# HackArmour: A Search Engine For Hackers

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

hackarmour.com

[1] Guwahati, Assam, India. ujjwal-kr@hackarmour.com

[2] Vadodara, Gujarat, India. rishiraj@hackarmour.com

## ABSTRACT

The HackArmour search engine is a tool designed to support the needs of security researchers and infosec enthusiasts. Built using a microservices architecture and implemented in NodeJs[1] and Golang[2], the HackArmour search engine offers a range of features and functionality that make it a valuable resource for the security community. These include the ability to search for CVEs[3] and Reddit threads, get security feeds, as well as access to a curated collection of resources and exploits. In this paper, we shall describe the design and implementation of the HackArmour search engine, evaluate its performance, and discuss its significance in the field of security research.

***Keywords — Search Engine, Information Security, microservices***

## I. INTRODUCTION

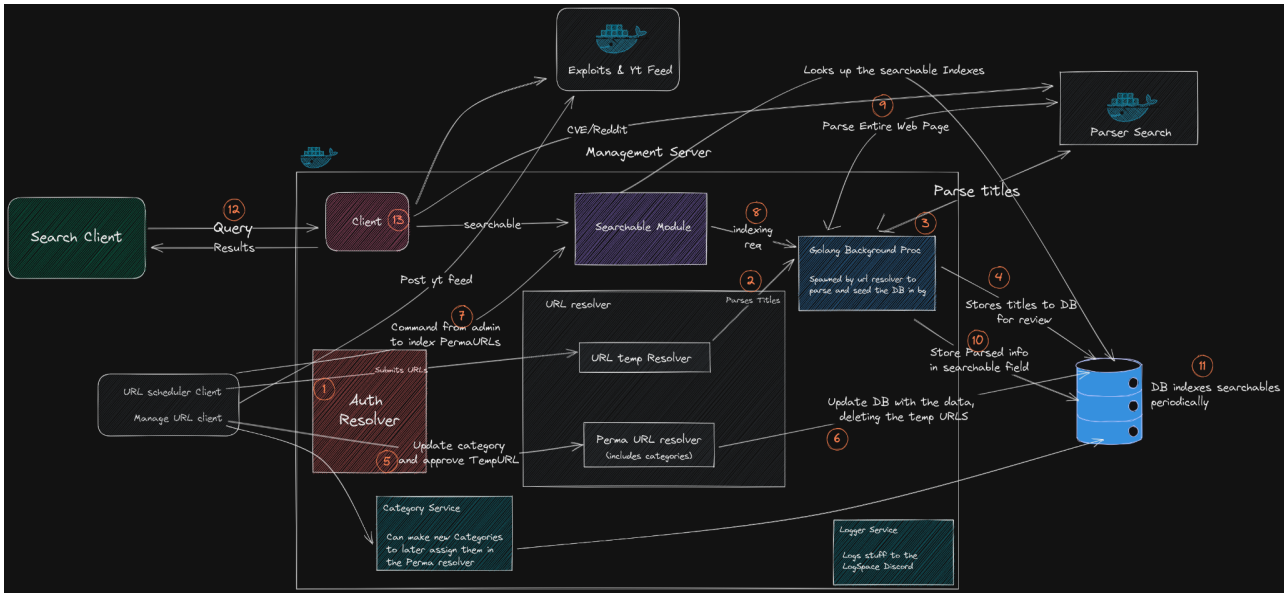
Security research is an important and rapidly growing field, with new vulnerabilities and exploits being discovered every day. In order to stay up-to-date with the latest developments in the field, security researchers and infosec enthusiasts need access to a wide range of resources and tools that can help them find and analyze information. However, the sheer volume of information available on the web can make it difficult to find the most relevant and accurate results.

To address this challenge, we have developed the HackArmour search engine, a comprehensive resource for security researchers. The HackArmour search engine is a web-based tool that allows users to search for CVEs and Reddit threads, get security feeds, as well as access a curated collection of resources and exploits, all in one place. The HackArmour search engine is designed to be highly reliable and fault tolerant, with a microservices architecture and a focus on logging and monitoring. In addition, the HackArmour search engine allows the community to contribute their own resources and exploits, ensuring that the information available is always up-to-date and relevant.

## II. IMPLEMENTATION

The HackArmour search engine was implemented using a combination of Typescript[3] and Golang. We used Typescript (nodejs) for its IO  capabilities and it is already well-suited for building the user interface and front-end of the search engine (react). Golang was chosen for its concurrency support and ability to handle large amounts of data, which made it well-suited for developing the multithreaded broker and indexer service that power the search engine.The search engine was designed using a microservices architecture, which allows for greater scalability and flexibility. Each microservice is responsible for a specific aspect of the search engine, such as indexing and searching YouTube videos, or searching for vulnerabilities and exploits. This modular design makes it easier to add new features or update existing ones, as well as to improve the performance and security of the search engine.

A custom build system using **make** 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]

The both **Search Client** and **URL Scheduler Client** are written in **Typescript** and **Reactjs**. All the requests from the scheduler client passes through the Authentication module from the main server, which is implemented using **bcrypt** and **JSON Web Tokens.** It validates if the user is a staff or not. The background processor running inside the main server is written in **golang**, and it talks to the parser container (written in **nodejs**

and **express**) and the database (**MYSQL**). The exploit search and youtube feed container can also be seen in the diagram and is built using **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:

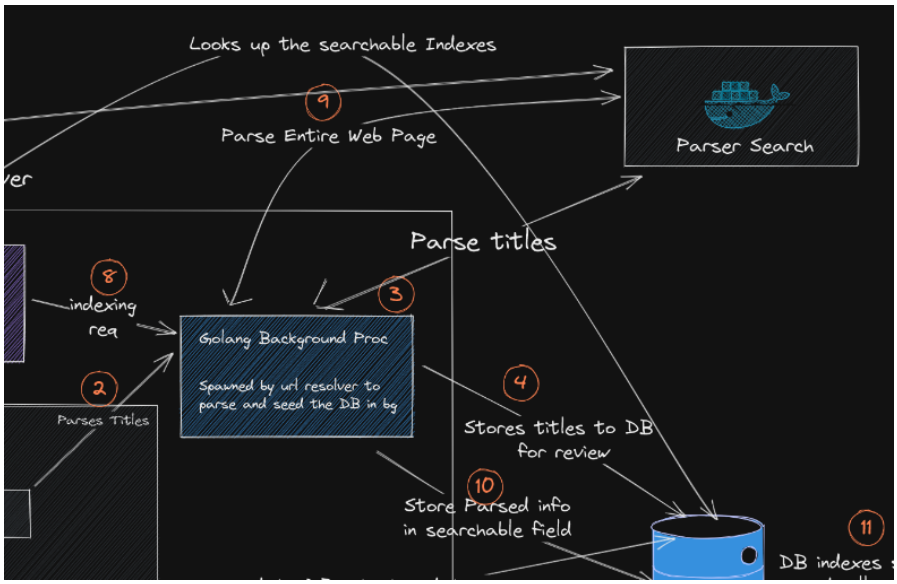
i. Staff shares URLs that could be indexed in the search engine.
ii. The URLs are stored in a temporary table in the database called *brokerTempURLs.*
iii. This event fires the broker, and it fetches all the URLs from the mentioned table and fetches its title through the **Parser** container, and stores 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 is to be noted that the workflow is quite fault tolerant, so it does not delete tables entirely after they are moved on to the next stage. It verifies if the operation was successful (of moving it to the next stage) using some error handling methods. If an unexpected error occurs, the system will recover itself, without corrupting a single bit of data.

Also, this is the current workflow while writing this paper and it is subject to change for some other scalability needs or changes in our specification. The detailed working of the **Parser** and **Broker** will be described later in this paper.

## IV. CONCURRENCY AND FAULT TOLERANCE

As said, **Golang** was used for its concurrency features. It is a background process spawned by the main server, in the main server container itself. It concurrently fires requests to the parser in order to increase performance. The recorded performance increase is almost 2x when compared to single threaded operations.  It is spawned when there is a post request to post URLs and also, when there are enough permanent URLs.




There are a lot of cases this can fail, so some checks are put in place to make it a reliable piece of software:

- 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 in the previous stages is only deleted after verifying that it has reached the next stage safely. Batching is the reason it won't have to start a large transaction again if it fails somewhere along the way.

- 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 ranking algorithm on the basis of clicks or even backlinks like google. We do not have a massive collection of data yet to perform automated ranking using machine learning either. The current ranking system is manual, a weight is added at the time of categorizing URLs by our higher staff and later when the user is searching the result, the query gives priority to the URLs with the most earned weight. This process will continue till we have a good collection of resources. After that we can incorporate the visit counts with the weight to adjust the ranking of a particular page or resource.

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

The weight is a number assigned by the staff while categorizing. 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. Apart from that, it also offers youtube feeds, to show the latest infosec content. These features are added to the engine to make these resources easy to reach to the users.

The **Exploit and Feed** container which is written in python takes care of managing and broadcasting feeds, and searching for exploits. It runs the *searchsploit[4]* command in the container and returns results based on the output.
The **Youtube feed** is a feed maintained by our community. It is inspired by *securityTube[5]* 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[6]* and other popular security blogs in hackarmour itself.

**CVE** feeds one of the most important feeds for security researchers. Even more important than exploits or latest blogs. They are a list of latest publicly known vulnerabilities and exposures. We provide this both as a **feed** and **searchable.** This module resides in the **Parser-Search** container, as it is written in Typescript and Node Js itself.
The Parser-Search container also contains **Reddit Search,** which allows the user to search relevant information in the most popular threads from *reddit.*

## VII. SIGNIFICANCE OF THE SEARCH ENGINE

Our goal has always been to preserve and distribute knowledge. And hackarmour's job is to do the exact same thing. There are a lot of ways in which a security researcher or a hacker can benefit by using hackarmour.

- They won't have to open a lot of tabs or windows to lookup for CVEs, exploits or some publications if it is available on hackarmour itself.
- They won't have to visit different websites like some popular blogs or the CVE website to keep track of different feeds and events in the security industry.
- Beginners can find a good roadmap and lists of open-source resources and labs in hackarmour itself.
- Hackarmour also has a feed to keep you updated about upcoming CTFs or some other conferences or events (We are working on a CTF event platform too).

## 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 as a founder would like to thank all the community members and contributors for helping us complete this project. It was one of the best things I have worked on. I would especially like to thank Rishiraj for maintaining the community, implementing some of the critical parts of hackarmour and as well as co-authoring this paper. Also, as the search engine is new the ranking algorithm is not fully developed and will undergo lots of changes in its specification As there is more data added.

## 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] Searchsploit: tool to search exploit-db https://www.exploit-db.com/searchsploit
[5] SecurityTube: Youtube but for infosec http://www.securitytube.net
[6] Google project zero: https://googleprojectzero.blogspot.com