Knowledge Graphs for Cybersecurity Reasoning

DESIGN DOCUMENT

Team 1
Benjamin Blakely - Client
Benjamin Blakely - Adviser
Brandon Richards - Frontend Development Lead
Micah Gwin - Python/ML Development
Alice Cheatum - Programmer
Nicklas Cahill - Tester/Programmer
Michael Watkins - Python/ML Development
Carter Kitelinger - Client Interaction

sdmay23-01@iastate.edu
https://sdmay23-01.sd.ece.iastate.edu/
Revised: 12/02/2022

Executive Summary

Development Standards & Practices Used

An essential development practice for our project is the proper use of version control such as Git in addition to testing standards. Everyone on the team uses feature branches and pull requests when writing code to ensure a clean main branch. These pull requests are to be reviewed by at least one other team member for feedback. In addition, pull requests can be easily reverted if there is an issue. Testing standards include minimum coverage of 75% for each module and 100% of tests should pass before code is committed. Specific specifications are also considered in this project, including:

- PEP8
- NIST
- CVE
- CVSS

Summary of Requirements

- The product should use no less than 3 sources for cybersecurity information.
- The product should crawl sources for new information at least once a day.
- The product should extract information using HTML parsing, OCR, and/or API.
- The product should perform Information Extraction by using a trained model.
- The product should produce a knowledge graph containing cybersecurity entities and relations.
- The product should persist this knowledge graph using the graph database Neo4j.
- Extensible by future developers by using interfaces and writing modular code.
- Code should have documentation on function definitions and up-to-date README.
- The crawl speed should be rated as to not disturb the resources of the sources.
- Only open-source libraries should be used in the product.
- Low-medium power GPU cluster for NER model training
- Responses to queries should support being displayed in graph or tree form.
- The user should be able to build queries using date, company, and vulnerability filters.
- Query results should be returned in less than 30 seconds.

Applicable Courses from Iowa State University Curriculum

- COMS 311

- CprE 310
- ENG 314
- COMS 309
- COMS 319
- CYBE 230
- CYBE 231

New Skills/Knowledge acquired that was not taught in courses

- Python
- Machine Learning
- Web Scraping
- Docker
- MongoDB
- TypeScript
- Neo4j
- Python Libraries: BeautifulSoup, scrapy, coverage, unittest, spaCy

Table of Contents

1 Team

1.1 Team Members

- Brandon Richards
- Micah Gwin
- Alice Cheatum
- Nicklas Cahill
- Michael Watkins
- Carter Kitelinger

1.2 Required Skill Sets for Your Project

- Python Programming Experience
- React
- Cyber Security Knowledge
- TypeScript
- Machine Learning Knowledge
- Docker
- MongoDB

1.3 Skill Sets covered by the Team

- Python Programming Experience - All
- React – Brandon, Micah, Michael
- Cyber Security Knowledge – Alice, Nicklas, Carter, Michael
- TypeScript – Brandon, Michael
- Machine Learning Knowledge - Michael
- Docker – Brandon, Michael
- MongoDB - Michael

1.4 Project Management Style Adopted by the team

Our team has chosen agile as a project management style because our project goal of developing a knowledge graph will require many iterations. New knowledge and metrics that we acquire during sprints will help us better understand and plan for the next set of tasks we need to perform in the next sprint.

1.5 Initial Project Management Roles

- Brandon Richards — Frontend Development Lead
- Micah Gwin — Python/ML Development
- Alice Cheatum — Programmer
- Nicklas Cahill — Tester/Programmer
- Michael Watkins — Python/ML Development
- Carter Kitelinger — Client Interaction

2 Introduction

2.1 Problem Statement

Cybersecurity threat reporting is currently spread out across multiple sources and written in a non-standardized format. Information is updated frequently, changing the landscape and requiring much effort to parse and read for relevant information. Cybersecurity researchers, Incident Responders, and System Administrators need to be able to efficiently query information about a specific software, malware, threat, etc., as well as new and emerging ones. Generating a Cybersecurity Knowledge Graph (CSKG) that contains relevant datapoints will allow for efficient information storage and querying capability.

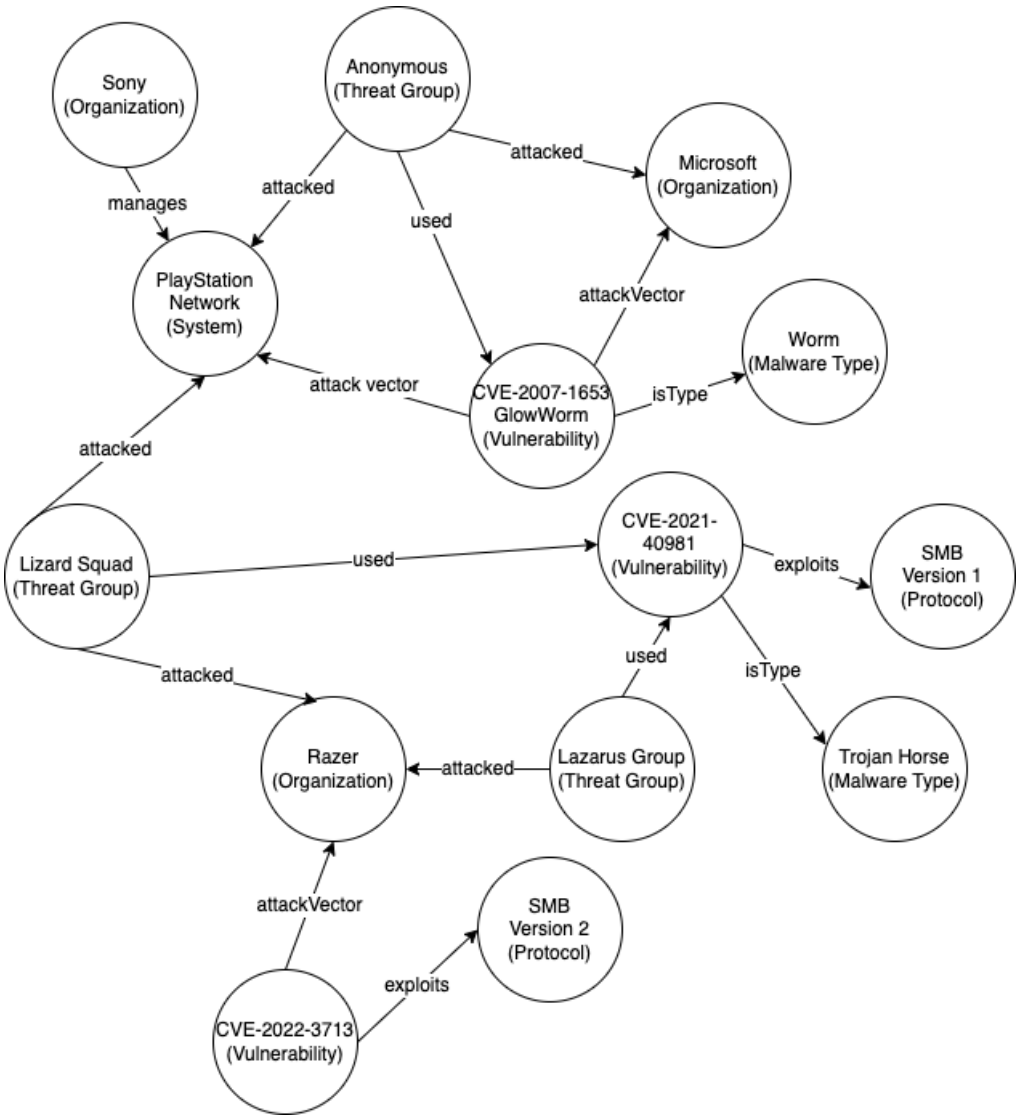


Figure: Example Cybersecurity Knowledge Graph

2.2 Intended Users and Uses

2.2.1 Cybersecurity Researcher

2.2.1.1 Persona

Demographics

- Post-grad students

Hobbies and interests

- Cybersecurity
- Data enthusiasts
- Tech savvy

Motivations (Who do they want to be? What do they want to do? How do they want to feel?)

- They want to efficiently find up-to-date information about new or specific cybersecurity threats.
- They want their computer systems to feel secure.

Personality and emotions

- Paranoia?
- Intelligent
- Flexible

Values (What is important to their identity?)

- Anonymity
- Privacy
- Informed

2.2.1.2 Empathy Map

Who? **Cybersecurity Researcher**

What / need to do?

- Be knowledgeful of relevant current threats.
- Understand context and implication of threats

See?

- News articles / research papers of new threats
- Attacks against enterprise and personal computer systems.

Say?

- I wish there was a quicker and easier way to find this stuff!

Hear?

- Other researchers talking about cybersecurity.
- Queries about how a cybersecurity threat affects a specific entity (company, university, software, etc.)

Do?

- Look at scattered reporting of cybersecurity threat.
- Parsing for relevance.

Think?

- I hate having to parse through many publications to find relevant information
- I hate having to maintain a list of reliable sources

Feel?

- Determined
- Curious
- Frustrated
- Annoyed
- Overwhelmed

Need Statement:

A Cybersecurity Researcher needs a way to parse relevant information quickly and efficiently because the landscape changes rapidly and sources are spread out and contain irrelevant details.

Benefit:

Researchers would see a reduction in time spent searching for new and related information, leading to better context and comprehension.

2.2.2 Incident Responder

2.2.2.1 Persona

Demographics

- College Grad
- Various Certificates

Hobbies and interests

- Penetration testing
- Keeping networks secure
- Cybersecurity

Motivations (Who do they want to be? What do they want to do? How do they want to feel?)

- Protecting business operations
- Protecting client information

Personality and emotions

- Investigative
- Curious
- Defensive
- Eye for small details

Values (What is important to their identity?)

- Intelligence
- Competence
- Integrity

2.2.2.2 EMPATHY MAP

Who? **Incident Responder**

What / need to do?

- Respond to network intrusions, access policy violations, cybersecurity threats
- Defend systems owned by their employers from attacks in the future

See?

- Current threats or intrusions to the company/business they are working for

Say?

- "I wish I knew of a quick and easy way to find information about this new vulnerability!"

Hear?

- Is our infrastructure safe?
- We've had a network intrusion; you need to fix this.

Do?

- Investigate and patch exploited systems

Think?

- Which software or hardware flaw is responsible for this intrusion?
- Who attacked us?

Feel?

- Attacked
- Defensive
- Rushed
- Panicked

Need Statement:

An Incident Responder needs a way to find vulnerabilities quickly because investigating and patching cybersecurity threats requires up-to-date information on a time crunch.

Benefit:

Quicker information gathering and analysis results in a faster response time to threats and stronger defenses in place for next time.

2.2.3 SYS ADMIN

2.2.3.1 PERSONA

Demographics

- At least Highschool Grad
- Certificates

Hobbies and interests

- Software or hardware systems
- Networks
- Servers
- Maybe a mild interest in cybersecurity

Motivations (Who do they want to be? What do they want to do? How do they want to feel?)

- Maintain the systems they are responsible for, focusing on uptime and usability.

Personality and emotions

- Meticulous
- Overworked
- Problem solver

Values (What is important to their identity?)

- Efficiency
- Network/server uptime
- Accessibility
- Supporting end-users

2.2.3.2 EMPATHY MAP

Who? **Sys Admin**

What / need to do?

- Keep the systems they are responsible for secure
- Know which threats are most relevant
- Balance security with usability of the systems

See?

- News articles about new vulnerabilities, exploits, and attacks

Say?

- Why isn't this working?
- What new vulnerabilities are there for the software we run?

Hear?

- Why isn't this working?
- Wasn't this supposed to be secure?
- I thought you maintained this?

Do?

- Maintain hardware and software on many systems

Think?

- I dislike having to keep up with the constant cybersecurity knowledge while still needing to maintain systems

Feel?

- Overwhelmed
- Unfamiliar

Need Statement:

A Sys Admin needs a way to learn about current cybersecurity threats without in-depth knowledge because their systems need to be secure but they also have other things to focus and work on.

Benefit:

Can save time understanding cybersecurity problems, allowing for communication with others, and making more time for administrating systems.

2.3 Requirements & Constraints

Functional Requirements

- The product should use no less than 3 sources for cybersecurity information.
- The product should crawl sources for new information at least once a day.
- The product should extract information using HTML parsing, OCR, and/or API.
- The product should perform Information Extraction by using a trained model.

- The product should produce a knowledge graph containing cybersecurity entities and relations.
- The product should persist this knowledge graph using the graph database Neo4j.

Non-Functional Requirements

- Extensible by future developers by using interfaces and writing modular code.
- Code should have documentation on function definitions and up-to-date README.
- The crawl speed should be rated as to not disturb the resources of the sources.
- Only open-source libraries should be used in the product.

Resource Requirements

- Low-medium power GPU cluster for NER model training

Aesthetic Requirements

- Responses to queries should support being displayed in graph or tree form.

User Experiential Requirements

- The user should be able to build queries using date, company, and vulnerability filters.
- Query results should be returned in less than 30 seconds.

2.4 Engineering Standards

- PEP8
    - PEP 8 is the official style guide for Python code. Our team's code will follow this style guide to ensure maximum readability and avoid developer issues due to different coding styles in the same project.
- NIST
    - NIST has many definitions on various cybersecurity topics that we will utilize in our project. Our project will provide quick and easy information that has ties to NIST Cybersecurity Framework (CSF) and NIST Risk Management Framework (RMF), thus these standards will be incorporated into the end product.
- CVE
    - Common Vulnerabilities and Exposures is a standard for computer security flaws our project will need to follow to obtain, process, and serve our cybersecurity information. Using CVE will keep vulnerabilities tied to their initial reports, which include a description of the flaw, making it easier for users to follow a chain of information from our knowledge graph.
- CVSS
    - The Common Vulnerability Scoring System will be used in our project due to our interaction with vulnerabilities. This framework has three metrics that are used to rate the severity of vulnerabilities our knowledge graph will contain. We will interpret and display data in a CVSS format to remain consistent with other sources of information and improve comprehension by our users.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team has chosen agile as a project management style because our project goal of developing a knowledge graph will require many iterations. New knowledge and metrics that we acquire during sprints will help us better understand and plan for the next set of tasks we need to perform in the next sprint.

Our team is going to use Git as a VCS and GitHub to host our repository. We will also be using JIRA as a ticketing system to better track tasks, subtasks, sprints, backlog items, and responsibility for each task.

3.2 Task Decomposition

1. Determine list of sources to obtain news articles and blog posts.
    a. Inspect sources for legitimacy and reputation
    b. Record URLs of main page and/or specific pages of interest (e.g., Malware blog)
2. Develop scraper to obtain news articles, blog posts, etc., using Scrapy
    a. Select a programming language
    b. Select libraries to perform downloads and parsing of HTML
    c. Create file specification for storing list of sources
    d. Write code for scraper
    e. Write testing code for scraper
    f. Output artifacts for later use by NER model
    g. Containerize with Docker
3. Develop one or more methods to clean up articles (may vary depending on type of article)
    a. Research existing methods for cleaning up irrelevant information
    b. (Potentially) Manually annotate relevant vs. irrelevant data in documents
    c. Verify on test cases that relevant information isn't being destroyed
4. Extract relevant entities (vulns, companies, software, exploits, etc.) and the relationships between them
    a. Research existing annotation techniques and cybersecurity-specific NER models
    b. Determine if we need to train custom NER model
    c. (Potentially) Train NER model:
        i. Manually annotate set of documents from selected sources
        ii. Perform supervised machine learning to train NER model
    d. Generate entities and relationships from cleaned-up source information
5. Use extracted entities and relationships to generate knowledge graph
    a. Collect output from Information Extraction
    b. Insert into graph database
6. Run pipeline created in steps 2-5 periodically and continuously on new articles
    a. Create job to run at interval

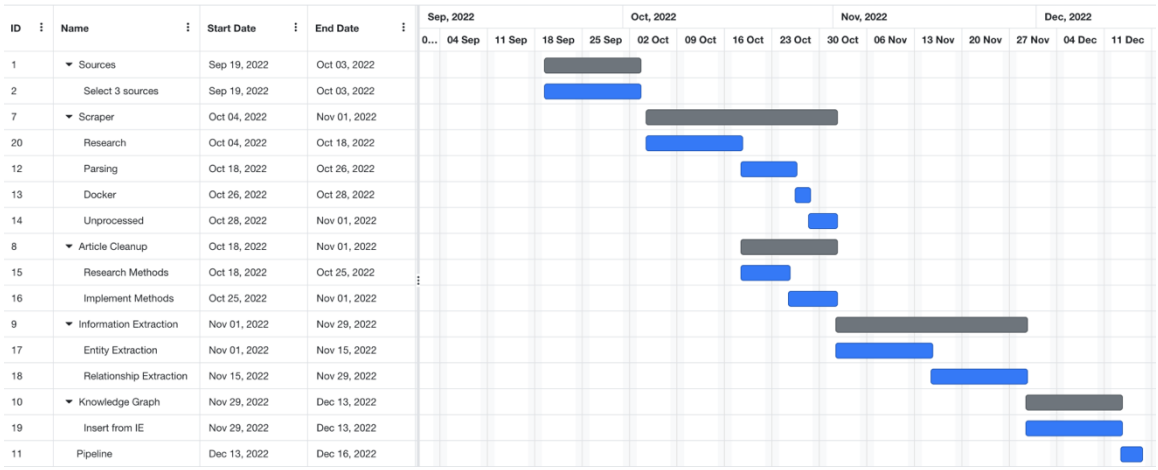
    b. Determine if new articles of interest have been posted
    c. Run new articles through pipeline
7. Develop a web interface to run queries on the graph
    a. Design
        i. Develop prototype
        ii. Receive feedback from client
    b. Develop
        i. Select framework to make website
        ii. Create API to query knowledge graph
        iii. Implement designs from prototype
        iv. Implement filters to query the graph
        v. (Stretch goal) Use natural language to query the graph

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

1. Sources
    1.1. 3 or more sources have been selected to scrape for information
2. Scraper
    2.1. Scraper can download and parse input sources.
    2.2. Runs inside Docker container.
    2.3. Identifies more recent unprocessed articles with 100% accuracy.
3. Article cleanup
    3.1. Articles achieve removing unnecessary information with 25% accuracy.
    3.2. Articles achieve removing unnecessary information with 50% accuracy.
4. Extract entities and relationships
    4.1. Software can identify subjects (companies, operating systems, vulnerability, etc.) with 75% accuracy.
    4.2. Software can identify relationships (vulnerability works on this OS and application run by this company) with 50% accuracy.
5. Generate knowledge graph
    5.1. Contains more than 15 entities including companies, operating systems, applications, malware, etc.).
    5.2. Nodes have properly labeled edges with 75% accuracy.
6. Pipeline periodic and continuous running
    6.1. The job runs on the specific interval 100% of the time.
7. Web Interface
    7.1. Prototype delivered to client with 80% satisfaction (satisfaction to be quantified with rating survey).
    7.2. API created to query 100% of knowledge graph entities and relationships.
    7.3. Filtering by Company, Application, OS, Vulnerability, or Malware can be performed.
    7.4. (Stretch) Natural Language query can serve intended results 50% of the time.

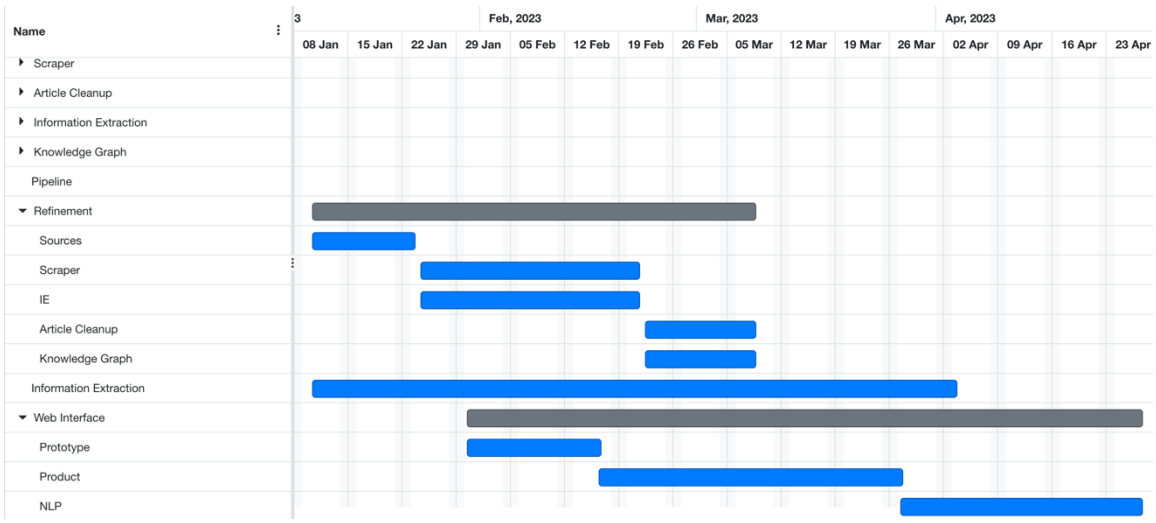
3.4 Project Timeline/Schedule

Semester 1 Schedule



ID	Name	Start Date	End Date
1	Sources	Sep 19, 2022	Oct 03, 2022
2	Select 3 sources	Sep 19, 2022	Oct 03, 2022
7	Scraper	Oct 04, 2022	Nov 01, 2022
20	Research	Oct 04, 2022	Oct 18, 2022
12	Parsing	Oct 18, 2022	Oct 26, 2022
13	Docker	Oct 26, 2022	Oct 28, 2022
14	Unprocessed	Oct 28, 2022	Nov 01, 2022
8	Article Cleanup	Oct 18, 2022	Nov 01, 2022
15	Research Methods	Oct 18, 2022	Oct 25, 2022
16	Implement Methods	Oct 25, 2022	Nov 01, 2022
9	Information Extraction	Nov 01, 2022	Nov 29, 2022
17	Entity Extraction	Nov 01, 2022	Nov 15, 2022
18	Relationship Extraction	Nov 15, 2022	Nov 29, 2022
10	Knowledge Graph	Nov 29, 2022	Dec 13, 2022
19	Insert from IE	Nov 29, 2022	Dec 13, 2022
11	Pipeline	Dec 13, 2022	Dec 16, 2022

Semester 2 Schedule



3.5 Risks And Risk Management/Mitigation

Description	Likelihood	Consequences	Risk	Mitigation
Training uses too many resources	Unlikely	Major	High	Optimize code, set resource usage limits,

				allocate more GPUs for cluster
Model is trained incorrectly	Moderate	Moderate	High	Start early, involve Michael in most decisions due to experience, Research
Source is too difficult to perform extraction	Moderate	Moderate	High	Pre-scout source's format, plan extraction logic ahead
Resources too high to display query	Moderate	Moderate	High	Limit number of displayed responses
Sources go down or block our traffic because of too much scraping too fast	Likely	Major	Extreme	Strict rate limits

3.6 Personnel Effort Requirements

Tasks This Semester	Sources (1)	Scraper Design (2)	Clean Up Articles (3)	Extract Entities and Relations (4)	Generate Knowledge Graph (5)	Total for Semester 1
Estimated Hours Per Person	2 hours	15 hours – Program Team	10 Hours – CybSec Team	20 - 30 hours – Both Teams (Varies Greatly)	8 hours – Both Teams	68-78 Hours

- Program Team Consisting of: Brandon Richards, Michael Watkins, and Micah Gwin.
- CybSec Team Consisting of: Alice Cheatum, Nicklas Cahill, and Carter Kitelinger.

Based on the table above, we broke the hours down into what we thought we could finish in the first semester. Using the tasks from the Task Decomposition section, we decided it would be best if we split into two teams: the Program Team, handling most of the Python programming, and the CybSec Team, handling most of the non-programming work that involves Cyber Security knowledge. As provided in the table above, each team was given an estimated hours per person.

This also gives each team a different part to work on during sprints, since things can be worked on in parallel.

3.7 Other Resource Requirements

When training the machine learning model on the annotated articles, we may require a GPU cluster to get a quicker and more efficient way of training that a normal computer would not be able to give us.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our project is primary focused on two communities: security researchers in academia and information security roles in industry. Although these communities will be directly affected by our project, the entire world will be indirectly affected as well. The increased efficiency of gathering cybersecurity knowledge will allow our target communities to better do their job and, increasing safety, security, privacy, and integrity in all software applications. As societies around the world grow more dependent on technology, our goal is to make the software they interact with safer.

Area	**Considerations**
Public health, safety, and welfare	All users of a technology companies' products are indirectly affected by our project due to the utility we provide information security staff. Helping these roles increases the public safety and welfare in their interaction with technology. It could also harm job opportunities of these positions, as an effective product would require less staff to research and fix problems. In the same way, researchers' use of our product will also improve the general populations' interaction with technology utilizing their discoveries.
Global, cultural, and social	Our project accurately reflects the values of our target cultural groups including security researchers in academia and information security staff in industry. Using extracted information to build a knowledge graph that can be queried is in line with practices to streamline cybersecurity information gathering.
Environmental	The environmental effects of this project are indirect. The software will be executed and hosted on servers that use a lot of electricity of unknown origin (renewable vs. nonrenewable).

	There is a potential impact of training ML models with GPU clusters in terms of energy usage, although our project would have to scale magnitudes larger for this to become a reasonable concern.
Economic	This project being successful will have an impact of increased productivity by information security roles in industry by getting them access to recent, condensed, and relevant information quicker. One pending consideration is if the project proves extremely useful what obligation we have to make it available to good actors in terms of cost.

4.1.2 Prior Work/Solutions

There have been research papers on the idea of using Knowledge Graphs to store information in the Cyber Security domain. One example is "TINKER: A framework for Open source Cyberthreat Intelligence". This research paper delves into creating a knowledge graph that is used primarily to "infer threat information from the [cybersecurity] text corpus". This differs from our project in that it attempts to strictly capture malware information from CTIs (Cyber Threat Intelligence). Our project is aimed more for researchers and industry professionals to be able to query a knowledge graph of many entity-types (companies, vulnerably, malware) that is consistently updating with new information automatically.

Some of the pros and cons of our target solution would be:

1. Pro: Web Interface with query input and data visualization
2. Pro: Updating periodically (e.g., every hour) with new cybersecurity information
3. Con: Information sourced from reputable cybersecurity blogs instead of CTIs.

Our project is not following previous work of any Senior Design project.

4.1.3 Technical Complexity

The design includes several components stored in Docker containers that collaborate to scrape website articles and construct the knowledge graph. These components include:

1. Web scraper and parser: uses frameworks to make a request for pages in the source list and get the raw HTML. Then uses programming organization principles to find and extract the useful information from the document. This requires structural knowledge of web pages and analysis of how to clean the pages into useable data for the following step.
2. Taxonomy definition and training Name-Entity Recognition Model: using the NLTK and spacy libraries with Python along with our taxonomy definitions to perform natural language processing (NPL) and queries on the data. This requires scientific and mathematical knowledge of language processing and article tagging to create relationships for the graph.

3. Relationship extraction to construct the graph and store in a database: Using the relationship data created above, use computer science principles to build a knowledge graph that has meaningful relationships between the nodes and is in a readable format.
4. Running natural language queries on the completed knowledge graph: requires computer science and mathematical principles regarding AI and language comprehension that can query the graph with a search term and return meaningful results.
5. Displaying knowledge graph and queries on frontend for consumption: Includes web programming principles and standards to host a page in the browser that can render the graph and provide a search feature for the natural language queries. Will need to efficiently display the results and scale based on the graph or query size.

4.2 Design Exploration

4.2.1 Design Decisions

1. Potential Sources: The biggest decision in terms of potential sources we ran into were whether to include blogs as a source in our project as many cyber security professionals don't have or never needed to obtain a college degree so the information, they put out isn't in a well-structured or well-defined manner. Excluding these sources could be a problem later in the project because these cyber security blogs are rich with information.
2. Cleanup: The cleanup of the text from sources is for training our Name-Entity Recognition Model which is a crucial part of our project. This decision is important because we need a well-formatted text to train the NER. Avoiding the best decision for our cleanup process would be detrimental to training the NER.
3. Taxonomy Definitions: Taxonomy is important in determining the scope of the project and which topics we should cover in the knowledge graph. We plan on utilizing the broader topics such as Vulnerabilities, Malware, and breaches but the decision comes down to including some of the smaller topics into our taxonomy like threat actors and phishing.

4.2.2 Ideation

For cleaning up documents to remove irrelevant information after they've been scraped and parsed, we considered several different options using the lotus blossom method for ideation.

- Guess and Check – Repeatedly attempt to clean up the document until the output meets certain criteria or is within certain parameters.
- Manual Annotation – Clean up a small set of documents by hand and then use that sample data to train a machine learning model to clean up documents.
- Find sources with no irrelevant data – Get information from sources that are known to only contain relevant information, or are in a structured format (e.g., JSON, XML, etc.) which can be queried for relevant information in a standard way.
- Do all cleanup by hand – Manually clean up all new articles every time one is added for as long as the project is running. Not a realistic solution.
- Define parameters for an algorithm – Figure out how to clean up each source and write a program to do it automatically. Program may need to be rewritten when sources change website design.
- Don't do it – Output all data, even if it is irrelevant. The easiest "solution" to this problem, but may make later steps in the process more difficult since we have to work around all of the noise.

4.2.3 Decision-Making and Trade-Off

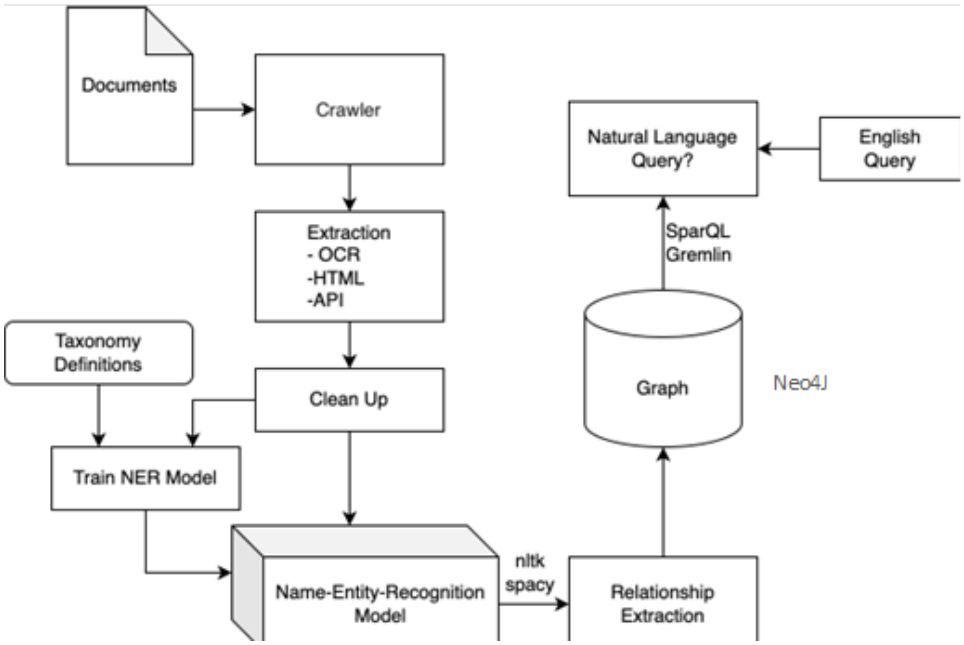
Cleanup Methods		Accuracy		Development Time		Development Complexity		Run Time	
Weights	Total	3		2		2		1	
Guess and Check	21	3	*9*	2	*4*	2	*4*	4	*4*
Find perfectly clean sources	26	4	*12*	4	*8*	1	*2*	4	*4*
By-Hand Cleanup	22	3	*9*	1	*2*	4	*8*	3	*3*
Manual parameters	23	3	*9*	3	*6*	2	*4*	4	*4*
Don't Clean up	22	1	*3*	4	*8*	4	*8*	3	*3*
Research Existing Techniques	26	4	*12*	2	*4*	3	*6*	4	*4*
Manual Annotation	25	3	*9*	2	*4*	4	*8*	4	*4*

For deciding which data cleanup method to use, we made the table above. The different methods added to a weighted decision matrix. We chose four main factors: accuracy, development time, development complexity and run time. Accuracy represents how accurate we predict the method to be. A lower number indicates that the method might have a high chance of improperly cleaning the data, leaving mistakes and invalid data. Accuracy was chosen as the most important factor, as good data is important for building machine-learning algorithms. Next, was development time, which represents how long it will take to implement the cleanup method. A lower number indicates that it will take a longer time to implement. Next is development complexity, representing how difficult the method is to develop or use. Lastly, is run time, which represents how long the cleanup method will run when cleaning up data.

4.3 Proposed Design

4.3.1 Overview

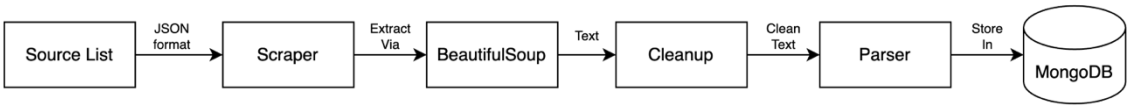
Our current design includes multiple software components, each with a roll to play to achieve our overall goal. One component is responsible for collecting data from external sources, another for extracting subjects and relationships between them from that data, and another for saving this information 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.

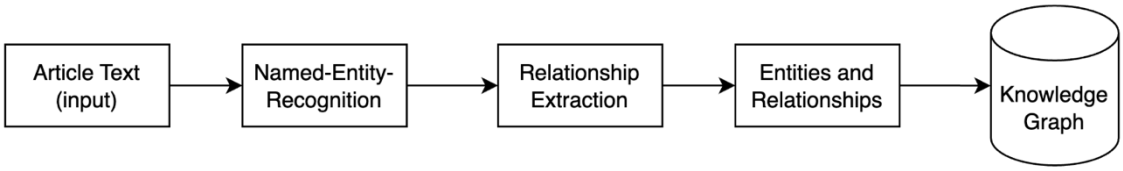
The NER subcomponent will extract the following types of entities from text:

- Organization

- Vulnerability - CVEs or named/known exploits
- Threat Group - Known malicious groups
- Malware Type - Virus', Trojans, Ransomware, etc.
- System - OS's, Hardware, Software
- Protocol, with the version if applicable

The RE subcomponent will extract the following relationships between entities:

- manages
  - Organization -> System
- attackVector
  - Vulnerability -> System
- used
  - System, Protocol, Filetype, Filename -> Organization, Threat Group, System
  - Port -> Protocol, System
  - Vulnerability, URL, IP -> Threat Group, Malware Name
- attacked
  - Threat Group -> Organization
- exploits
  - Vulnerability -> Protocol
- isType
  - Vulnerability -> Malware Type



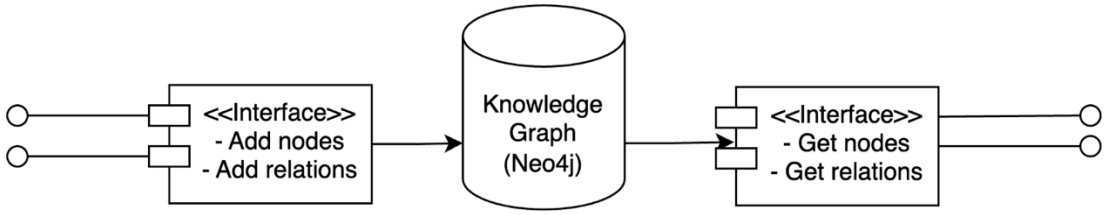
The parser currently has a couple of defined interfaces. They are as follows:

- AbstractInput
  - Description: Reads input from implementation-dependent source (MongoDB, file, hardcoded, etc.)
  - Methods:
    - get_inputs() -> list[str]
- AbstractParser
  - Description: Performs IE (NER + RE) on input and outputs entities and relationships
  - Methods:
    - parse(input: str) -> ParserResult
- AbstractOutput
  - Description: Outputs entities and relationships in an implementation-dependent way (file, database, etc.)
  - Methods:
    - output(result: ParserResult) -> bool
- ParserResult
  - entities: list[str]

    - relationships: list[ParserRelationship]
- ParserRelationship
    - from: str
    - to: str
    - type: str

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.

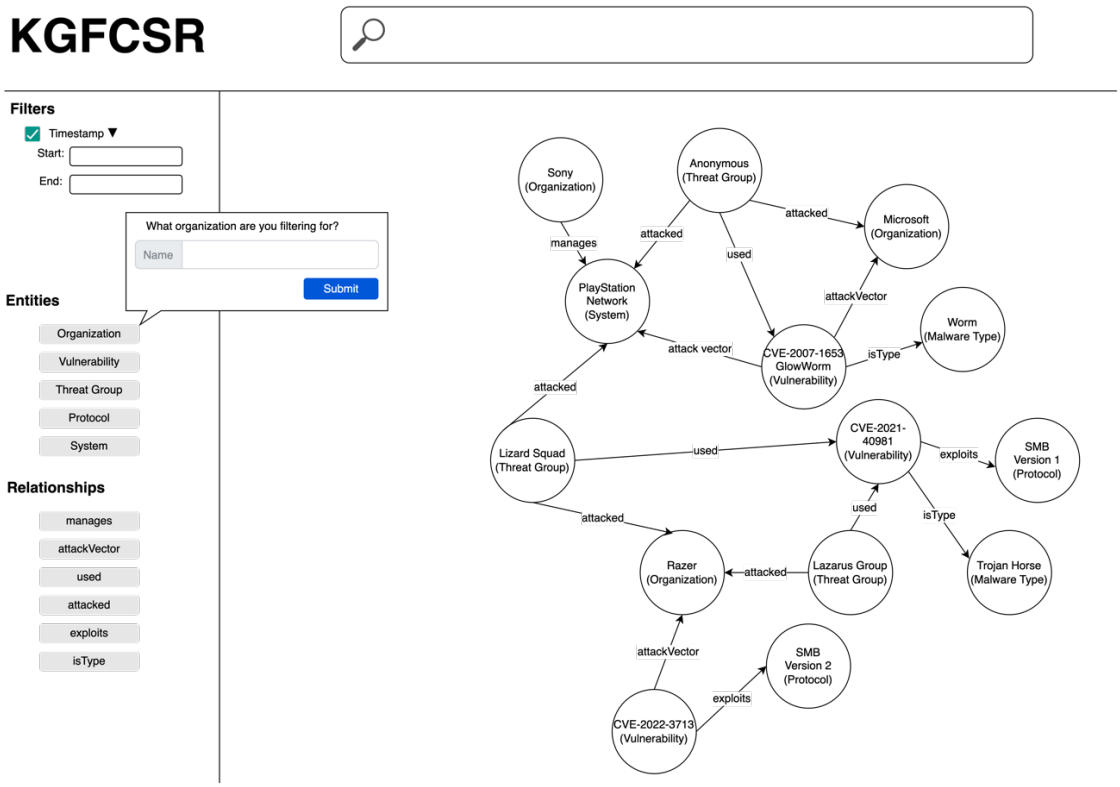


Figure: Frontend Mockup

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 can 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 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

Spacy will be a great library to assist with NER and/or RE 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 Natural 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 to create a model which can be used by the spacy tool 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

5 Testing

Our project will make use of many types of testing to test the various software components and subcomponents. For each of our components, our goal is to have at least 75% code coverage and 100% of tests pass. Each component is responsible for containing unit, integration, and system tests that verify its behavior internally, while additional integration and system tests will be outside of these components to test their interaction.

When changes are made in one component, it will be required to run all tests in that component in addition to all integration and system tests. New tests should be added to verify the behavior of newly added logic. Our main instruments for testing will be the Python testing framework unittest for our components written in Python and the testing framework jest for our website component written in TypeScript. This will work to run unit tests on individual components and integration and system tests across components.

The team's overall testing philosophy is to test early, test often, and use it as a tool. Testing can be a wonderful tool in a Test-Driven Development (TDD) framework because expected inputs and

outputs can be specified first, then the tests' results indicate to the developer whether they've implemented the desired functionality.

5.1 Unit Testing

Scraper

The scraper tests and code coverage should be run after every commit to ensure no breaking changes have been made. The testing plan is as follows:

1. Open terminal
2. Change directory to `scraper`
3. Run `make coverage` command
4. Verify no failures in tests
    a. If all tests pass, the text "OK" will show at the bottom of the output.
    b. If one or more tests fail, the bottom of the output will read "FAILED" with the number of failures in parenthesis (e.g., "FAILED (failures=1)")
5. Open generated coverage file `htmlcov/index.html`
6. Ensure test coverage is greater than or equal to 75%.

Parser

The parser tests and code coverage should be run after every commit to ensure no breaking changes have been made. The testing plan is as follows:

1. Open terminal
2. Change directory to `parser`
3. Run `make coverage` command
4. Verify no failures in tests
    a. If all tests pass, the text "OK" will show at the bottom of the output.
    b. If one or more tests fail, the bottom of the output will read "FAILED" with the number of failures in parenthesis (e.g., "FAILED (failures=1)")
5. Open generated coverage file `htmlcov/index.html`
6. Ensure test coverage is greater than or equal to 75%.

Website

The tests and code coverage for the website component should be run after every commit to ensure no breaking changes have been made. The testing plan is as follows:

1. Open terminal
2. Change directory to `frontend`
3. Run `yarn test` command
4. Verify no test failures
5. Open generated coverage file `htmlcov/index.html`
6. Ensure test coverage is greater than or equal to 75%.

5.2 Interface Testing

Our interfaces include the source list for the scraper, the entity-relationship output from the parser, and the graph API queried by the frontend to be rendered. To test these interfaces, we have tests in place that send data through the interface and verify its integrity. In addition, the parser

entity-relationship output can be tested by inputting mock article data and ensuring the parser output is as expected. For most of our existing and future tests we use the Python testing library such as unit test or mocking tools in a test class that asserts the scraper and parser are functioning properly. The graph API is provided by Neo4j and thus isn't tested in our project, although any bindings that we write for this API will be.

5.3 Integration Testing

The critical integration paths in our project begin with the human-developed source list being handled correctly by the scraper. The next is the scraper properly storing article information in the MongoDB database. Then after this is completed, the parser retrieves the MongoDB data and constructs the list of entities and relationships. Lastly this entity-relationship data is stored in the Neo4J graph database and then queried by the frontend for rendering. All the above paths are essential to the project operating correctly and can be tested by using Docker Compose to build and run all the containers on a local or cloud server, then use a script or library to verify the container are communicating properly with data expected by that test case.

5.4 System Testing

By using source lists with inputs for which the expected outputs are known we can test that the output of our system falls within our defined parameters. Our first step to test output we will use *portions* of documents we annotated manually to ensure the machine learning program is outputting what we expect it to and compare it to the accuracy requirements of the annotation. As the machine learning portion gets fed more documents and learns how to annotate with more accuracy, we will begin to give it larger portions of a document until it can annotate a full document within the accuracy percentage requirement we defined. Once this point has been reached, we will begin to give it full documents until the machine learning has been refined enough to where it is no longer required for us to test each output for accuracy. Our system testing will use all the tools mentioned in the unit, interface, and integration test sections to test the whole system, as each section needs to perform to our standards for the whole program to perform well. If one piece is not working up to standards, the output may contain unexpected behavior.

5.5 Regression Testing

Our strategy to implement Regression Testing is supported by two rules: Have a separate, clean, working copy of the code on our main branch and run unit, integration, and system tests on every pull request to this branch. The pull request is not able to be merged until all tests are passing. This will ensure existing functionality has not been broken, plus having the main branch be a clean working copy allows us to rollback changes that break existing functionality. Critical features that our project needs to ensure do not break are the scraper fetching sources and the parser processing these sources and outputting expected entities and relationships.

5.6 Acceptance Testing

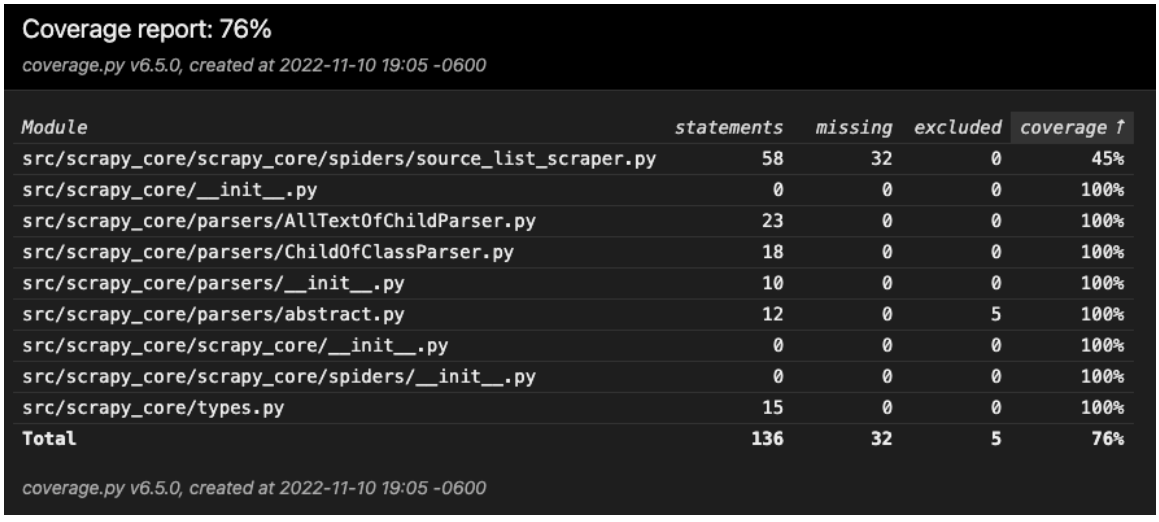
Acceptance testing of our functional requirements will be performed by running unit, integration, and system tests to provide verifiable evidence that our test cases are passing. Most of our project's non-functional requirements can be summed up as a satisfactory level of code quality, libraries being open-source and a license allowing free usage, and documentation of code. These non-functional requirements can be verified by reviewers of pull requests. For example, if a new library is added as a dependency, the reviewer should be double checking the license allows us to use it. Our client will be involved in acceptance testing by being made available the number of passed tests and code coverage, ensuring we've met the expectations set in the requirements and metrics/evaluation criteria. They will also be delivered our iterative beta builds after each sprint to evaluate our progress and correctness in our design implementation.

5.7 Security Testing

Security Testing will consist of testing that malicious queries from the frontend website are not able to perform any sort of remote code execution (RCE) on the backend database. To ensure this doesn't happen, all inputs from the user need to be sanitized. Testing plans for security testing include performing various types of penetration testing on the frontend, including NOSQL injection and cross-site scripting, by performing queries with malformed input to execute malicious code on the server or in the user's browser.

5.8 Results

The unit testing on the scraper had two testing requirements associated with it. All tests (100%) had to pass, and the test coverage of scraper Python source code had to be at least 75%. Our unit testing successfully achieved those goals; all unit tests passed, and the coverage is 76%. This was able to show that the scraper is working exactly as intended taking input from the source list and outputting the article data. The following figure shows an output of test coverage in our current implementation of the scraper. Note the total coverage is shown at the top.



Coverage report: 76%

coverage.py v6.5.0, created at 2022-11-10 19:05 -0600

Module	*statements*	*missing*	*excluded*	*coverage ↑*
src/scrapy_core/scrapy_core/spiders/source_list_scraper.py	58	32	0	45%
src/scrapy_core/__init__.py	0	0	0	100%
src/scrapy_core/parsers/AllTextOfChildParser.py	23	0	0	100%
src/scrapy_core/parsers/ChildOfClassParser.py	18	0	0	100%
src/scrapy_core/parsers/__init__.py	10	0	0	100%
src/scrapy_core/parsers/abstract.py	12	0	5	100%
src/scrapy_core/scrapy_core/__init__.py	0	0	0	100%
src/scrapy_core/scrapy_core/spiders/__init__.py	0	0	0	100%
src/scrapy_core/types.py	15	0	0	100%
Total	**136**	**32**	**5**	**76%**

coverage.py v6.5.0, created at 2022-11-10 19:05 -0600

The scraper testing serves as a proof of concept for each component containing unit, integration, and system tests. Future work includes creating tests for the other components, and then integration and system tests between components. All the testing described in this section will ensure that our implementation matches both our chosen design and verify our requirements are being met.

6 Implementation

During the Fall 2022 semester, our team accomplished implementing the scraper component of our system. The scraper accepts a source list as input (including the URL's and parsers that should be used to extract the article text), creates the spider necessary to perform the web crawling, and then stores the article along with other metadata in a MongoDB database.

Our preliminary implementation plan for next semester is to begin development of the parser by initially creating multiple implementations of our parser interface with different NER models. We will test these different libraries and see if there are any functionalities to gain from them or if we need to build our own model. We will know relatively quickly whether we need to train our own model, in which case we will use self-annotated articles and machine learning to generate it. The graph database and frontend of the project will be developed in parallel with the parser. For the frontend, a prototype in Figma will be developed first, and then the final product will be created.

7 Professional Responsibility

This discussion is with respect to the paper titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment", *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

Area of responsibility	**Definition**	**NSPE Canon**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.
Sustainability	Protect environment and natural resources locally and globally.	
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.

7.1 AREAS OF RESPONSIBILITY

Area of responsibility	Definition	NSPE Canon	IEEE
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	Maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	Seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, to be honest and realistic in stating claims or estimates based on available data.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	Hold paramount the safety, health, and welfare of the public. Disclose promptly factors that might endanger the public or the environment;
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	Avoid injuring others, their property, reputation, or employment by false or malicious actions, rumors or any other verbal or physical abuses;
Sustainability	Protect environment and natural resources locally and globally.		Strive to comply with ethical design and sustainable

			development practices.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	Improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies.

Difference between NSPE Canon and IEEE definitions:

Work Competence: Work to remain knowledgeable and grow/learn. Do not perform tasks unless confident in qualifications or communication with superior about honest level of skill. The NSPE and IEEE differ when the IEEE states to maintain and improve technical competence, while NSPE only makes mention of doing work you're qualified to do.

Financial Responsibility: NSPE and IEEE definitions differ in that IEEE does not address Financial Responsibility.

Communication Honesty: Accept and contribute meaningful and honest criticism of work performed, including potential errors and pitfalls. NSPE and IEEE differ in that IEEE addresses engineers honestly address each other about potential errors in technical work.

Health, Safety, Well-Being: When made aware of issues that could cause harm to health, safety, or well-being of the public, vocalize these issues to the proper authority. NSPE and IEEE use the exact same first sentence. One difference is IEEE mentions specifically mentions what holding the safety, health, and welfare paramount looks like.

Property Ownership: Do not use misinformation or lies to negatively impact others, both physically and verbally. NSPE and IEEE differ in that IEEE explicitly states to avoid damaging property and respecting its ownership.

Sustainability: In contrast to NSPE, IEEE has an entry for the sustainability professional responsibility. In a nutshell, it says to perform technical work in a way that isn't harmful to the environment and can be performed sustainably.

Social Responsibility: NSPE and IEEE differ in that NSPE is more focused on how engineers behave to not shine a negative light on the profession while IEEE is more focused on keeping the public informed on the implications of their work.

7.2 Project Specific Professional Responsibility Areas

Work Competence: This applies to our professional context because it is mainly a software project, which requires maintenance and modification to be performance, potentially much later and not by us. Our well-informed decisions in the building of these components will result in an

intuitive and easy-to-debug design. The team's work competence has been satisfactory thus far. We have not had issues writing code, reasoning about our design, or annotating articles to train a NER model on.

Financial Responsibility: This professional responsibility area applies to this project because hosting software can become expensive if not done carefully. This project consists of a database and webserver, and when choosing to host it on something like AWS, a cost will incur. If not careful about how many resources are being used on an Infrastructure as a Service product, it can quickly go above hundreds of dollars. This team's level of performance in terms of financial responsibility is not yet applicable. We have not begun hosting our project on servers, thus so far, our product has cost nothing to build in terms of money. We have only used readily available, free software and libraries.

Communication Honesty: This professional responsibility area applies heavily for multiple reasons. This product hinges on our development team having open communication about techniques, progress, and errors we encounter. Without proper communication, we will be unable to solve such complex problems like Named-Entity-Recognition or Relationship Extraction. This team's level of performance with honest communication at a high level. This team has had disagreements that we solve with reason, evidence, and group votes.

Health, Safety, Well-Being: This professional knowledge area is highly important to our project because the product we are developing could potentially be used by the attacker of a system rather than a defender. This misuse of our project has implications for the health, safety, and well-being of the public due to the prominence of technology in society. During our technical work we must consider the impact each decision may have on the public. This team's level of performance in professional responsibility of health, safety, and well-being has been lacking. We have put very little thought into ensuring only trusted individuals may use our service. One idea was to vet candidates of use using their credentials, thus disallowing many attackers from utilizing the service. More thought needs to be put into this area due to its importance.

Property Ownership: This professional knowledge area applies to our project because it uses websites hosted by other people and we have the potential to harm their servers if not careful. If we do not set a rate limit or large enough interval between fetching their website, we could accidentally perform a DoS (Denial of Service) attack on their website. We are taking special precautions to not damage their property in the process of obtaining their data. The team's level of performance with respect to property ownership has been at a medium level. Although we have not yet implemented rate limits for the scraper, we have been respectful of the source's servers and not scrape articles from them too often. We are planning on setting an interval of one hour and a rate limit of a couple seconds to not interrupt their typical traffic.

Sustainability: This area is hardly relevant to our project because we are performing operations typical of a software project of this type. Our impact in terms of using servers from data centers is less than that of small companies and feels very responsible. Our development practices are sustainable due to the small dataset we are working with. The team's level of performance in the professional responsibility of sustainability is not yet applicable. This is because we have not begun hosting our product on a server or used GPUs to train neural networks. We have just begun working on the software and testing it on our local machines.

Social Responsibility: This professional responsibility is not as applicable to this project due to the target audience. We have been less concerned about improving understanding for uneducated

individuals and society, but more about being a tool specifically for researchers and individuals already in the cyber security field. Our performance in the professional area of social responsibility has been low. In terms of the IEEE standard, we have made no effort to attempt to explain our product to a general audience. In terms of the NSPE, we have conducted ourselves respectfully when in communication with our advisor/client.

7.3 Most Applicable Professional Responsibility Area

The most applicable professional responsibility area for this project is Work Competence. We owe it to the users of the finished product in addition to future developers tasked with maintaining this software to put thought and care into our design and implementation decisions. A majority of our requirements also hinge on our team performing with a high level of work competence.

8 Closing Material

8.1 Discussion

So far, implementation performed by our project has been successful. Our scraper addresses requirements such as using 3 or more sources for cybersecurity information and article extraction using HTML parsing. The requirement of crawling the sources at least once a day has not been implemented yet due to infrastructure being needed to run this job. The scraper is currently a piece of software that is run once locally to grab the information. We've also met our requirements of having code that is written in a modular form with documentation and an up-to-date README file. The planning of the parser has been more challenging than initially expected. There are a couple of already existing Named-Entity Recognition models available trained on cybersecurity domain language, however this does not address the extraction of relationships and doesn't exactly match the entities we want to track.

8.2 Conclusion

So far, the scraper component of system has been implemented, including test running and generating code coverage of tests. All libraries being used in the various components have been selected, and inputs and outputs of each component have been defined. The goal of this project is to build a knowledge graph containing cybersecurity-specific terminology to allow querying of useful information. In our context, "useful" information can be defined as relationships between attackers, organizations, software, vulnerabilities, and timeframes. Another goal is to parse text from cybersecurity news articles to build this knowledge graph. The main constraint from achieving these goals is performing Named-Entity recognition and Relationship Extraction. These are complex tasks that need to be performed on a variety of sources with different writing styles.

8.3 References

P. Evangelatos, C. Iliou, T. Mavropoulos, K. Apostolou, T. Tsikrika, S. Vrochidis, and I. Kompatsiaris, "Named entity recognition in cyber threat intelligence using transformer-

based models," *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2021.

P. Ranade, A. Piplai, A. Joshi, and T. Finin, "Cybert: Contextualized embeddings for the cybersecurity domain," *2021 IEEE International Conference on Big Data (Big Data)*, 2021.

N. Rastogi, S. Dutta, A. Gittens, and M. Zaki, "TINKER: A framework for Open source Cyberthreat Intelligence," 21st International Conference on Trust, Security and Privacy in Computing and Communications, Oct. 2022.

8.4 APPENDICES

8.4.1 Team Contract

Team Members:

1) Carter Kitelinger
2) Brandon Richards
3) Alice Cheatum
4) Micah Gwin
5) Nicklas Cahill
6) Michael Watkins

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
    a. Friday 1-2pm at Parks Library or SIC in person.
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a. Our preferred method of communication is Discord.
3. Decision-making policy (e.g., consensus, majority vote):
    a. Decisions will be made by a ⅔'s majority, i.e. 4/6 team members will have to agree.
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. Minutes will be recorded in Google Drive. Each meeting will be a document and they can be archived into an "Archive" folder with the date as the document title. Team members will take turns recording minutes throughout the semester.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Team members will show up for each meeting within 5 minutes of the start time. Moderate participation is expected.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. Team members should meet deadlines or update the team with what is taking longer than expected and ask for help or clarification.

3. Expected level of communication with other team members:
    a. Team members should communicate if there are issues, a change in plans, etc.
4. Expected level of commitment to team decisions and tasks:
    a. Team members are expected to vote on every team decision.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    a. Client Interaction - Carter Kitelinger
    b. Python/ML Development Leads - Michael Watkins, Micah Gwin
    c. Frontend Development Lead - Brandon Richards
    d. Testing Lead - Nicklas Cahill
2. Strategies for supporting and guiding the work of all team members:
    a. Agile/SCRUM
3. Strategies for recognizing the contributions of all team members:
    a. After each sprint, the retrospective will capture all the accomplishments of each team member and progress they've made.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    a. Brandon Richards
        i. Frontend technologies (web, mobile)
        ii. Python
        iii. web scraping
        iv. Git
    b. Micah Gwin - Python/JS/NodeJS/C/Java development, Cloud development and security, NoSQL and SQL Databases, Linux/Shell, Terraform/Cloudformation, Drone and Jenkins CI, Git, React, Agile/Scrum, AWS Technologies like S3, DynamoDB, IAM, Lambda, EC2
    c. Alice Cheatum
        i. Python, Rust, Java, C, shell scripting
        ii. Linux
        iii. Backend development
        iv. Git
        v. Common cybersecurity tools
        vi. Penetration testing / kali linux
        vii. Server administration / network services
        viii. cryptography
    d. Carter Kitelinger - C, Java, Linux, Kali(Cyber Security tools), Python, Bash Scripting, Git
    e. Michael Watkins

i. Risk analysis/automated risk assessment
ii. Most common cybersecurity tools
iii. Networking
iv. Deep knowledge of IDS/netflow/log message parsing and alerting (SIEM)
v. Git/SVN, and code collaboration/scrum
vi. Using data sources: Twitter, MS-ISAC/CISA, Cisco Talos, security advisories
vii. Python/Go/Rust/Java/C, powershell/bash
viii. Web and desktop application development
ix. Linux sysadmin
x. Web scraping
xi. Backend data management (elastisearch, psql, nosql etc)
xii. Basic machine learning

f. Nicklas Cahill
i. C, Java, Bash/Powershell Scripting
ii. Linux
iii. Penitration testing/Kali Linux
iv. Network Defence/Firewall management
v. Backend data management(SQL)

2. Strategies for encouraging and supporting contributions and ideas from all team members:
    a. Questions will be asked in an open discussion and before moving on team members will be asked if there are any more comments and/or concerns.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. Write down which events, decisions, or discussions they were not properly included in. Bring to the team, then advisor.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. Deliver expected deliverables in a timely manner.
    b. Learn from people more experienced in a skillset.
2. Strategies for planning and assigning individual and team work:
    a. Use Agile/SCRUM to pick up work by experienced individuals and others who want to learn.
3. Strategies for keeping on task:
    a. Review minutes, add next week's tasks to Todo list (sprint), have retrospective after week is over.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

    a. Document infractions with link to corresponding rule in team contract, take to team and discuss if any rules need to be changed or additional support is needed.
2. What will your team do if the infractions continue?
    a. If the infractions continue, we will first raise the concern with the faculty advisor, then the professor.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*

b) *I understand that I am obligated to abide by these terms and conditions.*

c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Carter Kitelinger_____ DATE __09/16/2022_____

2) Micah Gwin_____DATE __09/16/2022_____

3) Nicklas Cahill_____ DATE __09/16/2022_____

4) Alice Cheatum_____ DATE __09/16/2022_____

5) Brandon Richards_____ DATE ___09/16/2022_____

6) Michael Watkins_____DATE ___09/16/2022_____

7) _____ DATE _____

8) _____ DATE _____