Crawling for Data Breach Reports

Project Plan

Team SDDEC19-01

Client: Benjamin Blakely (Argonne National Laboratory)
Advisor: Thomas Daniels

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

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

1: Frontal Materials

1.1: Table of Contents

1.2: List of Figures

1.3: List of Tables

1.4: List of Definitions

ML: Machine learning
CSO: Chief Security Officer
DNS: Domain Name System
UI: User Interface

2: Introductory Material

2.1: Problem Statement

Currently, there is no good way to be notified about every data breach that occurs, so important data breaches can easily fly under the radar. This can be dangerous for a company's security team, who needs to stay up to date with the latest data breaches in order to keep their company secure. In addition, it is important for these data breaches to be enumerated for security teams to reference later.

The purpose of this project is to serve as an early warning for CSO's on data 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. Then with this information the CSO will be informed on current security threats to their organization.

2.2: Use Case Diagram

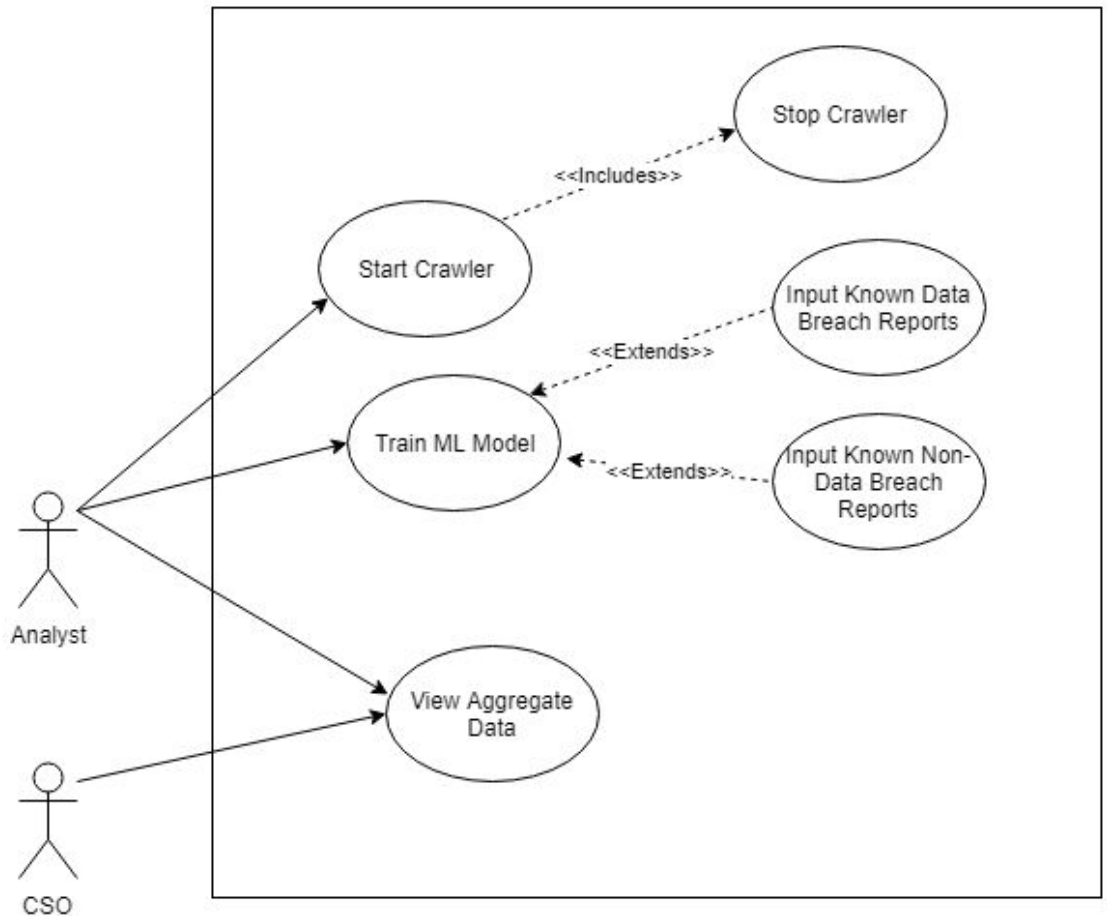

Figure 2.2.1 - Use Case Diagram

2.3: Operating Environment

At its core, our scraper is a python program, and can run on any machine with python 3 and an internet connection. 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 and training for the machine learning algorithm. Then when the final product is finished we will need an analyst to be on standby to train the model while its crawling after we do the initial training.

2.4: Intended Users and Uses

The first user we will discuss will be the CSO who generally will be very busy with day to day tasks and doesn't have time to spend sifting through data. So they will be able to see new

breach reports and learn about what potential security breaches could have an effect on their company and if they should raise flags about doing something. For the most part this will be an information gathering tool so the CSO in this case would need to act on the information given to them.

The second user is an Analyst who will be able to train our ML model and when it finds breach reports the analyst can tell whether or not they are relevant. Effectively the analyst will have a supervisor role to the crawling model. The analyst will eventually not be a necessary user once the model is fully trained as it will be able to accurately classify data breach reports and directly share the findings with the CSO.

2.5: Assumptions and Limitations

2.5.1: Assumptions

- The internet connection will not be interrupted.
- Power will not be interrupted.
- Will not get blacklisted from too many DNS requests. (We will attempt to limit requests to avoid this)
- Analyst training model will be fully competent in what a breach report is or is not.
- There won't be resource limitations for crawling - hardware, etc.

2.5.2: Limitations

- Dark web is off limits
- Budget is $0.
- Breach reports are limited to english.
- We have one year to complete the project.
- Our crawler will crawl at a speed of 32 pages per minute to avoid getting blacklisted.

2.6: Expected End Product and Other Deliverables

2.6.1: Core scraper

This deliverable is a python scraper that crawls the internet and uses machine learning to identify breach reports. This deliverable will also have an interface for an analyst to train the machine learning model, as well as an interface for a CSO to view and sort the found data breaches.

2.6.2: Natural Language Processing Module

This deliverable trims down and processes the raw text from the scraper module. This processed text will then be fed into the ML code.

2.6.3: Machine Learning Module

This deliverable contains all the ML code and feature detection code. This module determines what constitutes a data breach and what does not.

2.6.4: Python Front-End

This deliverable will allows the two main users of the system to access it. The CSO will be given a data breach report interface while the analyst will be given tools to train the ML module.

2.6.5: Breach Report Database

This deliverable is a MongoDB database that the scraper will use to store breach reports. The database is solely for the crawler's backend functionality, and will not be accessed directly by the users.

2.6.6: Documentation

For this deliverable there needs to be adequate documentation on our project. Python is a loosely defined language so documentation will be much more important.

2.7: Acknowledgement

We would like to acknowledge Benjamin Blakely of Argonne National Laboratory for his technical advice and for sponsoring this project.

3: Proposed Approach

3.1: High-Level Block Diagram of System

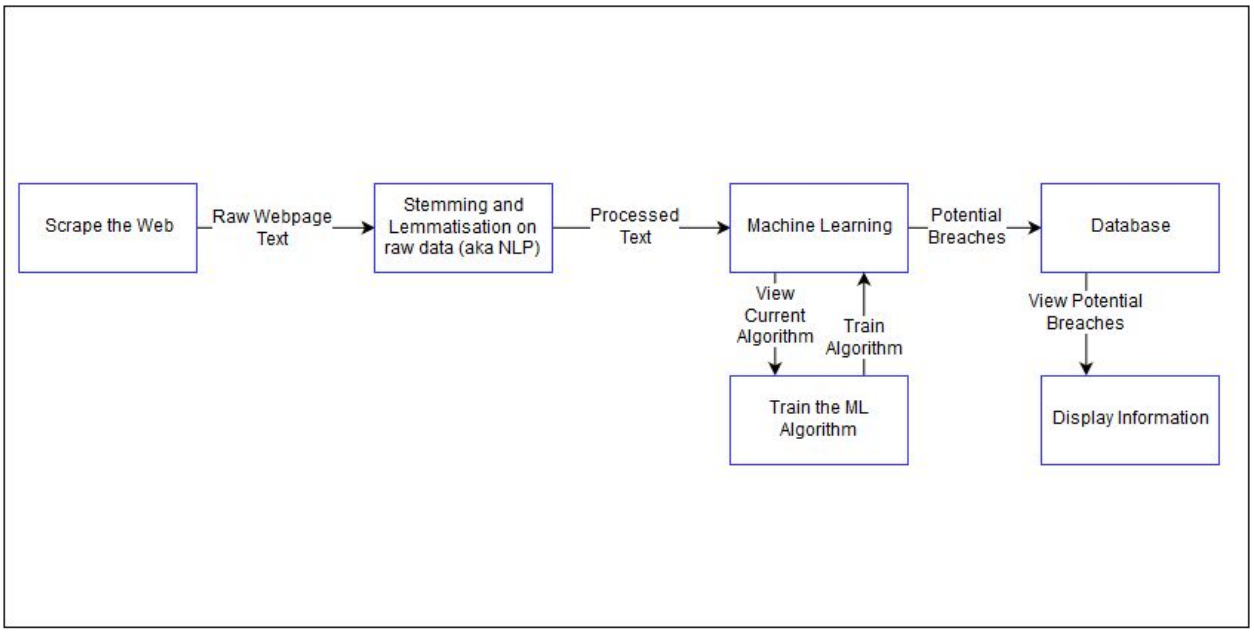

Figure 3.1.1 - High-Level Block Diagram of System

3.2: Functional Requirements

- **The system must scrape the internet.**
  The medium from which data breach reports will be pulled must be the Internet.

- **The system must save pages that contain data breach reports.**
  When the system finds a page with a breach report, it must detect that it is a breach report and save the page.

- **The system must allow for the training of the ML model.**
  A trainer user must be able to use a GUI to tell the ML agent what is and isn't a breach report in order to train it.

- **The system must support the displaying of the aggregate data.**
  The data collected and compiled about the scraped breach reports must be able to be viewed by a CSO in a GUI.

3.3: Constraints Considerations

- **The dark web is not a crawlable option.**
  The dark web can contain sensitive, illegal, and malicious data. For this reason, we will not be crawling it.

- **We need to schedule around the team's time constraints.**
  We are all busy students, so we need to find times where our schedules line up in order to work together.

- **The client can only meet every other week.**
  We need to plan our development schedule around Ben's schedule.

3.4: Technology Considerations

We considered both Java and Python as candidates for languages to build the crawler in. While Java is faster, Python already has plenty of web scraping libraries and machine learning libraries. For this reason, we chose to write our crawler in Python.

We will be using PyCharm as our IDE of choice, as it has the ability to set up a virtual environment inside the project and is free with our Iowa State email. As far as libraries go, we will be using Scrapy for scraping due to its simplicity and ease of use. We will be using MongoDB for our database due to its flexibility. We are currently looking into the following libraries to use in our machine learning model:

- Numpy for underlying ML math
- Goose3 for page extraction
- NLTK for Natural Language Processing
- Pandas for dataset manipulation
- Matplotlib for data plotting and visualization
- Scikit-learn, which offers more complex algorithms and is built from Numpy and SciPy
- Keras, an ML library that offers GPU support

3.5: Testing Requirements Considerations

We will need to allow the algorithm to run and find data for us and see whether what it finds is desirable or not. Also we will have to consult our known breach reports to determine whether the new ones we find are accurate. For this project run time isn't a consideration since reports happen in real time so there is no way of knowing how fast breach reports will pop up on the internet. It will also be an active crawler so theoretically there will be infinite run time. Another

test would be requiring that the database is being updated during the crawl. For the most part it will be module testing of the different components.

3.6: Safety and Security Considerations

- **The scraper should not scrape the dark web.**
  The dark web could inadvertently expose us and the machine to illegal or malicious material.
- **The scraper will not Denial of Service websites.**
  Sending too many requests too quickly could result in a remote server crashing or blocking our scraper.

3.7: Previous Work / Literature Review

As far as similar products go, there are plenty of services that host databases of breach reports. For example, Privacy Rights Clearinghouse hosts a breach report database that anyone can use to sort and view breach reports using their web interface (Data Breaches). The VERIS community database is a similar service directed at organizations, and relies on community members submitting breach reports to their database (elemind.com). Both of these services can be used to reliably find data breach reports that are older. However, they both rely on users to submit breach reports, so new, smaller breach reports have a chance to fly under their radar. Our scraper, on the other hand, has the advantage of being able to work 24/7 to find breach reports without requiring a salary, so new, smaller data breaches can be found faster.

3.8: Possible Risks and Risk Management

Possible Risks:

- The technologies we are using are new to us.
- Teammates may have to leave the project.
- Our project may require financial support for an unforeseen reason.

Risk Management:

- Utilize all available resources: advisers, APIs, internet, etc.
- Meet with/check-in with Ben often.
- Make sure team members are in close contact so that everyone knows as soon as possible if someone has to leave.
- Test during development cycle to make sure we are using new technologies correctly.
- Research how technologies interact with operating environment and each other.

3.9: Project Proposed Milestones and Evaluation Criteria

3.9.1: Research

Finalized libraries and technologies that we will use. Have a set list of seeding websites for the crawler as well as having example breach reports. Adequate understanding of Python for the team members and the libraries being used.

3.9.2: Basic web crawler

Crawler is able to pull and compile a list of links and potential breach reports to later be evaluated by the ML algorithm.

3.9.3: Data Vectorization

Create a lemmatization and vectorization pipeline. The lemmatization module will lemmatize data and remove extraneous information such as punctuation and HTML tags. The data vectorization module takes in the findings of the web crawler and outputs vectorized data(numbers) that will be fed into the ML algorithm.

3.9.4: Basic Machine Learning Model

Make a basic ML model that takes in the vectorized data and outputs a prediction on whether or not it is a data breach report.

3.9.5: Machine Learning Model

After prediction is made the model can take input on whether or not it was a good prediction. Effectively adding supervised learning functionality and a feedback loop.

3.9.6: UI

Create a UI for the analyst and CSO to view the detected data breaches. The UI should also allow the analyst to train the ML model.

3.9.7: Advanced Machine Learning Model

At this milestone the ML model should no longer need constant supervision, allowing it to independently learn and determine what is a breach report.

3.9.8: Testing

Test and tweak the ML model, ideally to >= 80% prediction accuracy. Ensure the ML model is not overfitting to the test data set.

3.10: Project Tracking Procedures

To track the progress of our project, we will meet weekly as a team and discuss what we have done and what we will do. This will ensure we are all working on different things, while also working towards the same goal, and allow for discussion on the status of the project. We will also run a Trello board to delegate/take on tasks. Also we will be using git so we can use the integrated git tracking for our project to check build health status and see what everyone is working on.

3.11: Statement of Work

3.11.1: Web Crawler

3.11.1.1: Objective

Create a web crawler that will look through the seeded websites we give it to collect and store breach reports. This will then process and store these breach reports and feed it to a ML algorithm which will classify each site as a breach report or not.

3.11.1.2: Approach

We will use the Scrapy library to collect reports from websites that we will initially give it. The crawler will be allowed to leave the initial seed sites and branch off after scraping the entirety of the seeds. The exception to this being the restrictions described in 3.6.

3.11.1.3: Result

A web-scraper that can efficiently collect breach reports from seed and connected websites. These potential breaches will be processed and quickly transferred to a ML model for classification.

3.11.2: Machine Learning Algorithm

3.11.2.1: Objective

Create a ML model that can discern between what is and is not a data breach report. When finding a breach report it will alert the security analysis who can forward the report to the CSO if they deem it a high priority breach.

3.11.2.2: Approach

Use Python libraries Pandas and Sklearn to develop a ML model and train it using known data breach and non-data breach reports. We first will lemmatize the breach report to process

unnecessary text and then vectorize it. Vectorizing it will turn the processed text into numbers that we can use to statistically classify each document using a ML algorithm.

3.11.2.3: Result

The ML model will be able to run unassisted, taking in input from the web crawler and selecting only the pages that contain breach reports.This model should be able to correctly classify breach reports at least 80% of the time with real time data.

3.11.3: Streamlined Crawler with Machine Learning

3.11.3.1: Objective

Have a product which can deliver breach reports with high fidelity to a CSO without having to be constantly supervised or otherwise managed during its life cycle. It will be able to actively scrape the web and find new breach reports to show to a CSO.

3.11.3.2: Approach

Integrate the two components of ML and the web crawler. Ensure that crashes and failures are handled gracefully by the program. The program will need minimal human assistance when running.

3.11.3.3: Result

Have a UI to display breach reports and possible security concerns. Allow the analyst to train the ML model during its training phase. Eventually have the crawler and ML model work together and evaluate the reports with minimal assistance allowing it to send results directly to CSO.

4: Estimated Resources

4.1: Estimated Resources

4.1.1: Personnel Effort Requirements

Task	Explanation	Estimate (Sprints are 2 weeks)
Research	Research Libraries and Technology to use	1 Sprint
Basic Web Crawler	Basic Crawler to crawl provided websites	2 Sprints
Data Vectorization	Stemming, Lemmatization, and Vectorization of data to be fed into the Model	1 Sprint
Basic Machine Learning Model	Basic model prediction	2 Sprints
Intermediate Machine Learning Model	Take input to determine if prediction was good or not - supervised learning	1 Sprint
User Interface	Interface to train model and view reports	1 Sprint
Advanced Machine Learning Model	No longer needs input to determine breaches	1 Sprint
Streamline of Model and Crawler	Helping to streamline the crawler and model together to no longer require links added to database	1 Sprint
Documentation	Document project and code	2 Sprints
Testing	Test components as they are developed and fused together	2 Sprints

Table 4.1.1.1 - Personnel Effort Requirements

4.1.2: Other Resource Requirements

The only resource we will need is a machine to run the crawler on. We anticipate using a rented machine from the university. A machine with a graphics card will be preferable for faster ML calculations.

4.1.3: Financial Requirements

We don't anticipate any financial requirements at this time, as all the technologies and computers we are using are free.

4.2: Project Timeline

Figure 4.2.1: Gantt Chart

Sprint Dates	2/10 - 2/24	2/25 - 3/10	3/11 - 3/24	3/25 - 4/7	4/8 - 4/21	4/22 - 5/5	SUMMER BREAK	8/26 - 9/8	9/9 - 9/22	9/23 - 10/6	10/7 - 10/20	10/21 - 11/3	11/3 - 11/17
Tasks													
Make Project Plan													
Do Research													
Build Web Crawler													
Design Basic ML Model													
Design Intermediate ML Model													
Design UI													
Design Advanced ML Model													
Streamline Model and Crawler													
Testing													
User Documentation													

5: Closure Materials

5.1: Closing Summary

With no current way to be notified about every data breach that occurs, many data breaches can easily be missed. This is dangerous and problematic for security teams who need to stay up to date in order to patch holes in their company's security networks.

By combining Machine Learning, Natural Language Processing, a Web Scraper, and a Database, users will be able to find data breach reports much faster. Additionally the detected breaches will be organized in a friendly to use interface to make identifying relevant breaches easier. This will allow the speed and depth of a machine to take over the tedious reading and categorizing of webpages.

With our solution in place, security teams will be able to identify many more potential data breaches that could affect their security networks. Additionally these teams will be notified faster than through traditional systems. With the resulting speed and depth of our solution, there should be far fewer instances of data breaches at companies using it.

5.2: References

Benjamin Blakely for machine learning algorithm and library recommendations.

"Data Breaches." *Privacy Rights Clearinghouse*, www.privacyrights.org/data-breaches.

Elemind.com. "VERIS The Vocabulary for Event Recording and Incident Sharing." *The VERIS Framework*, veriscommunity.net/index.html.