HackArmour: A Search Engine For Hackers

Ujjwal K. Kumar[1] • Rishiraj S. Behki[2]

hackarmour.com

[1] Guwahati, Assam, India. ujjwal-kr@hackarmour.com (***First author***)

[2] Vadodara, Gujarat, India. rishiraj@hackarmour.com (***Corresponding author***)

ABSTRACT

The HackArmour search engine is a comprehensive tool that has been specifically crafted to cater to the needs of security researchers and infosec enthusiasts. Utilizing a microservices architecture, the HackArmour search engine has been implemented using the popular programming languages NodeJs[1] and Golang[2]. As a result, it offers an array of features and functionalities that make it a valuable resource for the security community. One of the key features of the HackArmour search engine is its ability to search for Common Vulnerabilities and Exposures (CVEs)[3] and Reddit threads, providing users with access to a wealth of information that can be used to improve their understanding of current security threats. Additionally, the HackArmour search engine also offers a range of security feeds, providing users with up-to-date information on the latest vulnerabilities and exploits. Furthermore, the HackArmour search engine also boasts a curated collection of resources and exploits, which can be used by security researchers to develop new methods for protecting against cyber threats. In this paper, we will be delving into the design and implementation of the HackArmour search engine, evaluating its performance, and discussing its significance in the field of security research.

Keywords — Search Engine, Information Security, Microservices

I. INTRODUCTION

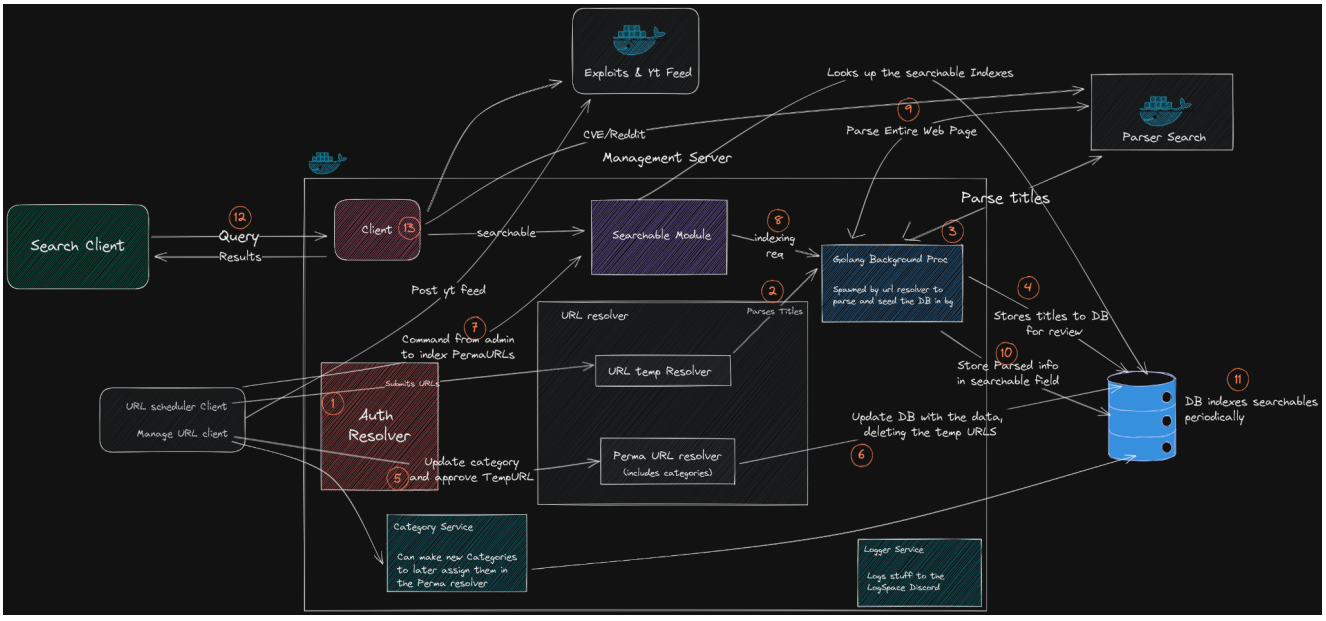
Security research is a crucial and rapidly advancing field, with new vulnerabilities and exploits constantly emerging. To keep abreast of the latest developments, security researchers and infosec enthusiasts require access to a broad spectrum of resources and tools that can aid them in discovering and analyzing information. However, the vast expanse of information available on the internet can make it challenging to locate the most pertinent and reliable results. The sheer volume of data can be overwhelming, and it can be difficult for researchers to separate the signal from the noise, to identify the most relevant and accurate information that can be used to improve their understanding of current security threats.

To address this issue, we developed the HackArmour search engine, which is a comprehensive resource for security researchers. The HackArmour search engine is a web-based application that enables users to search for CVEs and Reddit threads, as well as get security feeds and access a curated collection of resources and exploits, all in one spot. With a microservices architecture and a focus on logging and monitoring, the HackArmour search engine is meant to be extremely dependable and fault resistant.
Furthermore, the HackArmour search engine allows members of the community to submit their own resources and exploits, ensuring that the information offered is constantly current and relevant.

II. IMPLEMENTATION

The HackArmour search engine was built with a combination of Typescript and Golang. We picked Typescript (nodejs) for its IO capabilities, and it is already well-suited for constructing the search engine's user interface and front-end (react). Golang was selected for its concurrency support and capacity to handle enormous volumes of data, making it ideal for constructing the search engine's multithreaded broker and indexer services. The search engine was built with a microservices architecture for increased scalability and flexibility. Each microservice is in charge of a different part of the search engine, such as indexing and searching YouTube videos or looking for vulnerabilities and exploits.

A custom build system using **make**[4] was also developed for the search engine to streamline the development overflow and save time. It sets the dev environment using hot reloading features with docker volumes, installs dependencies, and configures, seeds and migrates the database. It is also used in production for deployment.



[Full Sized Image Here]

Both the **Search Client** and the **URL Scheduler Client** are developed in **Typescript** and **Reactjs**. All requests from the scheduler client are sent through the main server's Authentication module, which employs **bcrypt** and **JSON Web Tokens (JWT)**. It determines whether or not the user is a member of the staff. The background processor is developed in **golang** and communicates with the parser container (written in **nodejs** and **express**) and the database (**MYSQL**). The diagram also shows the exploit search and YouTube feed container, which is built with **nodejs** and **express**.

III. WORKFLOW

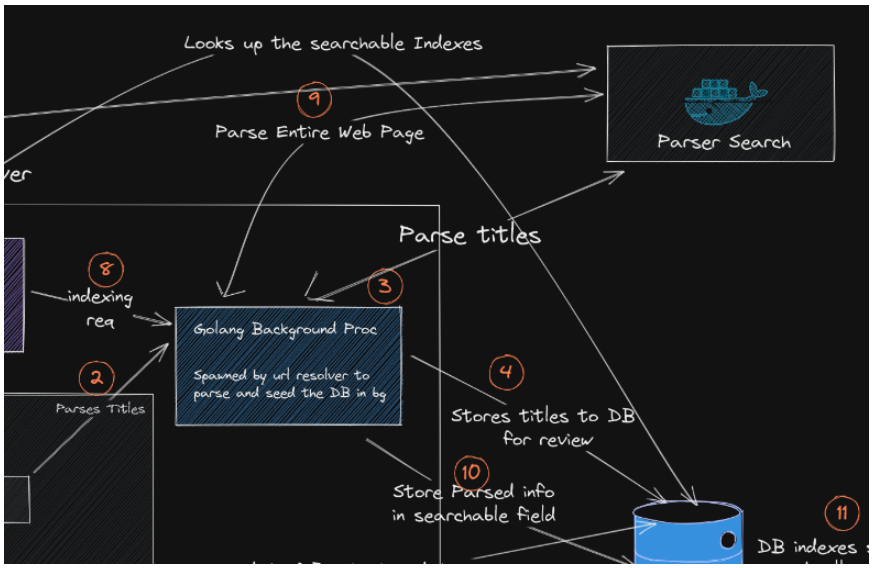
The numbers in the diagram are to show the rough timings of the events that can occur in the search engine. The events can be described as follows:

i. Staff shares URLs that could be indexed in the search engine.
ii. The URLs are stored in a temporary table in the database called *brokerTempURLs*.
iii. This event starts the broker, and it fetches all the URLs from the mentioned table and fetches its title through the **Parser** container, and stores them in a new table called *tempURLs,* deleting the previous table.
iv. Later on when the higher staff checks on the lists of tempURLs, they can approve a URL to be valid to be later indexed, along with attaching the category and optionally editing the existing information. This new data is stored in the *PermaURLs* table, and the *tempURLs* table gets deleted.
v. When there are a significant number of URLs in the PermaURLs table, the **broker** is called again to index the PermaURLs.
vi. The **Broker** then takes the PermaURLs with the additional information attached to it and using the **Parser** again, it extracts more information about the web page.
vii. This information is then stored into the *Searchable* table, waiting for the database to index it later on, and deleting the *PermaURLs* table.
viii. Now, the **Searchable** module in the main server can serve the indexed queries through the **Client** module.
ix. The **Client** module is also a router for all the client requests, as it fetches Exploits, CVEs and Youtube feeds from the other containers.

It should be noted that the workflow is quite fault tolerant, thus tables are not completely deleted when they are passed to the next stage. It uses error handling mechanisms to determine whether the operation was successful (moving it to the next stage). If an unforeseen error happens, the system will recover without damaging any data.

Furthermore, this is the current workflow as of the authoring of this article, and it is susceptible to change due to additional scaling requirements or changes in our specification. The **Parser** and **Broker** will be explained in full later in this article.

IV. CONCURRENCY AND FAULT TOLERANCE



As previously stated, Golang was selected for its concurrency features. It is a background process that is launched by the main server in the main server container itself. In order to improve efficiency, it sends requests to the parser at the same time. When compared to single threaded processes, the reported performance boost is about double. It is spawned when there is a post request for URLs, as well as when there are sufficient permanent URLs. There are several scenarios in which this may fail, thus various safeguards are in place to ensure its dependability:

- If multiple users post URLs at the same time, it can lead to multiple running instances of the broker which will lead to data corruption.
    - This is solved by storing the broker state to the database, if the broker is busy the URLs will just be stored and the broker won't be called.

- If the server stops unexpectedly, the data currently being processed may be lost.
    - The data from the previous stages is only erased when it has been verified that it has securely progressed to the next level. Because of batching, it won't have to restart a huge transaction if it fails somewhere along the road.

- If there are a lot of (say a million) URLs gathered already in the temporary broker table, it may lead to a memory overflow, which will crash the broker.
    - This will never happen as the broker does its operation in batches, so a large amount of data will never be copied in memory.

V. RANKING SYSTEM

There is no click-based or backlink-based ranking system like Google.
We also do not have a large enough data set to accomplish automatic ranking utilizing machine learning. The present ranking mechanism is manual; a weight is applied when our higher staff categorizes URLs, and when the user searches the results, the query prioritizes the URLs with the most earned weight. This method will be repeated until we have a sufficient collection of resources. Following that, we may combine the visit counts with the weight to alter the ranking of a specific page or resource.

```
score = (visits/totalVisits) * weight
```

The weight is a number between 0 to 100, and it always remains constant. It is also to be noted that this will only contribute to a minor part of the ranking algorithm, as visits can be increased with spam. Other parts of the ranking algorithm will be decided by ratings, and timely staff reviews.

VI. FEEDS CVEs AND EXPLOITS

Hackarmour also enables users to search through exploits and CVEs. Aside from that, it provides YouTube feeds to provide the most recent information security topics. These capabilities are introduced to the engine to make it easier for users to access these resources.

The **Exploit & Feed** container, developed in Python, is in charge of maintaining and broadcasting feeds as well as searching for vulnerabilities. It executes the container's *searchsploit*[5] command and delivers results depending on the output.

The **Youtube feed** is a feed maintained by our community. It is inspired by *securityTube[6]* which is no longer updated these days. This module in hackarmour tries to revive the likes of it. We are also planning to implement some other feeds such as the *google project zero[7]* and other popular security blogs in hackarmour itself.

One of the most essential feeds for security researchers is **CVE**. Even more significant than exploits or the most recent blogs. They are a compilation of the most recent publicly known vulnerabilities and exposures. We provide this as a **feed** as well as a **searchable** database.Because it is developed in Typescript and Node Js, this module lives in the **Parser-Search** container.
The Parser-Search container now includes **Reddit Search**, which allows the user to search for relevant information in the most popular *Reddit* discussions.

VII. SIGNIFICANCE OF THE SEARCH ENGINE

Our mission has always been to conserve and spread knowledge. And it's hackarmour's responsibility to do the same. A security researcher or a hacker can benefit from hackarmour in a variety of ways.

- They won't have to open many tabs or windows to hunt up CVEs, exploits, or publications if they can find them on hackarmour.
- They won't have to visit many websites, such as popular blogs or the CVE website, to stay up with various feeds and happenings in the security sector.
- In hackarmour, beginners may discover a clear roadmap as well as listings of open-source resources and labs.
- Hackarmour also provides a feed that keeps you up to speed on forthcoming CTFs and other conferences or events (We are also working on a CTF event platform).

CONCLUSION

The hackarmour community also hosts its own CTFs, and we have a very good team of hackers and engineers. This project is also a good opportunity for beginner developers or hackers to make infosec challenges, or work in a real world environment in case they want to help the community by improving the engine or making our specification better.

Making this search engine was a lot of fun, and I would like to thank all of the community members and contributors for their assistance in getting this project completed. It was one of the most enjoyable projects I've ever worked on.

I would like to thank Rishiraj in particular for managing the community, implementing some of the essential elements of hackarmour, and co-authoring this work. Furthermore, because the search engine is new, the ranking algorithm is not yet completely established and will experience significant modifications in its specification as and when additional data is supplied.

REFERENCES

[1] NodeJs: A javascript runtime environment https://nodejs.org
[2] Golang: The go programming language by google https://go.dev
[3] CVE: List of publicly known vulnerabilities https://cve.mitre.org/
[4] Make: A tool which controls the generation of executables and other non-source files of a program from the program's source files. https://www.gnu.org/software/make/
[5] Searchsploit: tool to search exploit-db https://www.exploit-db.com/searchsploit
[6] SecurityTube: Youtube but for infosec http://www.securitytube.net
[7] Google project zero: https://googleprojectzero.blogspot.com