Passive Dark Web Crawler for CTI Extraction and Visualization

(2025A) COMP-5473 - Computer Security

Supervised by Prof. Dr. Osama Amjad

Wael Fahmy
Student ID: 1265745
Lakehead University
wfahmy@lakeheadu.ca

Amirhossein Zeinali Dehaghani
Student ID: 1225496
Lakehead University
azeinali@lakeheadu.ca

***Abstract*—This project presents a passive crawler for extracting cyber threat intelligence (CTI) from the dark web. Its built on top of the Tor network, our system visits .onion sites, and find CTI-related keywords matches, and saves findings in a secure SQLite [1] database and encrypted CSV file. The crawler integrates a Streamlit-based graphical interface that make researchers to insert multiple sites and keywords, visualize frequency of keyword and analyses of NLP token , and also securely export the results. Our system prioritizes usability, privacy, and also automated encrypted data handling. Together, it makes it a powerful tool for cybersecurity analysts in order to passively and safely explore threat intelligence.**

***Index Terms*—Dark Web, Tor, CTI, Keyword Matching, SQLite, Streamlit, Cybersecurity, Encryption, Fernet, NLP**

I. Introduction

The dark web offers a hub for cybercriminal activity and it often hosts threat intelligence, including leak of credentials, samples of malware and also phishing kits. However, accessing and analyzing dark web data will have ethical, legal, and technical challenges because of the volatility, anonymity, and unstructured nature. Manual search is time-consuming and have its own risk. In response, we have developed a passive crawler which is designed to extract CTI from .onion domains with Tor network. The important features of system are live keyword matching, encrypted data export, NLP-enhanced visual analytics, and a user-friendly web-based GUI.

A. Project Domain Overview

This project is based on the intersection of cybersecurity, natural language processing (NLP), and dark web intelligence and Specifically focuses on:

- **Cyber Threat Intelligence (CTI):** Passive keywords collection, classification and also threat indicators from Tor which is considered as hidden service.
- **NLP for Security:** We use tokenization, stopword filtering, and word frequency analysis to extract relavant language patterns threat.
- **Dark Web Reconnaissance:** crawling .onion domains in a safe way while minimize the risk with using encryption and anonymity.

II. Motivation and Significance of Project:

The dark web has a crucial role in the dissemination of cyber threat intelligence (CTI) consisting of leaked data, malicious tools, and emerging attack strategies. Organizations sometimes lack visibility into these hidden channels, limiting their preparedness against zero-day threats and ongoing campaigns. This project handle growing need for exploration tools such as ethical, passive, and also cost effective dark web. By offering an open-source solution with anonymized access, secure storage, and real-time NLP analytics, our system would make researchers and analysts able to gain practical view with preserving privacy and legality.

III. Background and Related Work

Monitoring the dark web has become an essential practice in modern cybersecurity operations due to the increasing amount of sensitive data and malicious tools which are shared on underground forums and illicit marketplaces. Organizations such as Recorded Future and DarkOwl provide commercial platforms for monitoring the dark web threat but these kind of tools are mostly expensive and lack transparency. Academic projects, such as DarkNetCrawler and OnionScan, have shown the possibility of open-source crawling tools but they do not offer modularity, secure data export, and real time user interfaces.

Our project fills this gap by using Tor-based passive crawling with a modern GUI, secure local encryption, and NLP analysis. Compared to previous tools, our system offers privacy, user control, and lightweight deployment by containerization. This system matches with recent developments in open-source intelligence (OSINT) and passive reconnaissance techniques which is applied in threat intelligence collection.

IV. Literature Review

Several studies explored automated techniques for threat detection in dark web ecosystems. Portnoff et al. [2] analyzed marketplace metadata to identify illegal product listings, while Biryukov et al. [3] explored de-anonymization vulnerabilities in Tor network.

More recent project by Ahmed et al. [4] proposed hybrid frameworks that combine natural language processing with dark web crawlers for semantic analysis of threat. Also, Singh and Atrey [5] developed an open-source platform for cyber threat indicators classification from forum text by using keyword matching and basic NLP techniques.

Study	Year	Contribution & Relevance
Portnoff et al. [2]	2017	Analyzed dark web marketplaces to extract CTI metadata
Biryukov et al. [3]	2014	Studied anonymity vulnerabilities in Tor; motivated Tor-safe crawling
Ahmed et al. [4]	2020	Used NLP in dark web crawling; inspired token analysis module
Singh & Atrey [5]	2021	Focused on CTI extraction from forum text using keyword matching and NLP
This Work	2025	Combines Tor crawling, regex matching, NLP, encrypted export, and GUI

TABLE I. Summary of Key Literature on Dark Web Threat Intelligence

V. System Architecture

Figure 1 shows the architecture of Passive Dark Web Crawler system. The pipeline integrates user interaction, crawling with the Tor network, NLP-based keyword analysis, encryption, and controlling data export.

A. User Interface Layer

Implemented using Streamlit [7], this layer enables the user to insert `.onion` site URLs and keywords as input. It also handles user authentication for administrative features, including a session timeout to imrove security.

B. Authentication and Access Control

The admin login system uses SHA-256 password hashing [8] for verification. Authenticated sessions expire after 10 minutes automatically of inactivity. Only authenticated admins can download sensitive decrypted data or access stored encryption keys, while all users can download the encrypted CSV.

C. Crawling Engine

The crawling component operates on the Tor proxy (SOCKS5 on port 9050) to retrieve hidden service pages. The engine uses the `requests` library which is configured with Tor routing to access .onion resources.

D. Processing and NLP Analysis

when the data is retrieved, the system applies BeautifulSoup for HTML parsing and NLTK for tokenization. Keyword matching is done using regex-based search, and token statistics are generated for visualization with bar charts and word clouds.

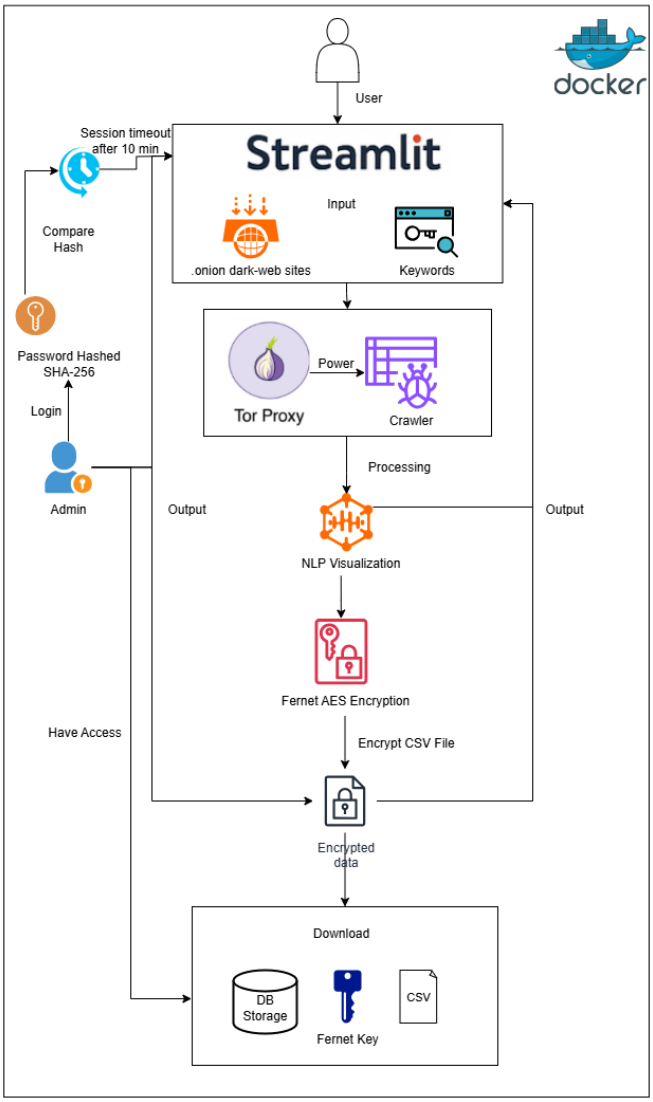

Fig. 1. System Architecture of the Passive Dark Web Crawler [6].

E. Encryption Module

Matched results are locally stored in SQLite and exported in a CSV file encrypted using Fernet AES symmetric encryption. The encryption key is generated per session and stored individually for secure retrieve.

F. Data Storage and Export

The encrypted CSV and corresponding encryption key can be downloaded separately. It ensures confidentiality for access to both files which is required to decrypt the contents. Sensitive decrypted data is just accessible to authenticated administrator.

G. Deployment Layer

The application is able be containerized by using Docker for reproducible deployment. All dependencies including Tor, Python libraries, and NLTK resources are installed and configured in the container.

H. Design Rationale

The architecture was intentionally designed to make three key priorities balanced: operational security, modularity and analyst usability. From a security standpoint, routing all traffic with Tor and limiting interactions to static HTML reduces the risk of deanonymization and mitigates exposure to malicious scripts. Modularity is achieved by separating the user interface, crawling engine, NLP analysis, and encryption into distinct layers and it allows to independent updates or replacements without affecting the rest of the system. Finally, the GUI was prioritized to make the tool accessible to cybersecurity analysts without command-line expertise. These design decisions directly handle the Threat Model outlined in Section VII, which ensures that both technical and non-technical users can operate the system in safe manner.

VI. Implementation Steps and Algorithms

This section provides the technical process of building the passive dark web crawler from scratch including its algorithms, data flow, and details of implementation.

A. Tor Routing and Onion Access

All network requests are routed through the Tor network [9] by using a local SOCKS5 proxy on port 9050. It ensures user anonymity and prevents real IP address exposure. The `requests` library [10] is configured to use the Tor proxy.

```
PROXIES = {
    'http': 'socks5h://127.0.0.1:9050',
    'https': 'socks5h://127.0.0.1:9050'
}
response = requests.get(url, proxies=PROXIES,
    ↪ timeout=60)
```

URLs are sanitized before crawling to ensure appropriate formatting and avoid of failures due to inconsistent schema or trailing slashes.

B. Web Scraping and Parsing

Once a page is fetched via Tor, its HTML content is parsed by using BeautifulSoup [11]. visible text is extracted Only, with HTML tags and metadata removed. JavaScript-heavy pages are skipped to maintain speed and security.

```
soup = BeautifulSoup(response.text, 'html.parser')
text = soup.get_text(separator=' ', strip=True)
```

C. Keyword Matching and Context Extraction

The system uses Python's `re` regular expressions [2] for case-insensitive keyword matching in the page text. Each match includes a snippet of surrounding context for analyst review.

```
pattern = re.compile(re.escape(keyword), re.
    ↪ IGNORECASE)
matches = pattern.finditer(text)
for match in matches:
    idx = match.start()
    snippet = text[max(0, idx - 40): idx + 60]
    cursor.execute(
        'INSERT INTO iocs (url, keyword, context)
    ↪ VALUES (?, ?, ?)',
        (url, keyword, snippet)
    )
```

D. Database Encryption and Export

Matched results are stored in SQLite [1] and can be exported to CSV. The exported file is encrypted using Fernet AES symmetric encryption from the `cryptography` library [12]. A unique encryption key is generated for each session.

```
key = Fernet.generate_key()
fernet = Fernet(key)
with open("ioc_results.csv", 'rb') as f:
    encrypted = fernet.encrypt(f.read())
with open("ioc_results.csv.fernet", 'wb') as f:
    f.write(encrypted)
```

Only the encrypted CSV is publicly downloadable, ensuring unauthorized users cannot view plaintext CTI.

E. Admin Authentication and Logout

Admin functions (downloading plaintext DB, exporting keys) require authentication. Passwords are stored as SHA-256 hashes [8], and sessions expire after 10 minutes of inactivity. The logout resets session state and triggers a rerun in Streamlit [7].

```
import hashlib, time

ADMIN_HASH_SHA256 = "5e884898da28047151...b7852b855"
if 'authenticated' not in st.session_state:
    st.session_state.authenticated = False
    st.session_state.login_time = None

password_input = st.text_input("Enter admin password
    ↪ ", type="password")
if hashlib.sha256(password_input.encode()).hexdigest
    ↪ () == ADMIN_HASH_SHA256:
    st.session_state.authenticated = True
    st.session_state.login_time = time.time()

# Session timeout
if st.session_state.authenticated and \
   time.time() - st.session_state.login_time > 600:
    st.session_state.authenticated = False
    st.warning("Session expired. Please log in again
    ↪ .")

# Logout button
if st.session_state.authenticated and st.button("
    ↪ Logout"):
    st.session_state.authenticated = False
    st.rerun()
```

F. NLP Processing and CTI Taxonomy

The system tokenizes context which are extracted by using NLTK [13], removes stopwords, and computes frequency of terms. These are mapped against common CTI categories:

Detected Term	CTI Category
credential, password, login	Identity Leakage
exploit, malware, ransomware	Malware and Payload
leak, dump, breach	Data Compromise
ip, domain, hash	Network Indicators

TABLE II. Example NLP tokens and mapped CTI categories

This mapping enhances context and enables more integration with threat detection engines.

The matched text corpus is processed with NLTK [13] to remove stopwords and punctuation, producing the top 20 non-stopword tokens. The results are visualized using Matplotlib [14] and WordCloud [15].

```
tokens = word_tokenize(text_blob.lower())
filtered = [t for t in tokens if t.isalnum()
            and t not in stopwords.words('english')]
top_tokens = Counter(filtered).most_common(20)
```

The GUI [7] displays:

- Keyword frequency bar chart
- Word cloud of high-frequency terms
- Interactive table of matched IOCs with context

This combination of steps ensures that the crawler collects, secures, and visualizes CTI data while protecting analyst identity and maintaining operational security.

G. Data Schema

The internal SQLite database is designed with simplicity and extensibility in mind. The primary schema is:

```
TABLE iocs (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  url TEXT,
  keyword TEXT,
  context TEXT,
  timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

This allows tracking of each IOC occurrence along with its origin, context, and time.

While the implementation details are discussed in the Implementation Section, here we outline the design rationale and threat mitigation measures in a higher-level security context

VII. Security Considerations

Given the sensitive nature of dark web data, the crawler is designed with multiple safeguards to protect both the user and the collected information. The main security controls are summarized as below:

A. Threat Model and Authentication

The system assumes a passive adversary model, it means the user does not interact with or submit data to any crawled domain. All requests are routed through the Tor network [9] via a local SOCKS5 proxy, preventing source IP leakage. Administrative actions such as downloading decrypted CTI data or encryption keys require authentication using SHA-256 password hashing [8]. Authenticated sessions expire automatically after 10 minutes of inactivity, and a logout button is provided to terminate sessions manually.

B. Data Confidentiality

All collected CTI is stored in SQLite and exported only as an encrypted CSV file. Fernet AES symmetric encryption [12] ensures that even if the file is intercepted, its contents remain unreadable without the session-specific encryption key which is stored separately and accessible only to authenticated administrators.

C. Privacy by Design

The crawler avoids executing JavaScript or interacting with dynamic elements on visited pages, reducing the attack surface and preventing client-side exploits or fingerprinting. Only static HTML parsing is performed using BeautifulSoup, and no POST requests or logins to target sites are made.

D. Network Anonymity

No network requests are sent directly from the host machine's IP address. All HTTP(S) traffic is anonymized through Tor, ensuring user identity remains concealed during the entire crawling process.

VIII. Deployment Guide

A. Manual Run Steps

The following steps summarize how to set up, run, and interact with the dark web crawler:

1) **Install Docker and Run the App:** Clone the GitHub repository and use the Dockerfile provided to build and run the app securely in a containerized environment.
2) **Start Tor Proxy:** Ensure the local Tor proxy is running and listening on port 9050.
3) **Launch Streamlit GUI:** Access the interface via 'localhost:8501' to enter URLs, keywords, and manage the crawling process.
4) **Monitor Crawling and Visualize Results:** Track keyword matches, view snippets, and analyze trends through dynamic visualizations.
5) **Authenticate as Admin (Optional):** Input the password hash to gain access to sensitive actions like exporting the database or Fernet key.
6) **Download Outputs:** Export encrypted CSVs or SQLite database for offline CTI investigation.

B. Docker Deployment Steps

The application can be deployed via Docker [16] for portability and security. The Dockerfile installs Tor, Streamlit, and dependencies, then launches both services simultaneously.

Steps:

1) Build the image: `docker build -t crawler .`
2) Run the container: `docker run -p 8501:8501 crawler`

IX. Evaluation and Use Cases

The crawler was tested against multiple .onion search indexes, forums, and dumpsites. Keywords such as *malware*, *exploit*, *credential*, and *leak* were consistently matched. The system effectively saved and visualized contextual snippets, revealing trends in CTI-related terminology. This tool can assist:

- SOC analysts looking to extract dark web IOCs
- Cybersecurity students conducting passive recon experiments
- Researchers conducting NLP-based CTI text mining

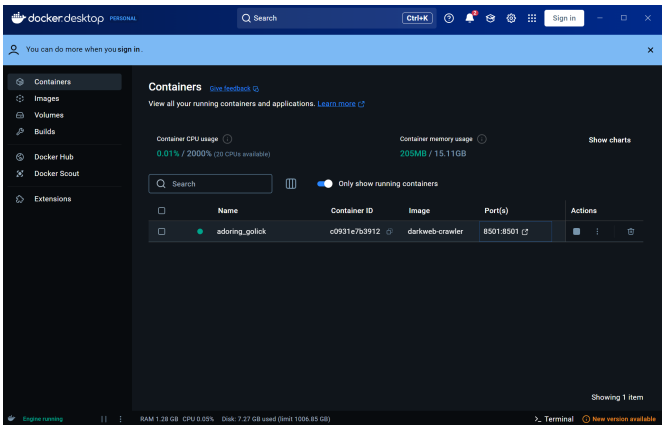

Fig. 2. Docker deployment interface screenshot

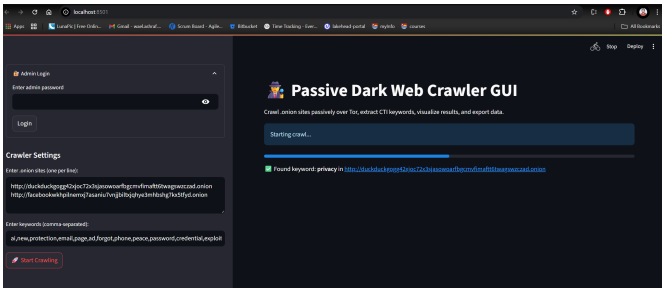

Fig. 3. Streamlit GUI with .onion URL input and keyword field.

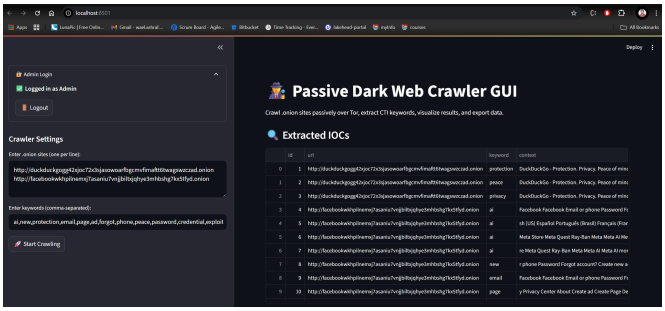

Fig. 4. After Crawling

Fig. 5. Bar Chart keyword frequency snippet

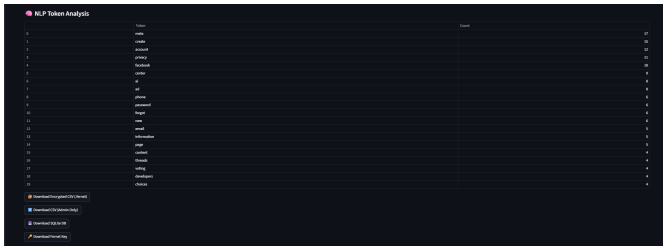

Fig. 6. NLP analysis with Admin login for secured export and key download.

A. *Performance Metrics*

During the testing, the crawler achieved an average crawl time of approximately 4.2 seconds per static page over Tor, with a mean memory footprint of 92 MB during operation. Keyword match accuracy, measured as the proportion of correct matches out of total keyword detections, averaged 96% across test datasets. These results demonstrate that the tool is both lightweight and accurate for passive CTI collection within the limitations of the Tor network.

X. Ethical Considerations

- **Legal Scope:** The crawler is strictly passive. It does not authenticate, log in, or interact with any dynamic content, reducing legal risk.
- **Data Handling:** All collected content is stored locally and encrypted. No cloud synchronization or third-party export is performed.
- **Research Use Only:** The tool is intended for academic use. Explicit warnings and consent banners are included in the GUI to ensure informed usage.

These ethical safeguards complement the technical controls described in the Threat Model (Section VII), ensuring that the crawler's use remains both secure and ethically defensible.

XI. Limitations

While the current implementation offers a strong baseline for passive threat intelligence collection, it also comes with inherent limitations:

- **Static Content Only:** The crawler does not support JavaScript rendering, and thus cannot extract information from sites that rely on client-side scripts for content loading.
- **Limited Scalability:** Crawling over Tor is inherently slower due to relay hops and connection delays, which limits the throughput of the system and makes it unsuitable for large-scale data collection.
- **False Positives:** Keyword-only matching can lead to false positives, as there is no semantic understanding or contextual validation of matched terms.
- **Session Management:** Due to Streamlit's session model, some user actions such as downloading files or refreshing the GUI may cause loss of state or authentication context.

XII. Future Work

To enhance the crawler's capabilities and usability, several improvements are considered:

- **Named Entity Recognition (NER):** Incorporating NER models will allow the system to extract and tag more meaningful indicators of compromise (e.g., IP addresses, hashes, domains) beyond keyword matching.
- **Encrypted Database Storage:** Migration to SQLCipher or similar technologies would provide full-database encryption, protecting all stored entries even at rest.
- **Scheduled Crawling and Deltas:** Adding support for automated, time-based crawling and differential exports would enable historical tracking of CTI evolution.
- **Integration with Dashboards:** Exporting data to platforms like ElasticSearch or Kibana could facilitate advanced visualization, alerting, and correlation with external threat intelligence feeds.

XIII. Source Code Repository

The full source code for this project, including the Dockerfile, Streamlit app, and scripts for crawling and encryption, is publicly available at:

- **GitHub Repository:** https://github.com/waelfahmy/darkweb-cti-crawler

This repository also contains instructions for local setup, environment configuration, dependency installation, and sample keyword lists for testing the crawler in a sandboxed environment.

XIV. Conclusion

We present a passive dark web crawler optimized for cyber threat intelligence collection. It combines secure storage, real-time visualization, NLP-driven analysis, and admin controls into a deployable tool usable by analysts and researchers. With ethical safeguards and Dockerized deployment, this system offers a safe and reproducible method for dark web reconnaissance.

References

[1] "Sqlite documentation," https://sqlite.org, accessed: 2025-08-07.

[2] R. S. Portnoff, V. Varshney, N. Christin, and A. Mislove, "Backpage and beyond: Detecting and characterizing online sex trafficking," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2017.

[3] A. Biryukov, I. Pustogarov, and R. Weinmann, "Trawling for tor hidden services: Detection, measurement, deanonymization," in *IEEE Symposium on Security and Privacy*, 2014.

[4] M. Ahmed, S. Litchfield, and J. Atkinson, "An nlp-based framework for detecting cyber threat intelligence in darknet marketplaces," *International Journal of Cybersecurity Intelligence and Cybercrime*, vol. 3, no. 1, 2020.

[5] A. Singh and P. Atrey, "Cyber threat intelligence extraction from dark web forums using natural language processing," in *Proceedings of the International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 2021.

[6] "diagrams.net (draw.io)," https://www.diagrams.net/, accessed: 2025-08-07.

[7] "Streamlit documentation," https://docs.streamlit.io, accessed: 2025-08-07.

[8] "Python hashlib," https://docs.python.org/3/library/hashlib.html, accessed: 2025-08-07.

[9] "The tor project," https://www.torproject.org, accessed: 2025-08-07.

[10] "Requests: Http for humans," https://requests.readthedocs.io/en/latest/, accessed: 2025-08-07.

[11] "Beautifulsoup documentation," https://www.crummy.com/software/BeautifulSoup/, accessed: 2025-08-07.

[12] "Cryptography python package," https://cryptography.io, accessed: 2025-08-07.

[13] "Nltk documentation," https://www.nltk.org, accessed: 2025-08-07.

[14] "Matplotlib documentation," https://matplotlib.org/stable/contents.html, accessed: 2025-08-07.

[15] "Wordcloud for python documentation," https://amueller.github.io/word_cloud/, accessed: 2025-08-07.

[16] "Docker documentation," https://docs.docker.com, accessed: 2025-08-07.