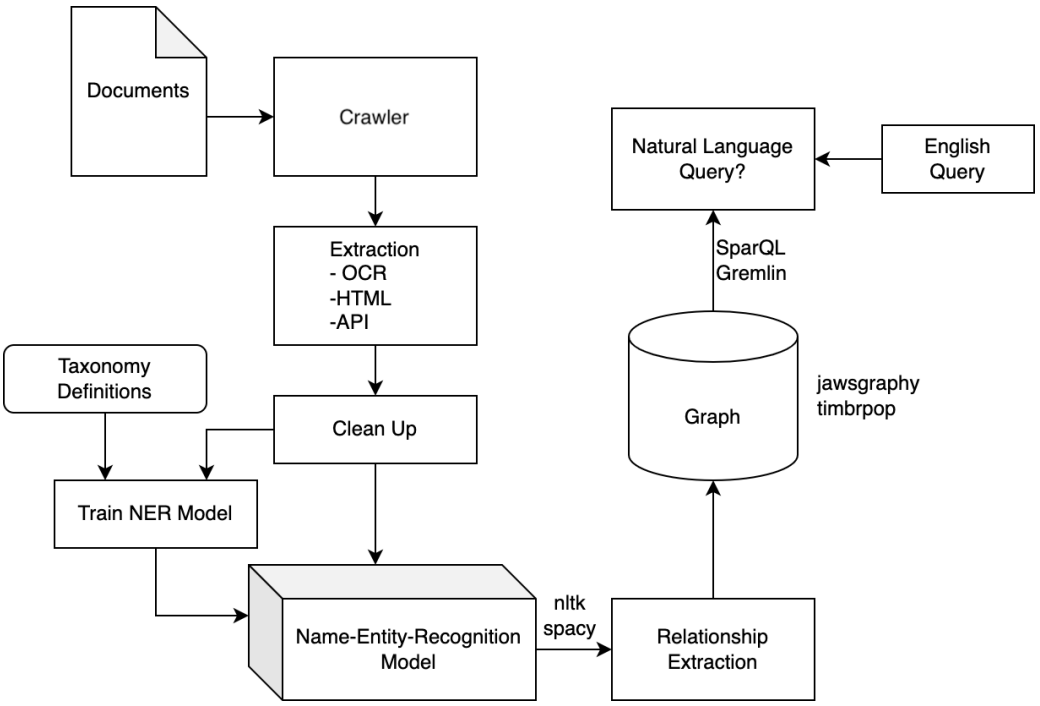
4.3 Proposed Design

4.3.1 Overview

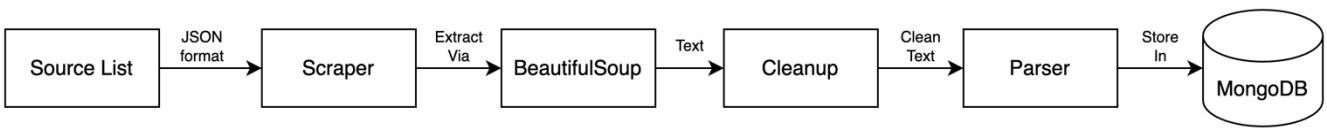
Our current design includes multiple software components, each with a roll to play to achieve our overall goal. One component is responsible for collecting data from external sources, another for extracting subjects and relationships between them from that data, and another for saving this information in a graph. This design works as a pipeline to start with raw text information and end up with a graph that can be queried for information. One last component is a frontend that users can use to perform the queries.



4.3.2 Detailed Design and Visual(s)

4.3.2.1 Scraper

The scraper is a Python module responsible for inputting a list of sources and outputting text files. The source list input should be in JSON format and be an array of objects, each with a "url" property that points to the URL of the main feed. Each output text file should be the text from an article not yet scraped from the inputted source list. Scraping should be done using the Python library Scrapy, and extraction of the scraped text should be done using the Python library BeautifulSoup. The next subcomponent of the scraper is the "Cleanup". The input will be the extracted text and the output will be text without irrelevant information. This will require little or much effort depending on the source and thus should be examined on a source-by-source basis. The last subcomponent of the scraper is a database to store information including the URL fetched, the article text, and a timestamp of when it was collected.



4.3.2.2 Parser

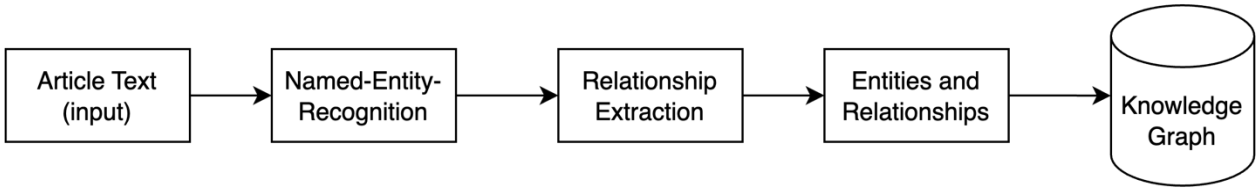
The parser is a Python module that takes input from the scraper (text files) and performs Named-Entity-Recognition (NER) and Relationship Extraction (RE). These two jobs allow for the creation of a graph structure with Named Entities as the vertices and relationships as the edges. The output of this component are entities and relationships for the knowledge graph.

The NER subcomponent will extract the following types of entities from text:

- Organization
- Vulnerability - CVEs or named/known exploits
- Threat Group - Known malicious groups
- Malware Type - Virus', Trojans, Ransomware, etc.
- Malware Name - Name of Malware
- System - OS's, Hardware, Software
- Port
- URL
- IP
- Protocol
- Filename
- Filetype
- Version - Patched versions or still vulnerable versions of SW

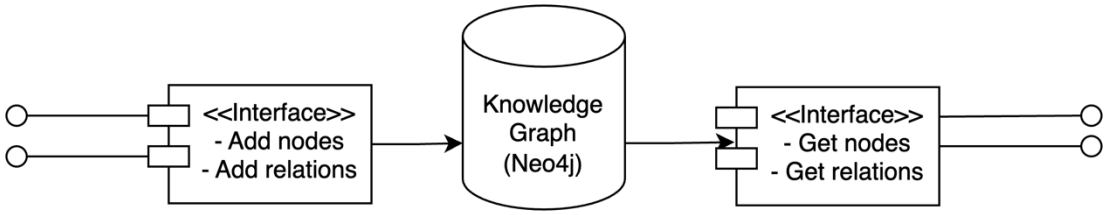
The RE subcomponent will extract the following relationships between entities:

- hasVulnerability
  - System, Protocol, Version -> Vulnerability
- usesVulnerability
  - Malware Type, Malware Name, Threat Group -> Vulnerability
- usedBy
  - System, Protocol, Filetype, Filename -> Organization, Threat Group, System
  - Port -> Protocol, System
  - Vulnerability, URL, IP -> Threat Group, Malware Name
- attacked
  - Threat Group -> Organization
- attackedBy
  - Organization -> Threat Group
- isType
  - Malware Name -> Malware Type



4.3.2.3 Knowledge Graph

The Knowledge Graph component will accept entities and relationships as input and store these into a graph database. Our current choice for the database is Neo4j as it offers good performance, no cost, and is graph based. The entities will be stored as vertices in the graph (called Nodes in Neo4j) and relationships between entities will be stored as edges in the graph (called "relationships" in Neo4j). This should be running in a Docker container but have an exposed API that the Parser is able to use to insert these entities and relationships. It should also expose an API to perform queries on the graph, such as getting all nodes related to a node by some relationship.



4.3.2.4 Frontend

The frontend of our project will be a React web application written in TypeScript. The web app will accept user input specifying what information should be queried and will perform an API request on the Knowledge Graph. An example could be a cybersecurity researching searching for "Samsung" and getting back all or a limited set of nodes connected to it, such as recent attacks launched against them. It should then display the result in a graph or tree format.

4.3.3 Functionality

Our design is intended to operate by the user visiting a web application. The user will enter in a query, such as a company, vulnerability, and/or timeframe they are searching for, and the Knowledge Graph will be queried with those parameters. The user will be able to navigate around the Knowledge Graph by dragging the mouse in different directions to explore the connections between nodes. The user may also choose to have their results displayed in a Tree View rather than a Graph View.

If the user enters in a query for which there are no results, they will be prompted that no results exist. In the case of an error in the query, the user will be prompted that their query was completed unsuccessfully along with any additional error information from the server.

Periodically the pipeline will run again, scraping new articles, parsing them, and inserted the extracted information into the knowledge graph. The interval is currently defined as 1 hour, although this is subject to change.

4.3.4 Areas of Concern and Development

The current design completely satisfies the client requirements and moderately meets the expected needs of our users. The area of most concern will be the development of the Named-Entity-Recognition model and performing Relation Extraction due mostly to many unknowns we have yet to encounter. Performing NER and RE is on track to be the most complex portion of the project. The immediate plan for developing the solution to this component is beginning testing whether we need to train our own models to perform these steps or if we may take advantage of existing technologies such as CyNER. If we are able to use a previously-trained model for the NER step, this would drastically decrease the level of effort needed for a

successful Parser component. We currently have no questions, as our client and faculty adviser has graciously provided us with scientific papers going into details about different attempts at Cybersecurity Knowledge Graphs and some Information Extraction on a text corpus.

4.4 Technology Considerations

Scraping – scrapy

For scraping we chose the scrapy library. It is one of the most popular Python web-crawling frameworks. It features a wide variety of inbuilt tools to help us collect the data we need. Ultimately, we chose it because of the rich community and how widespread the framework is.

One of the weaknesses of scrapy is how basic its parsing tools are. Scrapy is great for collecting data from websites, but not the best at actually parsing that data. A solution, which we are implementing is to use another framework to parse the data. We chose BeautifulSoup4 due to its rich feature set, some team member's prior knowledge and its widespread usage.

NER – spacy

We chose spacy as our NER library, due to the good support available and that some team members and our advisor have some background in this framework. There exist other options such as the NLTK, and cloud offerings. We did not want to use a proprietary framework like the Google Cloud Natual Language API to keep our project open source and free.

NLTK is more academic focused that spacy. It is meant to be a toolbox of machine learning tools for academic use, while spacy is more oriented towards developers. Spacy has a richer set of tools to help us accomplish our goal faster and more efficiently. NLTK focuses just on strings, while Spacy focuses on objects. Most of our data is objects, so spacy was the better fit.

Database – SQL (column based) vs Neo4j (graph based)

The decision to use a graph-based database was a very easy decision. We are building a graph, so a graph-based database fits our data type. Graph-based databases use more memory and are harder to do text queries on, but store trees and graph data more efficiently than a column-based database.

Furthermore, a graph-based database allows storing arbitrary data. A column-base database requires data to be in a very rigid format – data must fit into the specified columns. A lot of our data is wildly different, with types such as vulnerabilities, software packages and malware. This variance makes storing data in columns difficult and unscalable.

Web technologies & NLP stretch goal

Our project will make use of TypeScript and the React library. We came to this decision over other web development frameworks because of a large ecosystem of packages, previous experience of team members, and the easy-of-use of performing API calls and displaying the data in an aesthetic way. Other considerations were made such as Angular, Vue, or simply HTML/CSS/JS. Ultimately because of the previously described reasons, React won as the library of choice.

4.5 Design Analysis

Scraper:

- Implemented basic scraping and parsing functionality, scraper reads input of sources in a JSON document and outputs the parsed HTML document for each source.
- Integrated the BeautifulSoup4 parses in with the Scrapy spider. Now instead of raw HTML data, it is being cleaned up to a more readable format which will make storing in a database or running language processing on the data much easier.

Docker Container and GitHub:

- Constructed new folder setup in GitHub that allows each folder to be a Docker container.
- Used Docker Compose to organize and build each container.
- The scraper and MongoDB database that was constructed to store article information will each be a separate container.

Article Tagging:

- Began using the NER Annotator tool available online in order to create a model which can be used by the spacy tool in order to process the article information.

So far during the implementation of the items above, the proposed design has functioned well in organizing and linking the different components together. The scraper and the parser are currently being worked on however the other sections of the proposed design have not been started yet so an analysis of how they are working is not possible. For future design, the article tagging will continue after more information is scraped and stored in the database. Then the created model from NER tagging will be used to parse the article data and identify information to build the knowledge graph.