Requirements Specification

Team ZAM

4 May 2023

Instructor: Michael Leverington

Sponsor: Tim Wojtulewicz

Mentor: Daniel Kramer

Team Members: David Knight, Akiel Aries, Cody Beck, Nathan Chan

Version: 1.1

Accepted as baseline requirements for the project:

Client: _____ Date: _____

Team: _____ Date: _____

Table of Contents

1. Introduction

When deciding what precautionary actions to take to secure an organization's data, network security is a crucial consideration. By prioritizing network security, businesses can save a ton of money and relieve themselves of the fear of falling victim to cyber attacks. In a 2018 report[1] by The Center for Strategic and International Studies (CSIS), they discussed the economic impact of cybercrime and revealed that nearly $600 billion is lost annually. Fortunately, we have Zeek: an open-source network traffic analyzer software that provides a powerful and versatile set of tools for network security monitoring. Zeek also has the capability to measure performance and perform troubleshooting on a given network. These services all provide assistance concerning defending against malicious attacks.

We have partnered with Tim Wojtulewicz, a senior engineer and current release manager for Zeek. Tim and the Zeek team aspire to supply an open network monitoring framework that is able to empower their users by providing them with substantial insights into the activity of their systems. Zeek supports all users in need of their services ranging from simple home offices to the largest and fastest research and commercial networks. Zeek strives to build and support a dynamic community of developers, educators, and users. To achieve this, developers here aim to strengthen the Zeek ecosystem, allowing all users to deeply understand and defend their networks.

One of the ways Zeek has grown their ecosystem is by implementing the Zeek Package Manager Website[2] which allows users to create and publish their own third party scripts and plugins. By doing this Zeek is able to consistently provide users a wide variety of tools to enhance their network analysis.

[1] http://csis-website-prod.s3.amazonaws.com/s3fs-public/publication/economic-impact-cybercrime.pdf
[2] https://packages.zeek.org/

2. Problem Statement

Giving users the ability to share their own packages is a major factor when it comes to the workflow of the package manager site as it affects the volume in which packages are used and updated. Additionally, the most important feature the site lacks is an efficient search engine causing the users face several challenges when it comes to discovering and managing packages. The search engine is considered substandard and often returns irrelevant results. Moreover, the lack of consistency in the tagging practices used to label packages further complicates the problem, and makes it hard for users to identify the right package to suit their needs. This forces users to spend a considerable amount of time sifting through package descriptions. This not only hinders productivity but can also lead to a loss of confidence in the Zeek. Additionally, the current design of the package manager site is outdated and is in need of an update.

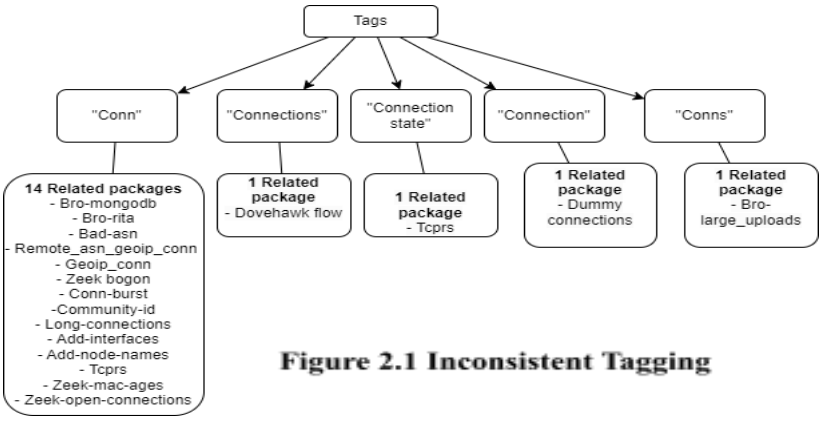

Figure 2.1 Inconsistent Tagging

Figure 2.1 represents the package results related to connections. There are five different variations of the tag and for the most part they all give distinct packages. We assume that the first use of the tag was “Conn” being that it includes fourteen related packages but somewhere along the way the lack of clear tagging practices deferred the results.

2.1 Deficiencies Deficiencies of the current state of the Zeek Package Manager website.

- User interface not visually appealing for users.
- The current search engine often displays irrelevant results.
- Updating the website with updates to new and existing packages needs to be automated.
- Inconsistent and inaccurate tags.

3. Solution Vision

To solve the problems with Zeek's current package manager site, we propose a new website that will provide a modern design, and will provide users with the ease of use they need to efficiently find packages. To do this, we will overhaul the package discoverability system, which is currently centered around search and tags. Additionally, we will overhaul the package viewer webpage to keep the current look and feel of the main Zeek website, these changes will allow users to spend less time searching for packages, and more time innovating network security solutions.

3.1 Specific Features

- A fast and effective search engine.
- A new package parser that uses a centralized metadata file to generate standard information about every package.
- A new package scraper that scrapes the *README* files of each package in order to highlight information that package publishers think is important.
- All of this will be implemented without the use of a database, making the back-end of the website more maintainable for the Zeek team.

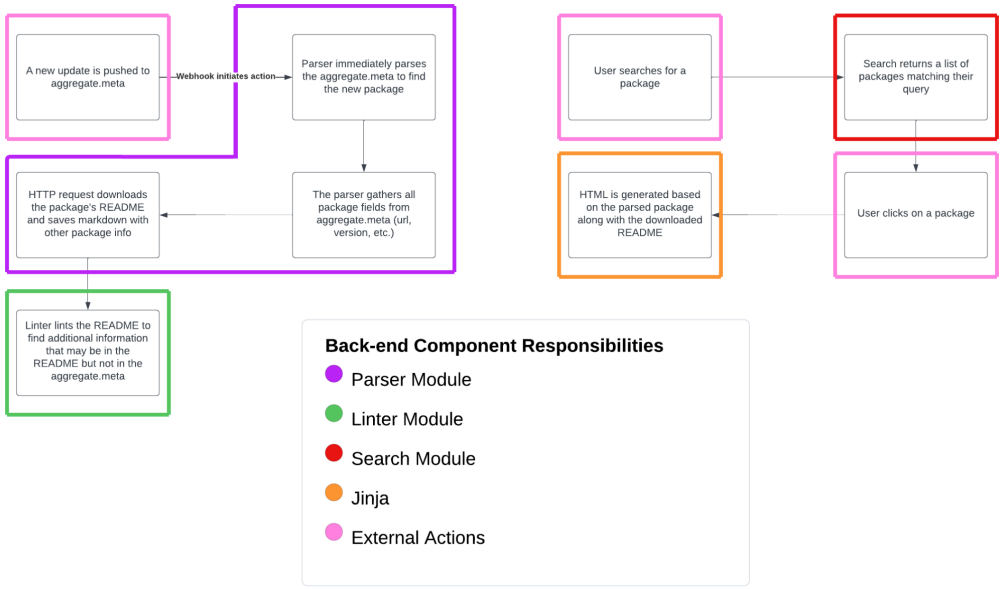

Figure 3.1 Back-end Components

Figure 3.1 shows a high level design of the various components that we plan to implement, along with the responsibilities they will have as either updates are pushed to the website, or the user interacts with the website.

4. Project Requirements

To create a website that both users and the Zeek development team are satisfied with, we will need to implement the following features: a fast, accurate search engine; an efficient metadata parser; a *README* scraper; and a modern front-end design. Each of these high-level functions will be broken into smaller, more manageable sub-functions in order to modularly implement the required functionality.

4.1 Functional Requirements

In assessing the functionality needed to complete this project, four major components are needed. Through internal team discussion and discussion with our client, we have identified the steps that will need to be taken to implement the search engine, metadata parser, *README* scraper, and the front-end design. These steps can all be represented in the form of different functions, which can be broken down into subfunctions, until base functionality is reached.

4.1.1 Search Engine

To appropriately implement the search functionality, we must not focus on searching for packages, but rather on ranking the packages based on search queries. After all, our users do not just want to enter a search query and see packages as results; our users want to enter a search query and see relevant packages as results. So, to provide relevant results, we have decided to implement our search functionality using the BM25 ranking algorithm, which assigns a numerical value to each package based on the search query. After each package has a score assigned to it, we will need to apply a bias to these scores, as not all packages will have extensive descriptions or *README*s, which will skew the results returned from the ranking

algorithm. After biasing these package scores to bump up the packages that have the search query in the name and/or tags, we simply need to sort the packages by their scores from highest to lowest, and return that sorted list of packages. From there, the front-end of the website can display the correct packages.

To implement the BM25 ranking algorithm, we must break it down into its constituent parts. First off, the BM25 algorithm is a modified version of the term frequency-inverse document frequency (TF-IDF) algorithm. TF-IDF works to properly rank text documents by weighing the frequency with which search terms appear against the number of documents in which the search term appears. This gives rarer terms within a search query more importance, as words like “the” will appear in almost every document, and will not be a good indicator of relevance. So, to implement the base TF-IDF algorithm, we will need to compute the term frequency of each word in the search query, by going through each package document to find the number of times each term appears. Then, we will need to implement the inverse document frequency part of the algorithm, by counting the number of documents each word in the search query appears in, and calculating the IDF score. Then, we will need to combine the TF and IDF scores by calculating the overall score according to the BM25 algorithm. To calculate the overall BM25 score, we will need to implement functionality to find one more calculated value, the average document length. With the TF, IDF, and average document length calculated, we can then get the overall score for each document.

4.1.2 Metadata Parser

To properly implement the parser, we work through a metadata file hosted on Zeek’s GitHub. In parsing this file to retrieve information about each and every package, we will need to

step through the file, we will need to gather *README* files off of each package's GitHub repository, and we will need to save all of our parsed information.

To step through our metadata file, we will most likely need to employ regular expressions, which use sequences of characters for pattern-matching in text. The metadata is stored in TOML, a data interchange format, making it easy to find the beginning and ending lines of a package, as we just need to look for opening and closing square brackets. Between sets of brackets, we will be able to find all the information package creators have added to this metadata file, including credits, aliases, descriptions, URLs, script directories, versions, building and testing commands, and versions, where each new item of information is on a new line. If there is more than one line of text for a given piece of information, all additional lines will be indented by four spaces. So, we will need to implement functionality to step through the metadata file to gather information based on this format.

Additionally, we need to determine what kind of information is stored on a given line. In doing this, we will have implemented all of the functionality needed to successfully parse the metadata file. After parsing the file, we will need to find each package's associated *README*, if one exists. Fortunately, each package has an associated URL, so the only functionality needed to get the *README* is to make requests to each package's URL asking for a *README* file. Once we receive the proper response from our requests, we know we have the package's *README.* Finally, once we have all of the necessary information about each package, we will need to implement functionality to save this information to a static file. We need to convert all package information to json format and write this formatted information to a file. After implementing these functionalities, we will be able to use the central metadata file to generate static files containing all of the available information about each package.

4.1.3 *README* Scraper

In order to implement our *README* scraper, we will need to have *README* files for each package on hand. Fortunately, the metadata parser already handles the task of getting *README*s from the internet and saving them, along with other package information, in json format.

First, we must specify the purpose of scraping each package's *README* file. While packages are able to specify important information, such as dependencies, build commands, etc., in the respective entries in the central metadata file, not every package does so. However, many packages do specify this information within their *README* files. So, the purpose of this scraper is to try to gather this important information from the *README* so that we can highlight it on package webpages. This will improve the user experience, as users will not have to skim through a *README* to find information, but instead, they can go to a specific section of a package's webpage in order to find the information they seek.

As entries in the metadata file are not uniform, the responsibilities of the scraper will not be uniform from one package to the next. If a given package has next to no information in its metadata entry, the scraper will have to try to find as much information within that package's *README* file. If instead the package has a lot of information within its entry in the metadata, the scraper may not have much to look for. So, the first task that the scraper must accomplish is to determine exactly what information it is looking for. A way of doing this is to simply compare the fields in the package's json file to the maximum number of fields that could be found in any package's json file. The missing fields are the one that the scraper should look for in the package's *README*.

Once the scraper has determined what it should be looking for, finding the information is not all that difficult. The scraper can simply make use of regular expressions to search for the things it needs. For example, if the scraper wants to find a package's build commands, it can just step through the *README* line-by-line while searching for the words "build commands". Additionally, as most *README* files are written in GitHub flavored markdown, the scraper could only check lines which are specified in the markdown as headers, since important terms such as "build commands" are likely to be the header of a section of text. Once it finds what it is looking for based on the header, it only needs to step through the text until it finds a code block in the text, again utilizing the fact that code blocks look different that normal text in markdown. Finally, the scraper can just pull the text out of the code block.

Not every single package will have a *README* file written in markdown. Yet, the majority of *README*s are written in markdown. So, we can simply specify that if the creators of a package want the scraper to work with their package, they must have a *README* file written in markdown.

Through this step-by-step, line-by-line process, we can bring to life a package scraper that is able to pull important information out of every *README* file, so that we can highlight this information on each package's webpage.

4.2 Non-Functional/Performance Requirements

Now that the functionality of each major component has been specified, we need to specify how each of these components will perform. For each component, the search engine, metadata parser, *README* scraper, and front-end, we have laid out a series of performance requirements that the components must meet. Once each component has achieved all of its requirements, it will be considered successfully implemented.

4.2.1 Search Engine

Two key metrics are applicable when evaluating the performance of a search engine: speed and relevance. Obviously, a search engine must be able to find results to a search query quickly, otherwise the user will either choose not to use the search engine, or choose not to use the website altogether. Additionally, the search engine must provide relevant results, otherwise the user should just navigate through a list of all available packages to find one that suits their needs.

Modern search engines are fast. So, in order to keep users on our website, our search engine must be fast. Our search engine needs to be able to take in a search query and serve results in less than half a second. In testing search engines around the internet, in a variety of domains, from Google to PyPi to GitHub, we have found that most search engines are able to return results in less than half a second. So, we have adopted this as our benchmark for a successful search engine implementation.

Although having a fast search engine is important, what is arguably more important is relevance. A good search engine needs to be able to provide users with a list of relevant results, yet defining a benchmark for relevance is a difficult task. Fortunately for us, Zeek currently has only 223 packages. So, we have decided to manually curate four lists of the five relevant packages for what we think will be common search terms: “ssh”, “cve”, “ja3”, and “spicy”. We have decided that once the search engine returns lists of packages containing our choices within its top eight choices, then it has done a good job of returning relevant results.

Once our search engine has met both of these performance requirements with respect to speed and the relevance of results, we contend that it has been successfully implemented.

4.2.2 Metadata Parser

The metadata parser has a rather simple requirement to meet in terms of performance: it has to be correct. Unlike the search engine, the parser is not a user-facing component, and it will only need to run in standard intervals throughout the day. As such, the parser need not be fast. However, it must be correct one-hundred percent of the time. The information that the parser gets from the aggregate metadata file will live on a package's webpage until the next time that the parser runs, an amount of time that will likely amount to a couple of hours. As such, the parser must not make errors when collecting information. Additionally, when it goes to search for a *README*, it must be able to find a *README* if one exists. This all amounts to giving us one requirement. The parser must find the information it needs correctly, and it must never fail to collect and save all of the information.

4.2.3 *README* Scraper

Much like the parser, the scraper has the same simple requirement: it must be correct. The scraper is also a component that the user will have no direct interaction with, and it will only be run when the parser is run. So, information it provides will also live on the website for at least a couple of hours, but it does not need to be fast. What the scraper needs to do is accurately assess the information that is missing from a package's json file, and comb through the package's *README* to find it. This means that if the information is located within a properly formatted *README* file, the scraper must be able to find it one-hundred percent of the time. Like the parser, the scraper must be able to do its job accurately, without fail, all of the time.

4.2.4 Front-end

The current repository website is not up to the clients standards in terms of the front-end look and appeal of the overall website. The website is lacking in terms of style, especially when compared to their current Zeek website[3]. In order for this project to be successful, the front end should look visually pleasing, while maintaining the integrity of the information we scraped from the packages. This means that there should not be information that does not load and the information should be accurate to the current package being viewed. The website should also be able to handle multiple viewports so the user has a pleasing experience regardless of their device type. Users on phones, tablets, and desktops should all be able to enjoy their experience the same as the next user.

These features are much more achievable now then in previous years thanks to updated web frameworks that make creating organized and good-looking websites much easier. Where previously developers would need to use tables for laying out pages or misuse CSS properties like “float” to align things horizontally, we are now able to use more updated and supported modules and frameworks to easily create assets to display content that will be easily maintainable within simple HTML and CSS rules and classes. Frameworks like these also allow for us to keep mobile users in mind with features like hamburger menus for site navigation, as well as resizing and moving items around to ensure the site remains readable without ruining the look of the website. These “mobile friendly” features are also currently utilized on the current Zeek website.

[3] https://zeek.org/

Figure 4.1 The zeek.org Homepage on mobile (left) and desktop (right) devices

To keep up with the look and feel of the current website, we want to ensure that it remains mobile friendly, contains information about Zeek on the homepage, and uses some of the same elements such as colors, fonts, and images. Our client has provided us with the logos and some of the CSS that they currently use on their website. Features like buttons with rounded edges and shadows will be replicated as seen on the current Zeek website.

Figure 4.2 Button from zeek.org

The page for searches should return the results neatly in a vertical list similar to the current Zeek package website, however we will need to improve the style of this list visually. The current screen for “Get Zeek” on the main website has vertical boxes that look more visually pleasing which would be a good style to follow throughout the page. Some of the pages store

content makes use of a card like system with border radius and shadows. For searches and packages, these style choices seem appropriate for the new package repository website.

Pages like the home page may not need to make use of this type of content and would make use of a more linear and textual format including a hero image and search bar to search for packages, with information about Zeek as well. For our inspiration, the Zeek team said that a site like the Rust Package Registry[4] had some solid examples of more useful data to incorporate as a stretch goal down the line.

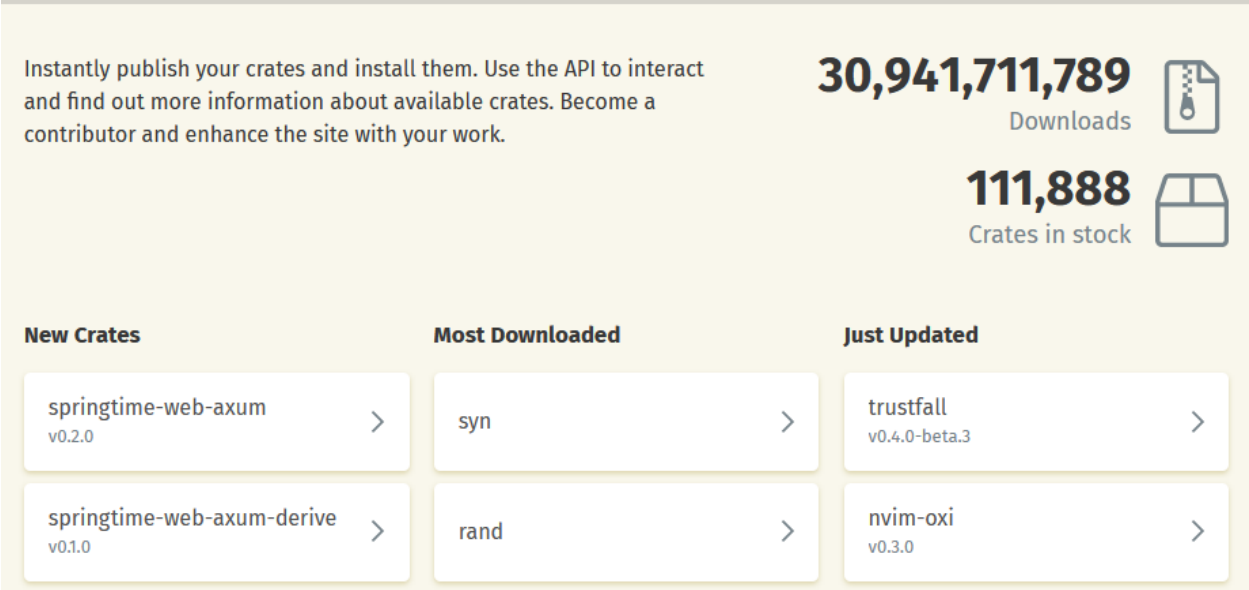

Figure 4.3 Homepage from crates.io

4.3 Environmental Requirements

From an environmental standpoint, there are few constraints placed upon this project. The constraints that do exist are more related to the deployment of the website rather than anything to do with the development of the website.

[4] https://crates.io/

With that being said, the deployment will be containerized using Docker, and deployed using Amazon Web Services (AWS). The website will also utilize a Python web server on the back-end, as such technologies, like FastAPI, Django, and Flask, make websites easy to maintain. Given these environmental constraints, we have a very open sandbox in which to implement our solutions to the problems presented by this project. Our implementations can happen using different popular languages, such as Python, C, or Java, so long as everything looks like a Python module to the web server.

Deploying using Docker will provide more work to be done to set up the website on an AWS instance, as we will need to create a docker compose file in order to automate much of the setup process. Using Docker will ensure that so long as Docker is properly configured on the AWS instance, we will not need to fiddle around with many of AWS' configuration settings to keep the website up and running, as Docker keeps the operation of the web server logically separated from the host system.

4.4 User Stories

In order to efficiently summarize all of the requirements, we have created user stories related to our requirements. This will allow for ease of use in discussions about requirements, help to direct implementation, and provide simple reminders of the requirements as we move forward with this project.

- As a user, I want to be able to efficiently search for packages, so I do not have to waste time finding the package that suits my needs.
- As a package creator, I want the search engine to return relevant packages, so I know that users will be able to find the package I have created.

- As a package creator, I want the information in my package's *README* file to be highlighted on the website, so my package's users do not have to waste time searching for it.
- As a website maintainer, I want information about packages to be automatically updated based on the aggregate metadata file, so that I do not have to make any changes to the website when creators make changes to their packages.
- As a website maintainer, I want package information to be properly stored utilizing static files, so that the website is not over-engineered.
- As a website maintainer, I want the ability to easily start and stop operation of the website at a moment's notice, so that when problems arise I can fix them easily.

5. Potential Risks

Since the Zeek package manager website relies on accuracy of the data sources of the package information and our methods of collecting it risks are involved with displaying accurate and relevant data to users. We feature methods of parsing the central metadata file that Zeek stores package information that features a *TOML* style format. Since Zeek's 3rd party packages are open source including *READMEs* for a general overview of the package, we are able to scrape them for additional information without trouble. Given the methods in place for collecting package information, there comes a risk with the accuracy of our source data as there are not frequent updates to it. This is partly due to the fact that many of the packages we are working with have not been updated recently.

There are additional risks that come with the accuracy of our results from parsing and scraping the package information. Given that our parser relies heavily on the central metadata file's *TOML* like structure, any changes or updates to the format of the file itself would break the functionality of our website entirely. Mitigating this risk would require the authors of Zeek packages to change the entries of their package information to a more suitable format like *JSON* or *CSV*. This core risk with our information collecting process is the assumption the metadata file will keep the same format. These risks introduce another for our search engine as it is dependent on the accuracy of the data sources and the methods to collect it. The ability to return meaningful results given the surplus of information we will gather could mean issues with very simple or more complex queries from end users.

Our application makes use of data collection and searching as well as injecting the gathered data into unique *HTML* pages based on the Zeek package name. Each package page should include information only relevant to that package, however the risk of *HTML* pages being

filled in with mismatching information becomes a possibility. Given that Zeek only contains around two hundred packages, we will want to take into account the possibility of this number increase and assuring our methods of data gathering and searching can scale to larger numbers.

These issues all lead to a less than satisfactory experience with our application since the sole purpose of the website is to display accurate information pertaining to Zeek's 3rd party packages. Improper information injection into the *HTML* pages could lead to users seeing irrelevant data. Our search algorithm not being able to work efficiently given a plethora of text to search could lead us to implementing a lacklust search engine similar to the one in place also leading to a less than ideal user experience.

Given that many of Zeek's 3rd party packages have not been updated in recent time, the likelihood of outdated information being displayed to users increases drastically. This raises an issue of packages which have not been updated recently being incompatible with current versions of Zeek which could cause misleading results on our website. While our methods of collecting data prove sufficient and accurate, further testing these methods will solidify their durability leading to a low probability of issues arising in the future with our source data collection and search methods.

6. Project Plan

Looking ahead, we have been able to create a small timeline with our plans of production in order to finish this project within the proper time frame. Finishing the project includes finishing the requested core functionality of the project as well as handing over the project to Zeek.

To finish the core functionality, we will need to finish the scraper for the *README* files, as well as work with our client to create a linter for *README* files. On top of this, we will also need to ensure that the *README* files are being displayed correctly and do not break the style and flow of the website. This also includes making sure that all information is being displayed and maintains the look and feel of the Zeek website. We currently envision these goals being completed some time by the end of August to early September .

Once this is finished, we will need to begin working with the client to deploy the project and we will begin extensive testing on the production server to ensure that the project's functionality is working as intended and is to the client's satisfaction. Afterwards, we can begin looking into completing our stretch goals such as generating usage statistics on packages, extending the linter to generate information to the aggregate metadata file, and finish documenting the project to make it easier to understand and maintain.

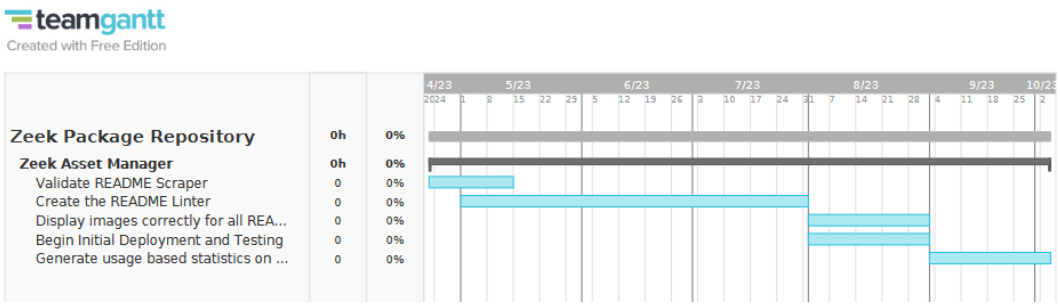

Figure 6.1 The Gantt Chart for our project plan

7. Conclusion

As technology continues to expand this calls for the network security sector to keep pace and evaluate how new risks can be mitigated. Fortunately, the Zeek Package Manager has been an excellent means to allow users to develop and share packages to enhance network analysis. Nonetheless, developers on the Zeek team feel that the user experience of the site can be reworked and will then alleviate the current technological challenges users face.

The problem revolves around improving the current user experience by making discovering and managing new packages easier. This is detrimental to the workflow because it adds to the time spent filtering through packages and can cause users to lose confidence in the utility of Zeek. Another problem we are faced with is revamping the quality of the website. Zeek wants the package manager website to have a modern up to date look while still preserving the overall look and feel of Zeek.

As a team we are confident in our plan to manufacture a new website that will retain a modern, sleek design and will provide users with an intuitive experience with regard to discovering packages. On top of that, we plan on implementing a new search engine utilizing the BM25 ranking algorithm. While also developing a new package parser and scraper that allows us to generate the standard information of each package and scrape the captured *README* files to potentially gain more information of a given package.

In this document we discussed the features and functions of our proposed solution, including the functional, performance and environmental requirements. We also highlighted the potential risks of our system and our current plan of execution moving forward. As a result, this solidifies a thorough set of requirements that will guide the fruition of the new Zeek Package Manager website and we look forward to providing Zeek and its users a newly enriched user experience.