breach reports. This method eliminates the need for human actors to spend time finding breach reports and enables them to use that time to act on existing breach reports. The libraries we used to build the scraper and machine learning agent are not suited to be used when implementing a permanent version of this tool. However, they were sufficient to prove the effectiveness of this method.

5.2: Recommendations for Future Work

For a future team picking up this project, we have two major recommendations. Firstly, the scraping engine should be changed out in favor of an engine that can run forever without continuously consuming more memory. This is necessary in order to have the program scrape the internet without having to stop and dump memory. Secondly, the machine learning model should be changed in favor of a model that supports supervised learning. It is recommended that you test these features early in order to make sure they function properly.

Aside from core changes, a useful feature that could be implemented is link prioritization. As it is, the scraper wastes a lot of time searching through links that have little to do with breach reports. A feature that recognizes links that are more likely to lead to breach reports and prioritizes them would save a lot of time. However, it is important that link prioritization doesn't prevent the crawler from finding breach reports in places they would be less likely to be found.

5.3: References

1. “PEP 8 -- Style Guide for Python Code.” *Python.org*, www.python.org/dev/peps/pep-0008/#introduction.
2. “829-1998 - IEEE Standard for Software Test Documentation.” *IEEE*, www.standards.ieee.org/standard/829-1998.html

“1008-1987 - IEEE Standard for Software Unit Testing.” *IEEE*, www.standards.ieee.org/standard/1008-1987.html

Appendix I - Operation Manual

Virtual Machine Setup

1. Import the OVA file by opening Oracle's VirtualBox then clicking file > import appliance.
2. Browse to the OVA file and hit import. Keep hitting next until the machine is imported to ensure that all the settings remain the same so the UI will be available on the host machine.
3. To start the virtual machine, hit start after selecting the newly imported VM.

Web Crawler Setup

1. Once in the virtual machine login to user *team1*. The password is *sddec19-01*, please change this if a more secure password is needed. (Note team1 user has root ability).
2. Once in team1 running the command *breachCrawler* followed by a starting url will start the crawler on that url. Note that due to a memory issue with scrapy the crawler will stop when it runs out of memory so there will be a need to restart the crawler as needed. Example usage: *breachCrawler https://krebsonsecurity.com/2019/10/avast-nordvpn-breaches-tied-to-phantom-user-accounts/*

After running breach crawler scrapy will start outputting to the console with information about websites it is crawling. When the crawler finds a breach report it will store it in the database. In order to see these breach reports in a user friendly manner, connect to the frontend.

Database Setup

The Database comes as part of the Virtual Machine. It is set to run on start under the mongod service daemon. If the database structure needs to be changed, simply just update the database interface. A feature of Mongo is that it will update the page layout on the fly when new pages are added. Additionally Mongo will add a new collection and database by simply changing the names and adding a new page.

Front End Setup

In order to connect to the front end, the following steps must be taken:

1. Find the IP address of the Virtual Machine
    a. Running *ifconfig* on the VM will list its IP address
    b. The VM is currently set to run in bridged mode, so it will have its own IP separate from the host machine
2. Start the Frontend Web Server
    a. On the VM terminal *cd* into the main project directory and navigate to the frontend folder
    b. Once inside the frontend folder, issue an *ls* command, you should see *manage.py*
    c. If you see *manage.py*, then you are in the right place
    d. Issue the command *python manage.py runserver*
    e. This will start the web server on port 8000 by default
3. Connect to the front end
    a. On a different machine open up a web browser
    b. Type in *http://IP:8000/* where *IP* is the VM's IP address
    c. This will connect you to the main page of the frontend

Appendix II - Initial Design Versions

Our initial design version involved a human-in-the-loop machine learning model training methodology. A human would have been able to feed in known breach and non-breach reports to the machine learning model after it was initially created, improving its learning capabilities. However, due to limitations in our machine learning library, Scikit Learn, we were unable to create this style of machine learning model. To mitigate this problem, we tried to create the best model we could on initial training.

Also, our initial design suggested the use of a hash vectorizer. However, after initial testing of the prototype with this type of vectorizer, we found it to be using an unsustainable amount of memory. To mitigate this problem we switched to a count vectorizer.

Appendix III - What We Learned

After completing this project, we learned the basics of machine learning, web crawling in Python, Mongo, and how to connect a front-end, back-end, and database. We also learned that finding a good machine learning model can take a very long time and a lot of processing power. Funny enough we found our model when accidently inputting the wrong parameters for loading the file. It ended up working great for us and is now the model being used for the project. Note that this project can be improved on since it was mainly created as a proof of concept for our client. There is most likely a better model out there that can be used which will allow for on the fly training.

Glossary

CSO
: Chief Security Officer

DNS
: Domain Name System

Lemmatize
: Process of shortening and processing words for a machine learning algorithm to easily categorize

ML
: Machine Learning

Model
: Statistical function creating a prediction output

NLTK
: Natural Language ToolKit library for Python

OVA
: Open Virtualization File Extension

Web Scraper
: A program that automatically harvests data from websites

Scrapy
: A Python library used for scraping the internet

UI
: User Interface

Vectorize
: Process of turning a string into a vector of floats so that a statistical model can understand them

Virtual Machine

Software that emulates a computer and runs on a host machine

VM

Virtual Machine