**Live Cyber Threat Map Documentation**

Live Cyber Threat Map Documentation

Date: June 04, 2025

**Table of Contents**

[**1. Introduction 5**](#_5224fcnce19k)

[1.1. Project Overview 5](#_fmh4uo994r9c)

[1.2 Project Flow 6](#_uytr1xqyevwc)

[**2. Backend Architecture 7**](#_djfqijg6232z)

[2.1. Technology Stack 7](#_ieaf2tj4in8e)

[2.2. Dependencies 7](#_g8px3g30k3vn)

[2.3. Directory Structure 8](#_bfjyvx0jspk)

[2.4. Data Flow, Processing and Management 9](#_aqbf2ia2mxwi)

[2.4.1. Threat Data 9](#_p953cfnqhj26)

[2.4.1.1. Collection 9](#_1d130oze2qgf)

[1. FortiGuard Outbreak API 9](#_583tm2wz7jn3)

[2. Radware Attacks API 12](#_yy1q3pbp6t9z)

[2.4.1.2. Aggregation 13](#_658gkit1mezg)

[2.4.1.3. Processing 14](#_b2ut9wxm6vj1)

[1. Deduplication 14](#_q986v6gw84ay)

[2. Grouping 14](#_mncxihvabfsm)

[2.4.1.4. Output & Queuing 14](#_7qw2nbutnhoy)

[2.4.1.5. Optimization Steps 15](#_4rnbawfmqfwv)

[2.4.2 News Data 15](#_p2f5oxv068bn)

[2.4.2.1. Collection 15](#_4ete3w41kx9t)

[1. Source Configuration 15](#_lhyupedn1l10)

[2. Data Ingestion 15](#_whrp8i6cye02)

[3. Error Handling 15](#_u928s7bwmpyb)

[2.4.2.2. Processing 16](#_s0x90u2doa03)

[1. Keyword Filtering Logic 16](#_yne3nc7m27ly)

[2. Filtering Rules 16](#_20n2bkimy0tm)

[2.4.2.3. Output 16](#_glb10tlylxe8)

[2.4.3. Malicious IP Data Sources 17](#_opoggaa4fgbo)

[2.4.3.1. Collection 17](#_fm9rkcc7roox)

[1. Source Configuration 17](#_gst5raj5kl9a)

[2. Data Ingestion 17](#_t6on1txt33z4)

[3. Error Handling 18](#_2h65kef6j46)

[4. Normalization 18](#_j5xuhrihl52v)

[2.4.3.2. Processing 18](#_nhxlj94csag9)

[1. IP Validation 18](#_eidtyauonrv)

[2. Geolocation Enhancement 18](#_xcnt3rwliiyg)

[3. Threat Type Classification 19](#_ut8z6gzcsrq6)

[4. Deduplication 19](#_k04lclyp9pdn)

[2.4.3.3. Output 19](#_73hazp3fd7e)

[2.5 Flask Implementation 20](#_uevlwfqcxre)

[2.5.1. Endpoints 20](#_hpduowqz6n5s)

[2.5.1.1. Threat Data Stream (/threats) 20](#_x7lhcbnx2mp)

[Features 20](#_bmqeucpt8qs3)

[2.5.1.2. News Data (/news) 21](#_lrsfadzhbv1u)

[Features 21](#_jx37e7q5jnab)

[2.5.1.3. Malicious IPs (/malicious-ips) 21](#_xdc4vy51vklr)

[Features 21](#_l35v2nwbqj81)

[2.5.2. Data Handling 22](#_uh13dhl66j9j)

[2.5.3. Asynchronous Data Collection 22](#_7h4yrbrhgn91)

[2.5.3.1. Batching Mechanism: 22](#_exyfahqowwhj)

[2.5.3.2. Error Handling: 23](#_jjs5wepr27b)

[2.5.4. Configuration Features 23](#_6ihei8cmto95)

[2.5.4.1. CORS Setup: 23](#_tlp2102zwscm)

[2.5.4.2. Event Loop Management: 24](#_uqkodeqdwhjr)

[**3. Frontend Architecture 25**](#_z439rdorfogm)

[3.1. Technology Stack 25](#_sueku7dtm3x1)

[3.2 Frontend Directory Structure 25](#_qm01pi67fe7r)

[3.3 Dependencies 28](#_kpj4udhskqe4)

[3.3.1. Core Dependencies 28](#_z9z1w6h7wiry)

[3.3.2. Mapping and Visualization 28](#_dezm7qicvv8u)

[3.3.3. Animation and UI 29](#_iefy3vbcne58)

[3.3.4. Development Tools 29](#_jkymmmxagw21)

[3.3.5. Type Definitions 29](#_etqthdnnmxtt)

[3.4. Component Architecture 29](#_tnow8b8g6j0g)

[3.4.1. Core Components Section 29](#_bow4nam4k5ef)

[3.4.1.1. WorldMap Component 29](#_n2zmhcv2bmdh)

[3.4.1.2. AttackVector Component 30](#_l4ed4ogckl4c)

[3.4.1.3. Stats Component 30](#_117imem0x3b7)

[3.4.1.4. News Component 30](#_cpom9zrluzx3)

[3.4.2. Components Communication 31](#_6108hgo2q0zj)

[3.4.3. Error Handling Section 32](#_wuy9c9390h5t)

[3.4.3.1. Component-Level: 32](#_ge1tegwff75d)

[3.4.3.2. Data-Level: 32](#_4njvbo6hcfqg)

[3.5. Data Fetching and Handling 32](#_uhbvu9dpwr2f)

[3.5.1. Threat Data Endpoint (/threats) Section 32](#_c6koj0cep1n)

[3.5.2. News Data Endpoint Section 33](#_pdivjt9pzlxc)

[3.5.3. Malicious IPs Endpoint (/malicious-ips) Section 34](#_b7o3ys53wjck)

[3.6. Attack Vector and Malicious IP Mapping 35](#_kvib1qbzac4f)

[3.6.1. Attack Vector Mapping Section 35](#_u9bi9froebhn)

[3.6.1.1. Path Calculation 36](#_hq92gvy8uehb)

[3.6.1.2. Visual Representation 36](#_2dr069ng33pj)

[3.6.1.3. Animation Sequence 36](#_j0gxh7nah8gl)

[3.6.2. Malicious IP Mapping Section 36](#_ei74p7arhqi2)

[3.6.2.1. Marker Rendering: 36](#_r7xzdrkkvcho)

[3.6.2.2. Performance Optimizations: 36](#_6gbu0uk1a431)

[3.6.3. Map Interaction Section 37](#_h0aqwik8rpmn)

[3.6.3.1. Country Hover: 37](#_wvxw9shfjbs)

[3.6.3.2. Map Controls: 37](#_kancr3he4ke2)

[3.6.4. Visual Styling Section 37](#_pzx8ddmn8nyh)

[3.6.4.1. Base Map 37](#_ardvjmram1pl)

[3.6.4.2. Interactive Elements: 37](#_w3s7lye5i6ot)

[3.6.5. Error Handling Section 37](#_qb7nymj9cgq)

[3.6.5.1. Path Validation: 37](#_4t7lhlk141eq)

[3.6.5.2. Rendering Safeguards: 37](#_wesla95ycdvu)

[**4. Usage Instructions 39**](#_zig2d3kcm76s)

[4.1. Backend Setup 39](#_rdpkls66lajp)

[4.1.1 Prerequisites 39](#_923nze4raszq)

[4.1.2 Installation 39](#_70x5dbm4824l)

[4.1.3 Running the Server 39](#_5mesxmcu8qpa)

[4.1.4 API Endpoints 39](#_u92hngaf1p06)

[4.2. Frontend Setup 40](#_iiqb6bgzbk5t)

[4.2.1 Prerequisites 40](#_cc3lccqq1nlf)

[4.2.2 Installation 40](#_22jw5htod5jd)

[4.2.3 Run Development Server 40](#_56m9pl6t928p)

[**5. Conclusion 41**](#_vdw727jmu001)

[**6. References 41**](#_8g63ctl9vk48)

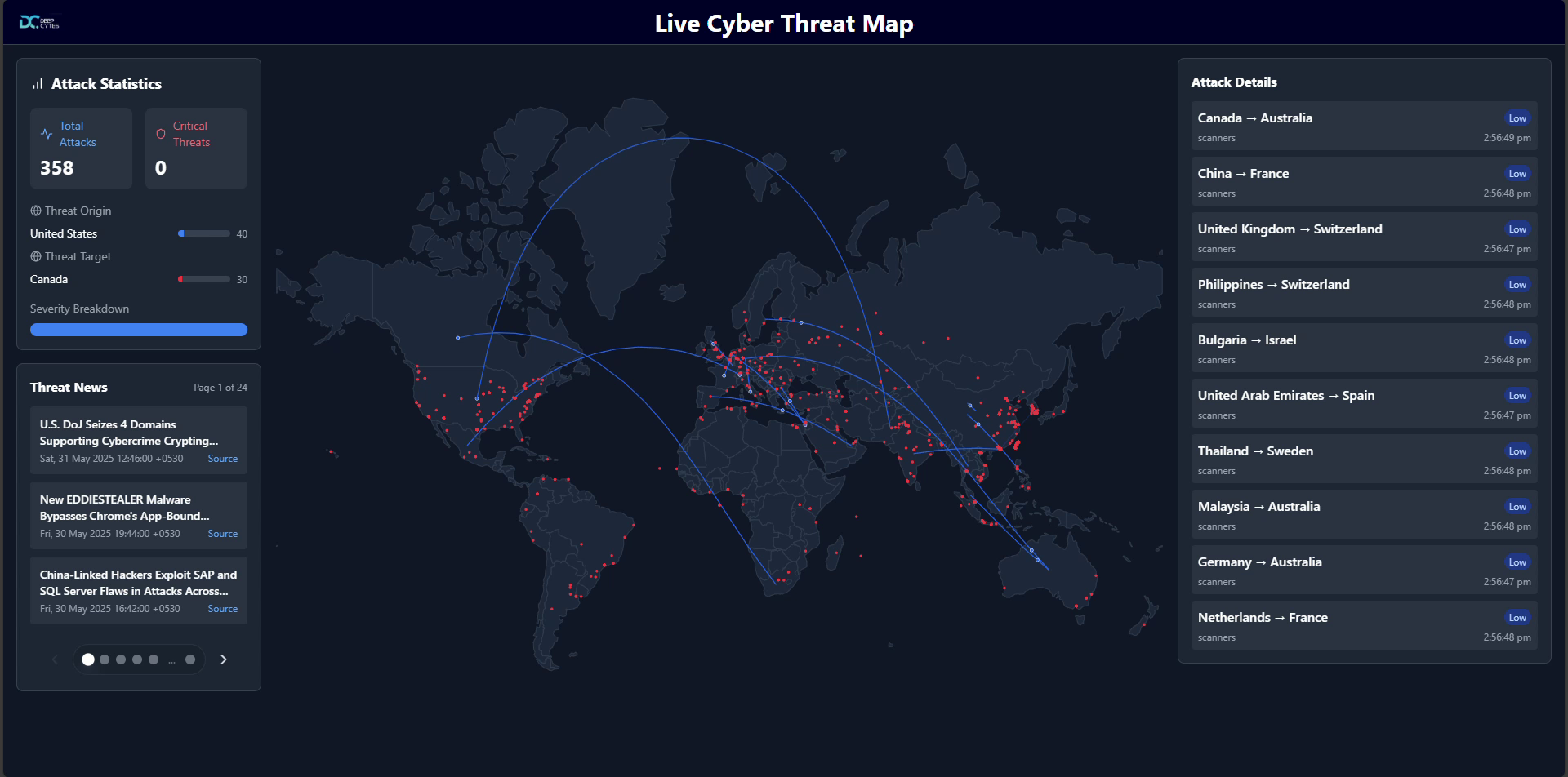
## 

# 

# 1. Introduction

## 1.1. Project Overview

We have developed a Live Cyber Threat Map, an interactive platform that visualizes ongoing and recent cyber threats and attacks across the globe in real time. By aggregating and analyzing data from multiple threat intelligence sources,including attack vectors, malicious IP addresses, and cybersecurity news the map provides a clear and intuitive view of the evolving cyber landscape.  
The primary goal is to enhance situational awareness for security professionals, organizations, educators, and the general public. The map displays attack patterns, geographic origins, and threat types, enabling users to identify trends, emerging risks, and potential vulnerabilities quickly.It serves as a tool for monitoring, analysis, and strategic planning, helping users understand the global threat environment.  
This document is organized into five key sections, providing a logical progression from introduction to conclusion.

Figure 1: Live Cyber Threat Map

## 1.2 Project Flow

### 

Figure 2: Project Flowchart

The system starts by initializing core assets and launching three parallel data collectors. The Threat Data Collector fetches cyber attack information from sources such as FortiGuard and Radware. The Malicious IP Collector gathers suspicious IP addresses from providers including AlienVault, BinaryDefense, FraudGuard, and Talos. The News Data Collector retrieves cybersecurity news from multiple RSS feeds. Each collector operates asynchronously, fetching and validating incoming data. Threat entries are filtered to remove incomplete records, deduplicated, and grouped by source and destination countries with aggregated attack counts and types. Malicious IPs are validated, deduplicated, and enriched with geolocation data using the MaxMind GeoLite2 database, mapping them to standardized country codes and coordinates. News articles are filtered using cybersecurity-specific keywords to extract only relevant content. Once processed, threat data is stored in a dedicated in-memory queue on a First-Come-First-Serve (FCFS) basis, while news articles and malicious-ips are stored in a list. A central **ThreatIntelligenceAggregator** class manages all collectors and coordinates the continuous or on-demand retrieval of data. This workflow enables fast, accurate, and geospatially contextual monitoring of global cyber threats.

# 

# 

# 2. Backend Architecture

## 2.1. Technology Stack

The backend implementation utilizes:

* **Python 3.10+**: The core programming language used for all backend logic
* **Flask**: Lightweight web framework for serving API endpoints

## 2.2. Dependencies

**Live Cyber Threat Map** uses a set of Python dependencies that enable real-time, asynchronous data collection, processing, and delivery.

* **feedparser 6.0.11**:RSS/Atom feed parsing
* **flask 3.0.3**:Web framework
* **aiohttp 3.10.5**: Asynchronous HTTP client/server for making requests to data sources
* **asyncio**: Python's built-in library for writing concurrent code using async/await
* **geoLite2-City.mmdb**:A binary geolocation database from MaxMind used by geoip2 to determine the city, country, and coordinates of IP addresses.
* **geoip2 24.8.0:** Enables IP geolocation by resolving IP addresses to physical locations.
* **pycountry 24.6.1**:Provides ISO standardcountry names and codes, essential for location mapping and standardization.
* **countries\_coordinates.json**:A JSON filecontaining centroid coordinates for all countries, used for mapping IP sources on the threat visualization map.

The architecture follows an event-driven model where data is continuously collected from multiple sources, processed, and made available through API endpoints.

## 

## 2.3. Directory Structure

Backend/

├─ assets/

│ ├─ country\_coordinates.json

│ └─ GeoLite2-City.mmdb

├─ templates/

│ └─ index.html

├─ cyber\_threat\_intel.py

├─ requirements.txt

└─ server.py

**1. assets/**

This folder has important data files that help the backend know where attacks are happening on the map.

* **country\_coordinates.json :** This file has the location details (latitude and longitude) of different countries. It helps the map show where in the world the attacks are happening.
* **GeoLite2-City.mmdb :** This is a database used to find the exact city or area from an IP address. It helps the system figure out where an attack is coming from or going to, so it can be shown on the map.

**2. templates/**

* **index.html:** This is a basic HTML template page that serves as a fallback or testing interface for the API endpoints. It displays data provided by the backend APIs and can be used for development testing. As the server currently runs independently and primarily serves data through REST API endpoints rather than rendered templates, in production, this template can be replaced with a dedicated frontend application.

**3. cyber\_threat\_intel.py**

This Python file collects and processes the cyber attack data.

* It gets data from different online sources (threat data, malicious-ips and news).
* It finds the locations of attacks by using the IP address data.
* It prepares the data so the map can show attacks in the right places.

**4. requirements.txt**

This file lists all the Python tools and libraries needed to run the backend.

* It makes sure anyone who wants to run the backend installs the correct dependencies by running one simple command.

**5. server.py** This is the main program that runs the backend server.

* It serves the application on the specified domain and port.
* It provides cyber attack data through REST API and SSE endpoints to stream real-time updates.
* It connects with cyber\_threat\_intel.py to get the latest threat intelligence.
* It keeps the backend running and handles API requests from clients.
* It exposes data via endpoints like /threats, /news, and /malicious-ips for frontend consumption.

## 2.4. Data Flow, Processing and Management

### 2.4.1. Threat Data

#### 2.4.1.1. Collection

##### FortiGuard Outbreak API

The FortiGuard Outbreak API provides live, global threat intelligence focusing on malware outbreaks and attack events. Data is retrieved from: <https://fortiguard.fortinet.com/api/threatmap/live/outbreak?outbreak_id=0>  
Using the parameteroutbreak\_id=0 ensures that all threat categories are included without filtering.

**Raw Data Format**

The response consists of a nested JSON structure under the ips key. Each entry contains metadata including:

* Attack ID (id)
* Vulnerability type and name (vuln\_type, vuln\_name)
* Severity level
* Source and destination countries and their geographic coordinates
* Attack count and timestamp (or fallback to current UTC time if missing)

**Raw Data:**

{

"ips": {

"1748951935958": [

{

"id": "40772",

"vuln\_type": "ips",

"vuln\_name": "D-Link.Devices.HNAP.SOAPAction-Header.Command.Execution",

"severity": "High",

"dest\_country": "JP",

"dest\_lat": 35.11246,

"dest\_long": 136.20776,

"src\_country": "AR",

"src\_lat": -32.94424,

"src\_long": -60.65054,

"hit": 1

}

]

}

}

 **Transformation and Filtering**

* Raw records are parsed into structured JSON entries.
* Country names and codes are resolved using the pycountry library.
* Entries missing either source or destination country code are discarded.
* Valid records are enriched with standardized fields:
  + Country names, codes
  + Latitude/longitude
  + Attack type and name
  + Timestamp (UTC fallback if missing)
* Duplicates within FortiGuard (same Attack Name, Source Country, Destination Country) are removed.

**Structured Data:**

[

{

"Source Country Code": "AR",

"Source Country Name": "Argentina",

"Source Latitude": -32.94424,

"Source Longitude": -60.65054,

"Destination Country Code": "JP",

"Destination Country Name": "Japan",

"Destination Latitude": 35.11246,

"Destination Longitude": 136.20776,

"Attack Count": 1,

"Attack Types": [

"exploit"

],

"Timestamp": "2025-06-03T14:39:04.450045"

}]

This results in clean, location-aware threat entries ready for aggregation.

##### Radware Attacks API

Radware provides additional global threat activity, often related to scanning or low-weight attack patterns.

**Raw Data Format**  
Radware’s data is structured as a list of lists, with each item containing:

* Source and destination country names
* Attack type
* Attack time
* Optional weight indicators (e.g., "Light", "Heavy")

**Raw Data**

[[{

"sourceCountry": "NL",

"destinationCountry": "SE",

"millisecond": 396,

"type": "scanners",

"weight": "Light",

"attackTime": "2025-06-03T14:56:45.3966667"

}]]

**Transformation and Filtering**

* Entries are flattened and parsed into structured JSON.
* Records lacking source or destination country are discarded.
* Since latitude/longitude values are often missing, the system uses pycountry and a country\_coordinates.json asset to infer geographic centroids.
* All entries are enriched with:
  + Country codes and names
  + Estimated geographic coordinates
  + Attack type and name
  + Timestamp
* Duplicate Radware records are filtered out during preprocessing.

**Structured Data:**

[{

"Source Country Code": "NL",

"Source Country Name": "Netherlands",

"Source Latitude": 52.1326,

"Source Longitude": -74.9528,

"Destination Country Code": "SE",

"Destination Country Name": "Sweden",

"Destination Latitude": 40.0305,

"Destination Longitude": -74.9528,

"Attack Count": 1,

"Attack Types": [

"exploit"

],  
 "Timestamp": "2025-06-03T14:39:04.450045"}]

This ensures Radware data aligns structurally and semantically with FortiGuard data.

#### 2.4.1.2. Aggregation

After individual collection and cleanup, the threat data from FortiGuard and Radware is combined into a single unified dataset. At this stage:

* Entries are merged into one pool.
* Records from different sources are normalized to a consistent format.
* Cross-source duplicates (i.e., same attack name, source country, and destination country across sources) are eliminated.

This aggregation ensures the resulting dataset is comprehensive, reliable, and free of redundancies.

#### 2.4.1.3. Processing

Once the dataset is aggregated, the following steps are applied:

##### Deduplication

Unique threat events are retained based on:

* + Attack Name
  + Source Country Code
  + Destination Country Code  
    Only one instance per unique threat pattern is kept.

##### Grouping

Records are grouped by country pair (e.g., AR → JP).

For each group:

* + The total Attack Count is aggregated.
  + Attack Types are collected as a list (e.g., ["exploit", "scanner"]).
  + Source/destination names and coordinates are preserved.
  + A single representative timestamp is retained.

#### 2.4.1.4. Output & Queuing

The final structured entries include:

* Source Country Code, Source Country Name, Source Latitude, Source Longitude
* Destination Country Code, Destination Country Name, Destination Latitude, Destination Longitude
* Attack Count
* Attack Types (list)
* Timestamp

These cleaned and grouped entries represent distinct, aggregated attack paths and are designed for direct use in real-time threat map visualizations. By consolidating attack flows between countries and attaching summarized metadata, the system allows users to quickly grasp high-level threat activity patterns.

Once processed, each structured record is appended to a central queue for downstream consumption, such as visualization engines, alerting systems, or long-term storage modules. This queue-based architecture decouples data processing from visualization and delivery, enabling real-time responsiveness and modular system design.

The data is collected every 10 seconds, during which, based on observed testing data, it processes and stores nearly 150 to 170 threat entries per cycle.

#### 2.4.1.5. Optimization Steps

* Concurrent fetching minimizes latency during data retrieval from multiple sources and ensures that collection from other sources is not blocked.
* Filtering ensures only complete entries (with both source and destination) are processed.
* Grouping reduces redundancy and provides metrics on attack volume.
* Standardized output format ensures consistent API responses.

### 

### 2.4.2 News Data

#### 2.4.2.1. Collection

##### Source Configuration

The NewsDataCollector class asynchronously retrieves cybersecurity-related news articles from multiple RSS feeds. The default sources include:

* **The Hacker News**:<https://feeds.feedburner.com/TheHackersNews>
* **Dark Reading**:<https://www.darkreading.com/rss.xml>
* **The 420 Cybersecurity Feed**: <https://the420.in/feed>

These feeds are fetched using the feedparser library with a maximum of 3 retry attempts per source. The collector operates in batch mode (i.e., no delay or periodic streaming), collecting fresh entries on each invocation.

##### Data Ingestion

Each RSS feed entry is parsed, and the following fields are extracted:

* **Title**: Must be present
* **Summary/Description**: If summary field is missing, falls back to the description field
* **Link**: Must be present
* **Published Timestamp**: Defaults to the current UTC time (ISO 8601 format) if unavailable

Entries that are missing **either a title or link** are automatically discarded. All string fields are trimmed for whitespace consistency.

##### Error Handling

Feed parsing failures or HTTP/network issues are logged but do not interrupt the overall collection process. The system allows for partial success, meaning articles from other functional sources are still collected even if one feed fails.

#### 2.4.2.2. Processing

##### Keyword Filtering Logic

To ensure relevance, a three-tier filtering mechanism is applied to each article by scanning the combined text of its title and summary. Matching is case-insensitive and uses word boundary regex to avoid partial word matches.

* **Primary Keywords** (must match at least one):   
  ransomware, malware, exploit, vulnerability, breach, zero-day, attack, compromised, infected, stolen, hacked, leak, backdoor, trojan, rootkit, spyware, security, cyber
* **Secondary Keywords** (used to enhance confidence but not required):  
  cybercrime, phishing, ddos, apt, hacking, credential, cyberattack, databreach, hack, payload, threat, botnet, mitigation, critical, authentication, attacker, command and control, lateral movement, exfiltration, intrusion, security flaw
* **Exclusion Terms** (any match causes discard):  
  webinar, workshop, training, course, certification, conference, roundtable, partner, sponsored, promotion, discount, offer, register now, sign up, earn, sale, subscription, tutorial, guide, how to, introduction to

##### Filtering Rules

* + **Immediate Discard**: If an article matches **any exclusion term**, it is excluded without further evaluation.
  + **Relevance Check**: An article must match at least **one primary keyword** to be retained.
  + **Secondary Keyword Matches** are recorded internally but do not affect inclusion directly.

#### 2.4.2.3. Output

The final output is a **clean list of relevant articles**, where each article is a dictionary containing three standardized fields:

* "title" – Cleaned title of the article
* "link" – Direct URL to the article
* "timestamp" – ISO 8601 formatted publication date (or fallback UTC time)

Matched keywords are identified during filtering but **not included** in the output.

**Output Format**

[{ "title": "Android Trojan Crocodilus Now Active in 8 Countries, Targeting Banks and Crypto Wallets",

"link": "https://thehackernews.com/2025/06/android-trojan-crocodilus-now-active-in.html",

"timestamp": "Tue, 03 Jun 2025 15:04:00 +0530"}]

This concise structure ensures the results are lightweight and display-ready for dashboards or downstream processing.

### 2.4.3. Malicious IP Data Sources

#### 2.4.3.1. Collection

##### Source Configuration

The **MaliciousIPCollector** class asynchronously retrieves threat intelligence from multiple cybersecurity feeds. The supported sources include:

* **AlienVault OTX Reputation Feed:** <https://reputation.alienvault.com/reputation.unix>
* **Binary Defense Banlist:** <https://www.binarydefense.com/banlist.txt>
* **FraudGuard Threat Intelligence:** <https://api.fraudguard.io/landing-page-map>
* **Cisco Talos Intelligence:** <https://talosintelligence.com/cloud_intel/top_senders_list>

Each source is fetched using a maximum of **5 retry attempts**. The collector operates in a configurable mode, supporting both **batch collection** (interval = 0) and **continuous streaming** (with interval delay).

##### Data Ingestion

Each feed is parsed according to its specific format, with the following common steps:

* **IP Extraction**: All raw data is scanned for IP addresses using the regex pattern: “\b(?:\d{1,3}\.){3}\d{1,3}\b”
* **Feed-specific Parsing**:
  + AlienVault and Binary Defense: raw text lists
  + FraudGuard: embedded JSON threat data
  + Talos: structured spam-related JSON
* **Validation**:
  + Extracted strings must pass IPv4 address validation using Python’s built-in **ipaddress** module.
  + Invalid or malformed entries are automatically discarded.

##### Error Handling

Network failures, timeouts, and data parsing issues are logged per source without interrupting the overall process. A failure in one feed does **not block** others. The GeoLite2 geolocation lookup also gracefully degrades when lookups fail (e.g., missing coordinates).

##### Normalization

* + IPs are converted to a consistent structure with the following:
    - **IP address** (string)
    - **Latitude/Longitude** (nullable float)
    - **Type** (either **"malicious"** or **"spam"** based on source)
  + IPs are **deduplicated** and **sorted** to ensure a clean and consistent output set.

#### 2.4.3.2. Processing

##### 1. IP Validation

All extracted strings undergo strict validation:

* Validity is checked using Python's **ipaddress.ip\_address()** to ensure they are legitimate IPv4 addresses.
* Non-IP content (e.g., strings, malformed addresses) are discarded at this stage.

##### 2. Geolocation Enhancement

Each validated IP is enriched with geographic data via the **GeoLite2-City** database:

* **Database Integration**: Uses **GeoLite2-City.mmdb** from MaxMind.
* **Coordinate Extraction**: Latitude and longitude are pulled in decimal degrees.
* **Fallback Handling**: IPs without available geolocation are retained with null **latitude** and **longitude.**

##### 

##### 3. Threat Type Classification

Each IP is tagged with a type based on the originating source:

* "malicious": AlienVault, Binary Defense, FraudGuard
* "spam": Cisco Talos (focused on email spam sources)

##### 4. Deduplication

To avoid redundant entries:

* A unique key is derived from the IP address.
* Only one record per IP is retained in the final dataset.

#### 2.4.3.3. Output

The final output is a **deduplicated list** of structured threat indicators. Each entry contains the following fields:

* "ip" – Validated IP address (string)
* "latitude" – Geographic latitude in decimal degrees (nullable)
* "longitude" – Geographic longitude in decimal degrees (nullable)
* "type" – Classification label: "malicious" or "spam"

**Standardized Format:**

[

{

"ip": "1.161.219.86",

"latitude": 24.9889,

"longitude": 121.3176,

"type": "malicious"

},]



## 2.5 Flask Implementation

**Flask server** serves as the interface between the data collection system and frontend clients, providing:

* **RESTful API Endpoints** for accessing threat intelligence data
* **Server-Sent Events (SSE)** for real-time threat data streaming
* **CORS Support** to enable cross-origin requests from web applications
* **Data Transformation** between internal Python structures and JSON responses
* **Error Handling and logging** for operational visibility

The server is built with production-ready logging (e.g., logging.INFO) and runs asynchronously using **asyncio** and **aiohttp** for non-blocking data operations.

### 2.5.1. Endpoints

#### 2.5.1.1. Threat Data Stream (/threats)

The **/threats** endpoint delivers a real-time stream of threat intelligence using **Server-Sent Events (SSE)**. It is designed for integration with live dashboards, alerting systems, and visual analytics platforms that require low-latency updates.

##### Features

* **Live Data Streaming with SSE:** Utilizes **Server-Sent Events (SSE)** to continuously push live threat data to clients, eliminating the need for repeated polling.
* **Efficient Batching:** Threat entries are collected in bulk and split into smaller batches before dispatch, reducing bandwidth usage and client-side processing overhead.
* **Batch Dispatch Logic**
  + **Collection Interval**: Every 10 seconds, new threat data is fetched.
  + **Dispatch Frequency**: One batch is sent every second regardless of batch size, for a smooth and consistent stream.
  + **Batches per Cycle**: 10 total batches per 10-second cycle.
  + **Formula**:  
    Items per second in a batch = N / S,   
    where N is number of items collected,  
    S is the interval in seconds (10 by default)
* **Async Streaming Loop:** The endpoint creates a new event loop for each client connection and synchronously streams data from an async generator.
* **Resilient connectivity:** If no data is available, empty arrays are still streamed every second to maintain the connection.

#### 2.5.1.2. News Data (/news)

The **/news** endpoint provides a lightweight, RESTful interface for retrieving curated cybersecurity news articles from trusted sources such as **The Hacker News**, **Dark Reading**, and **The 420 Cybersecurity Feed**. This endpoint is optimized for dashboards, API clients, or periodic polling systems that require relevant, filtered news updates.

##### Features

* **RESTful Access:** Exposes a standard REST API that returns current cybersecurity news articles. Feeds are parsed using the feedparser library, supporting RSS-based sources. Provides a simple HTTP GET endpoint that returns a list of relevant news articles parsed from RSS feeds.
* **Pre-Filtered Articles:** Delivers a pre-filtered list of articles that match predefined primary keyword rules (e.g., "breach", "ransomware", "exploit") while excluding promotional or irrelevant content via regex-based exclusion terms.
* **Robust Error Handling:** Implements detailed logging and appropriate HTTP status codes:  
  + **200 OK**: Articles found and returned successfully
  + **404 Not Found**: No relevant articles matched the filter
  + **500 Internal Server Error**: Source parsing or retrieval failure  
     Error logs include feed source name and failure reason for traceability.
* **Lightweight Design:** Designed for efficiency and quick responses:  
  + Ideal for dashboards or automated polling systems
  + The MaliciousIPCollector fetches new malicious IP data every 60 seconds (1 minute)
  + The **/threats** endpoint streams threat data to clients every second
  + The **/malicious-ips** endpoint fetches data on demand with no periodic interval or caching; each request triggers a fresh fetch

#### 2.5.1.3. Malicious IPs (/malicious-ips)

The **/malicious-ips** endpoint provides an on-demand snapshot of malicious IP addresses by instantiating a fresh collector on each request. It is designed for one-time queries and integration into dashboards or automation systems.

##### Features

* **On-Demand Data Fetching:** A new **MaliciousIPCollector** instance is created for each request. This guarantees that the IP data is fresh and not reused from earlier cycles.
* **Isolated Async Execution:** The server creates a dedicated asyncio event loop for the request using **asyncio.new\_event\_loop()** and runs the fetch pipeline synchronously via **run\_until\_complete()**.
* **Clean Resource Management:** The collector and event loop are explicitly closed in a finally block, ensuring that:
  + **await session.close()** is always called
  + The event loop is properly shut down, this prevents memory leaks or dangling network resources.
* **Simple REST Interface:** Returns a JSON list of threat indicators retrieved from the configured sources. The format and enrichment (e.g., geolocation) are handled internally by the collector logic.
* **Error Handling**
  + Logs all exceptions with context (e.g., network failures, fetch errors)
  + Returns 500 Internal Server Error with a JSON error message on failure
  + Returns 200 OK with an empty list if no IPs were fetched

### 2.5.2. Data Handling

* **Concurrency Model**:
  + Each API request creates a dedicated asyncio event loop for isolated data retrieval (e.g., /malicious-ips initializes a new loop per call).
  + **Exception**: The /threats SSE endpoint shares a global event loop but uses a thread-safe deque for queue management.
* **Data Structures**:
  + **deque**: Used for threat data (stream\_threat\_data()) to support efficient O(1) appends/pops during streaming.
  + **list**: Used for news articles (/news) and malicious IPs (/malicious-ips), where ordering is preserved but FIFO behavior is unnecessary.
* **Data Consistency**:
  + All responses are serialized to JSON with consistent schemas:
    - **Threat/news data**: Array of objects with source, timestamp, and threat-specific fields.
    - **Malicious IPs:** Array of objects with ip, threat\_type, and geolocation.
  + Schema validation occurs during collector preprocessing (e.g., in ThreatIntelligenceAggregator), not in the Flask layer.

### 2.5.3. Asynchronous Data Collection

#### 2.5.3.1. Batching Mechanism:

* **Threat Data**:
  + Batches are dynamically sized (math.ceil(queue\_length / 10)) and emitted every second (await asyncio.sleep(1)).
  + Ensures steady client updates without overwhelming the frontend (e.g., 100 queued items 10-item batches over 10s).
* **News/Malicious IPs**:
  + Fetched on-demand (/news, /malicious-ips) without batching, as they are less time-sensitive.

#### 2.5.3.2. Error Handling:

* **Structured Fallbacks**:
  + Logs errors with context (e.g., logger.error(f"Error fetching news data: {e}")).
  + Returns HTTP 500 with error details (e.g., jsonify({"error": str(e)}), 500) or empty arrays ([]) for non-critical failures.
* **Fault Isolation**:
  + /malicious-ips creates a new MaliciousIPCollector instance per request to avoid session conflicts.
  + Individual collector failures (e.g., RSS feed timeout) don’t crash other endpoints.

### 2.5.4. Configuration Features

#### 2.5.4.1. CORS Setup:

* **Implementation**:   
  Configured via:

CORS(app, resources={

r"/\*": {

"origins": "\*",

"methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],

"allow\_headers": ["Content-Type", "Authorization"]

}

})

to allow all origins (**\***), methods, and headers like **Content-Type**.

* **Purpose**: Enables frontend-backend integration when deployed separately (e.g., frontend on netlify.app, backend on localhost:5000).
* **NOTE:** For **production deployments**, you should **restrict CORS to your actual frontend domain** for security reasons.  
    
  For example, replace "\*" with your deployed frontend domain:

CORS(app, resources={

r"/\*": {

"origins": ["https://your-frontend-domain.com"],

"methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],

"allow\_headers": ["Content-Type", "Authorization"]

}

})

This ensures only your frontend can access the backend API, preventing unauthorized third-party sites from making API requests.

#### 2.5.4.2. Event Loop Management:

* **Initialization**:
  + The global event loop starts at server launch (loop.run\_until\_complete(initialize\_aggregator())).
  + Collectors run in the background with fixed intervals (e.g., ip\_collector.interval = 60.0).
* **Collector Workflow**:
  + **Threats**: Streamed via threat\_collector.stream\_data() → deque → batched SSE.
  + **News**: Fetched on-demand via news\_collector.fetch\_data().
  + **IPs**: Fetched on-demand with fresh collector instances to avoid state conflicts.
* **Resource Cleanup**:
  + Loops and connections are closed in finally blocks (e.g., new\_loop.close(), aggregator.close()).

# 

# 3. Frontend Architecture

The frontend of the Live Cyber Threat Map provides a comprehensive visualization of global cyber attack activity, presenting users with up-to-date information on ongoing and recent threats. It features a central world map displaying attack origins and targets, flanked by panels that summarize attack statistics, detail individual incidents, and present the latest cybersecurity news.This interface enables users to monitor the cyber threat landscape in real time, offering aggregated statistics, severity breakdowns, and curated news updates in a single, cohesive view. The workflow, illustrated in the accompanying flowchart given under introduction, demonstrates how the system continuously collects, processes, and presents threat data, news, and malicious IP information from various sources. By integrating these elements, the frontend delivers a reliable and informative overview of cyber threats and aids in staying informed about the evolving digital risk environment.

## 3.1. Technology Stack

* **React**: JavaScript library for building dynamic UI components.
* **Vite**: Build tool for fast development and bundling.
* **TypeScript**: Adds static typing to JavaScript for improved reliability.
* **Tailwind CSS**: Utility-first CSS framework for styling.

## 3.2 Frontend Directory Structure

Frontend/

├─ src/

│ ├─ components/

│ │ ├─ AttackVector.tsx

│ │ ├─ news.tsx

│ │ ├─ Stats.tsx

│ │ └─ WorldMap.tsx

│ ├─ data/

│ │ ├─ countries.ts

│ │ ├─ maliciousIPs.ts

│ │ └─ threatData.ts

│ ├─ Image/

│ │ └─ logo\_DC.png

│ ├─ types/

│ │ └─ index.ts

│ ├─ workers/

│ │ ├─ newsWorker.ts

│ │ └─ threatWorker.ts

│ ├─ App.tsx

│ ├─ index.css

│ ├─ main.tsx

│ └─ vite-env.d.ts

├─ .gitignore

├─ eslint.config.js

├─ index.html

├─ package-lock.json

├─ package.json

├─ postcss.config.js

├─ README.md

├─ tailwind.config.js

├─ tsconfig.app.json

├─ tsconfig.json

├─ tsconfig.node.json

└─ vite.config.ts

1. **src/**Contains all the main source code that powers the cyber threat visualization interface for the frontend. It includes components, data files, images, and logic that run in the user’s browser.

* **components/**This folder has different parts of the website interface that are shown to the user:
  + **AttackVector.tsx**Displays the types of cyber attacks happening. Helps users understand what kind of threats are active.
  + **news.tsx**Shows the latest cybersecurity news updates to keep users informed about recent threats.
  + **Stats.tsx**Displays numerical data such as total attacks, countries affected, or attack trends.
  + **WorldMap.tsx**Shows a visual world map where attacks are displayed in real-time or based on collected data.
* **data/**This folder contains static data used by the components:
  + **countries.ts**Contains information about countries like names and codes. Helps in mapping and data display.
  + **maliciousIPs.ts**Lists harmful or suspicious IP addresses used for threat detection.
  + **threatData.ts**Stores the structured threat information to be shown on the map or in stats.
* **Image/**This folder holds images used in the frontend.
  + **logo\_DC.png**This is the logo image displayed on the website.
* **types/**This folder defines TypeScript types to ensure data is used consistently and safely across the code.
  + **index.ts**Contains data types and interfaces for threat data, news, IP info, etc.
* **workers/**Web workers that run background tasks without affecting the main interface performance.
  + **newsWorker.ts**Handles background tasks to fetch and update news data regularly.
  + **threatWorker.ts**Fetches and processes cyber threat data without blocking the main app.
* **App.tsx**The main component that puts together all other components and displays the full interface.
* **index.css**This file contains the main styles for the frontend, including Tailwind CSS styles.
* **main.tsx**The entry point of the frontend. It starts the app by rendering App.tsx to the browser.
* **vite-env.d.ts**A TypeScript file that helps Vite (the build tool) work with environment variables.

1. **.gitignore** This file tells Git which files or folders to ignore when pushing code to version control. For example, it ignores node\_modules or build files.
2. **eslint.config.js** Configures ESLint, a tool that checks the code for errors and formatting issues.
3. **index.html** This is the HTML template used by Vite to load the React app into the browser.
4. **package-lock.json** Automatically generated file that locks the exact versions of installed packages. Helps maintain consistency when installing dependencies.
5. **package.json** Lists the dependencies (libraries used), project info, and scripts for running or building the frontend.
6. **postcss.config.js** Configures PostCSS, a tool used with Tailwind CSS to process styles.
7. **README.md**A markdown file that contains information about how to set up, run, and understand the project.
8. **tailwind.config.js** Customizes Tailwind CSS styles such as colors, spacing, and font settings.
9. **tsconfig.app.json**, **tsconfig.json**, **tsconfig.node.json** These files set up TypeScript settings for the app, development environment, and node-related parts of the code.
10. **vite.config.ts** Configuration file for Vite. It tells Vite how to build and serve the React app during development and production.

## 

## 3.3 Dependencies

### 3.3.1. Core Dependencies

* **React (v18.3.1)** - The primary UI library used to build the interactive components and manage the application state
* **React DOM (v18.3.1)** - Provides DOM-specific methods for React applications
* **TypeScript (v5.5.3)** - Adds static typing to JavaScript, improving code reliability and developer experience

### 3.3.2. Mapping and Visualization

* **react-simple-maps (v3.0.0)** - Provides the base world map component with country boundaries and projection handling
* **d3 (v7.9.0)** - Core data visualization library used for advanced mapping features
* **d3-geo (v3.1.1)** - Geographic projections and transformations for the world map
* **d3-geo-projection (v4.0.0)** - Additional map projections and geographic calculations
* **d3-shape (v3.2.0)** - Shape generators for attack vectors and markers
* **topojson-client (v3.1.0)** - Handles TopoJSON data for efficient map rendering

### 3.3.3. Animation and UI

* **GSAP (v3.13.0)** - Professional-grade animation library used for smooth attack vector animations and transitions
* **lucide-react (v0.344.0)** - Icon library providing consistent and customizable icons
* **Tailwind CSS (v3.4.1)** - Utility-first CSS framework for rapid UI development and consistent styling

### 3.3.4. Development Tools

* **Vite (v5.4.2)** - Next-generation frontend build tool for fast development and optimized production builds
* **ESLint (v9.9.1)** - Code linting tool for maintaining code quality
* **PostCSS (v8.4.35)** - CSS processing tool used with Tailwind
* **Autoprefixer (v10.4.18)** - Automatically adds vendor prefixes to CSS

### 3.3.5. Type Definitions

* **@types/react** - TypeScript definitions for React
* **@types/react-dom** - TypeScript definitions for React DOM
* **@types/d3** - TypeScript definitions for D3.js
* **@types/react-simple-maps** - TypeScript definitions for react-simple-maps
* **@types/topojson-client** - TypeScript definitions for topojson-client
* **@types/uuid** - TypeScript definitions for UUID generation

## 3.4. Component Architecture

The frontend is built with a modular component architecture that focuses on real-time visualization and data presentation. Each component is designed to handle specific aspects of the cyber threat map interface.

### 3.4.1. Core Components Section

#### 

#### 3.4.1.1. WorldMap Component

Primary visualization component using react-simple-maps

* **Features:**
  + Interactive world map with country hover detection
  + Real-time attack vector rendering
  + Malicious IP marker placement
  + Zoom and pan controls
  + Custom styling with dark theme
* **Props:**
  + width: number - Map container width
  + height: number - Map container height
  + onCountryHover: (country: Country | null) => void
  + maliciousIPs: MaliciousIP[]
  + attacks: Attack[]

#### 3.4.1.2. AttackVector Component

Handles individual attack visualization

* **Features:**
  + GSAP-powered animations
  + Path calculation with date line handling
  + Severity-based color coding
  + Animated marker movement
* **Props:**
  + attack: Attack - Attack data to visualize

#### 3.4.1.3. Stats Component

Displays real-time attack statistics

* **Features:**
  + Total attack counter
  + Critical threats counter
  + Top source/target countries
  + Severity breakdown visualization
* **Props:**
  + attacks: Attack[] - Array of current attacks

#### 

#### 3.4.1.4. News Component

Shows cybersecurity ne.ws with pagination

* **Features:**
  + Flip animation for news items
  + Pagination with smart truncation
  + SSE support for real-time updates
  + Fallback to dummy data
* **Props:**
  + useSSE?: boolean - Enable Server-Sent Events
  + itemsPerPage?: number - Items per page

### 3.4.2. Components Communication

Data Flow:

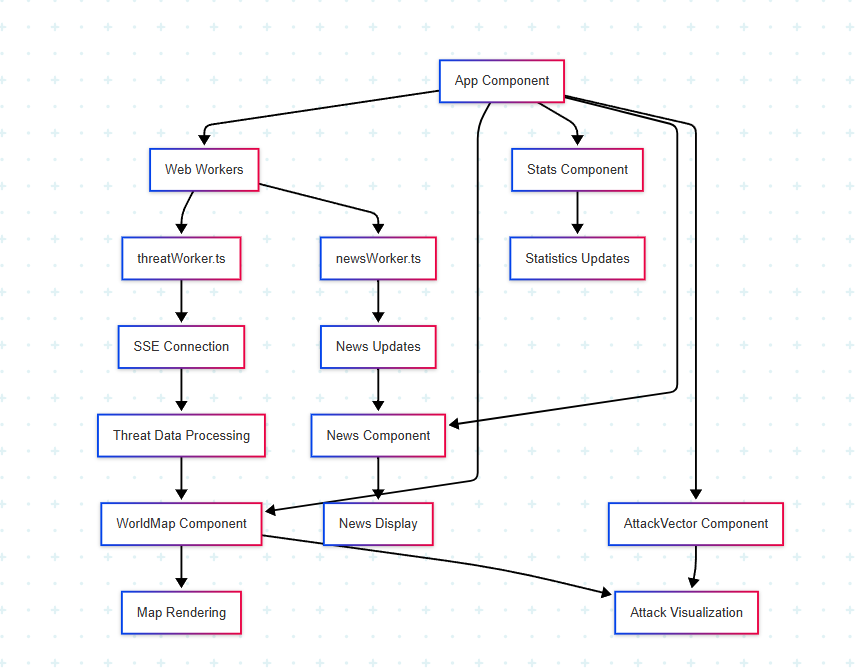


Figure 3: Data flow diagram

* **Components update dynamically based on incoming data streams:**
  + **Threats:** threatWorker.ts receives threat data via SSE every second. New attacks are validated, typed, and appended to the map with animations. Expired entries are removed to reduce clutter.
  + **News:** Fetched once on load with retry logic. The News component updates with smooth transitions and paginated display.
  + **Malicious IPs:** Fetched similarly to news. Markers are rendered on the map using normalized coordinates.
* **Performance Optimizations:**
  + Uses React.memo and memoized props to avoid re-renders.
  + Coordinates and entries are filtered for validity.
  + Cleanup logic ensures no memory leaks on unmount.
* **Error Handling:**
  + Fallback to dummy data on failure.
  + Logs errors and continues with partial data when needed.

### 3.4.3. Error Handling Section

#### 3.4.3.1. Component-Level:

* Fallback UI for failed data fetches
* Error boundaries for component crashes
* Graceful degradation of features

#### 3.4.3.2. Data-Level:

* Validation of incoming data
* Fallback to dummy data
* Proper error logging
* Retry mechanisms for failed requests

## 3.5. Data Fetching and Handling

### 3.5.1. Threat Data Endpoint (/threats) Section

* The frontend uses Server-Sent Events (SSE) to receive real-time threat data
* A Web Worker (threatWorker.ts) handles the SSE connection to avoid blocking the main thread
* Raw threat data is received in this format:

interface RawSSEThreat {

"Source Country Code": string;

"Source Country Name": string;

"Source Latitude": number | null;

"Source Longitude": number | null;

"Destination Country Code": string;

"Destination Country Name": string;

"Destination Latitude": number | null;

"Destination Longitude": number | null;

"Attack Count": number;

"Attack Types": string[];

Timestamp: string;

}

* The worker converts raw data to the frontend's Attack type with proper typing and validation
* Data is then passed to the main thread where:
  + Attacks are filtered based on severity and valid coordinates
  + Each attack is added to the map with a visual delay for better UX
  + Statistics are updated in real-time
  + The WorldMap component renders the attacks as animated vectors

#### 

### 3.5.2. News Data Endpoint Section

* Regular HTTP GET request mode.
* Raw news data is received in this format:

interface RawSSENews {

title: string;

timestamp: string;

source: string;

}

* In regular mode:
  + Makes a GET request to http://localhost:5000/news
  + Handles errors with fallback to dummy data
* The News component:
  + Implements pagination
  + Shows items with flip animations for transitions
  + Formats timestamps for display
  + Maintains state for current page and displayed items

### 3.5.3. Malicious IPs Endpoint (/malicious-ips) Section

* Uses a regular HTTP GET request with retry mechanism
* The fetchMaliciousIPsWithRetry function:
  + Makes requests to the endpoint
  + Implements exponential backoff for retries
  + Returns an empty array if all retries fail
* Raw Malicious IPs are fetched in the following format:

interface RawMaliciousIP {

ip: string;

latitude: number;

longitude: number;

type: string;

}

* Data is fetched once when the App component mounts
* The WorldMap component:
  + Receives malicious IPs as props
  + Renders them as markers on the map
  + Each IP has its own marker with location data

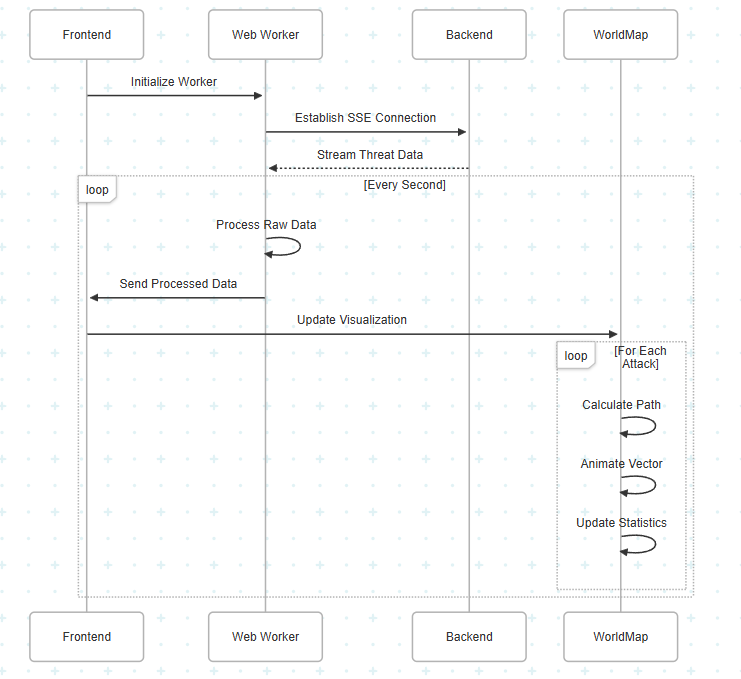


Figure 4: Sequence diagram

### 

## 3.6. Attack Vector and Malicious IP Mapping

The frontend implements sophisticated mapping logic for visualizing cyber attacks and malicious IPs on the world map. This section details how these elements are rendered and animated.

### 3.6.1. Attack Vector Mapping Section

The AttackVector component handles the visualization of cyber attacks between source and target locations:

#### 3.6.1.1. Path Calculation

* Implements a `getDirectPath` function that calculates the optimal path between source and target coordinates
* Handles date line crossing by normalizing longitude values to [-180, 180] range
* Implements strict bounds checking to ensure paths stay within visible map area
* Filters out invalid paths that would curve outside visible bounds

#### 3.6.1.2. Visual Representation

* Uses GSAP (GreenSock Animation Platform) for smooth animations
* Renders two main elements:
* An animated line showing the attack path
* A moving marker (circle) that follows the path
* Color coding based on attack severity:
* Critical: #dc2626 (red)
* High: #ea580c (orange)
* Medium: #ca8a04 (yellow)
* Low: #2563eb (blue)
* Unknown: #9333ea (purple)

#### 3.6.1.3. Animation Sequence

* Initial state: Line is invisible (strokeDashoffset: 100%)
* Marker moves along path (duration: 0.8s)
* Line draws behind marker (duration: 1s)
* Both elements fade out together (duration: 0.3s)
* Total animation duration: 2.1s

### 

### 3.6.2. Malicious IP Mapping Section

The WorldMap component handles the visualization of malicious IP locations:

#### 3.6.2.1. Marker Rendering:

* Uses react-simple-maps Marker component
* Renders each IP as a small red circle (r: 1.7)
* Implements smooth transitions for marker appearance
* Clips markers to visible map area using SVG clipPath

#### 3.6.2.2. Performance Optimizations:

* Memoizes geography data URL to prevent unnecessary re-renders
* Uses React.memo for AttackVector component
* Implements efficient coordinate normalization
* Filters out invalid coordinates before rendering

### 

### 3.6.3. Map Interaction Section

The WorldMap component provides interactive features:

#### 3.6.3.1. Country Hover:

* Implements hover detection for countries
* Shows country information panel with:
* Country name
* Latitude/Longitude coordinates
* Uses smooth transitions for hover effects

#### 3.6.3.2. Map Controls:

* Implements zoomable group for map navigation
* Uses Mercator projection with custom scale and center
* Provides smooth zoom and pan animations
* Maintains aspect ratio and proper scaling

### 3.6.4. Visual Styling Section

The map implements a dark theme with consistent styling:

#### 3.6.4.1. Base Map

* Background: #111827 (dark blue-gray)
* Country fill: #1E293B
* Country stroke: #334155
* Hover state: #2D3B4F

#### 3.6.4.2. Interactive Elements:

* Smooth transitions (250ms duration)
* Hover effects with cursor pointer
* Consistent border radius and shadows
* Backdrop blur for overlays

#### 

### 3.6.5. Error Handling Section

The mapping system implements robust error handling:

#### 3.6.5.1. Path Validation:

* Validates source and target coordinates
* Handles edge cases (date line crossing, invalid coordinates)
* Provides fallback for invalid paths

#### 3.6.5.2. Rendering Safeguards:

* Clips elements to visible area
* Handles null or undefined values
* Implements proper cleanup for animations
* Manages memory efficiently with proper unmounting

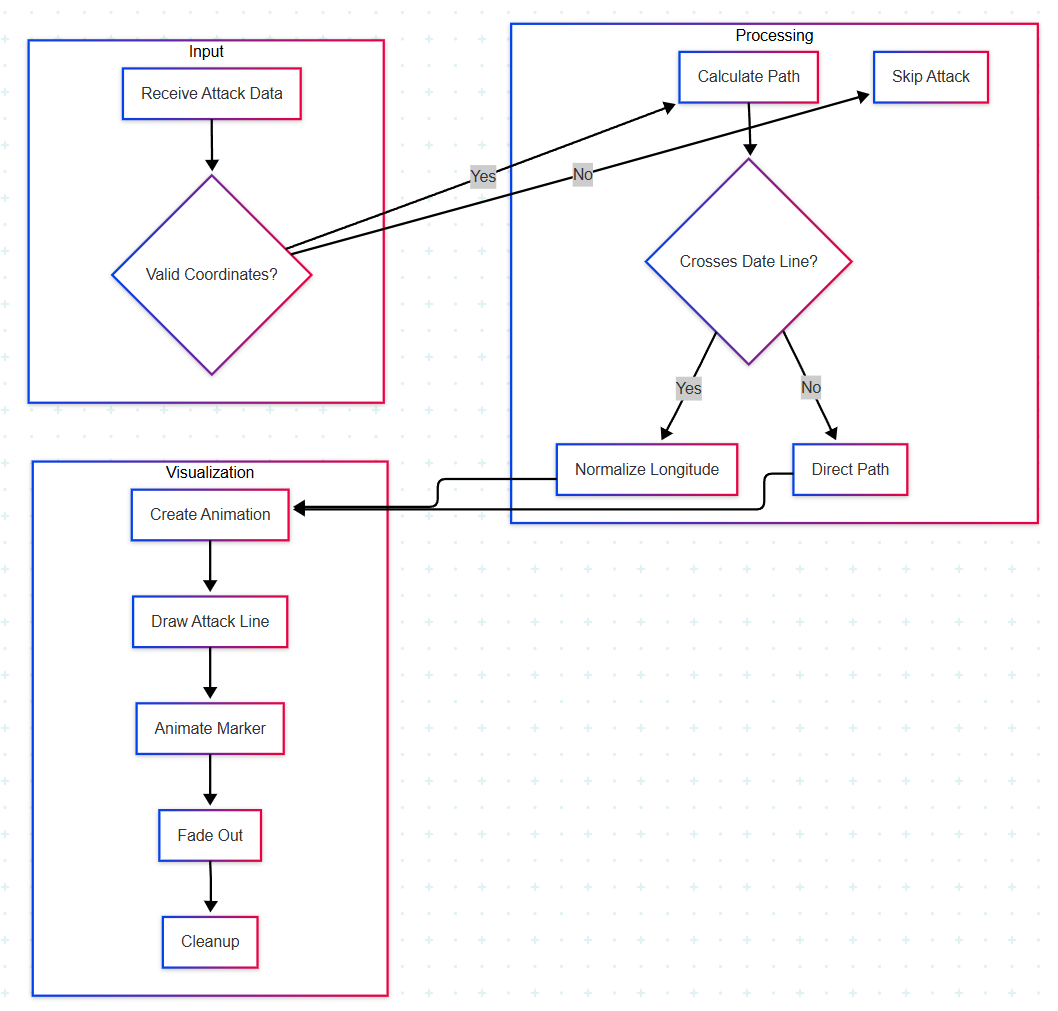


Figure 5: Attack Vectors mapping

# 

# 

# 

# 4. Usage Instructions

This project consists of two components: a backend that delivers real-time threat intelligence via APIs, and a frontend that displays this data on a dashboard.

Please note that the backend and frontend must be started and run separately.

## 4.1. Backend Setup

### 4.1.1 Prerequisites

* Python 3.10+
* Pip

### 4.1.2 Installation

cd DC\_LCTM\_Backend

pip install -r requirements.txt

### 4.1.3 Running the Server

* For development:

python server.py

* For production (recommended):

gunicorn -w 4 -b 0.0.0.0:5000 server:app

### 4.1.4 API Endpoints

* /threats – Live cyber threat data via Server-Sent Events (SSE)
* /news – Filtered cybersecurity news articles (GET)
* /malicious-ips – Geolocated malicious IPs (GET)

## 4.2. Frontend Setup

### 4.2.1 Prerequisites

* Node.js (v14 or higher)
* npm or yarn

### 4.2.2 Installation

cd DC\_LCTM\_Frontend-v2

npm install

# or

yarn install

### 4.2.3 Run Development Server

npm run dev

# or

yarn dev

****

The frontend will be available at **http://localhost:5173** by default.

# 5. Conclusion

The Live Cyber Threat Map is more than just a visualization - it's a window into the constantly shifting landscape of global cyber threats. It helps users make sense of where attacks are coming from, what form they take, and how they evolve over time. Whether it’s a security analyst looking for patterns, an organization monitoring potential risks, or a curious user trying to understand the digital world better, the map delivers clear, timely insights. With its seamless integration of live data, intuitive interface, and focus on situational awareness, it plays a crucial role in helping people stay one step ahead in an increasingly connected and vulnerable world.

# 

# 6. References

* Flask 3.0.3 Documentation:<https://flask.palletsprojects.com/en/3.0.x/>
* AIOHTTP 3.10.5 Documentation:<https://docs.aiohttp.org/en/stable/>
* Feedparser 6.0.11 Documentation:<https://feedparser.readthedocs.io/en/latest/>
* Pycountry 24.6.1 Documentation:<https://pypi.org/project/pycountry/>
* Geoip2 4.8.0 Documentation:<https://geoip2.readthedocs.io/en/latest/>
* GeoLite Database: <https://github.com/P3TERX/GeoLite.mmdb>
* FortiGuard API:<https://fortiguard.fortinet.com/api/threatmap/live/outbreak>
* Radware API:https://ltm-prod-api.radware.com/map/attacks?limit=20
* The Hacker News RSS:<https://feeds.feedburner.com/TheHackersNews>
* Dark Reading RSS:<https://www.darkreading.com/rss.xml>
* The420.in RSS:<https://the420.in/feed>
* AlienVault Reputation:<https://reputation.alienvault.com/reputation.unix>
* Binary Defense Banlist:<https://www.binarydefense.com/banlist.txt>
* FraudGuard API:<https://api.fraudguard.io/landing-page-map>
* Talos Top Senders:<https://talosintelligence.com/cloud_intel/top_senders_list>

**temp**



**1. FortiGuard Outbreak API**The FortiGuard Outbreak API provides live, global threat intelligence data focusing on malware outbreaks and attack events. The system retrieves data from <https://fortiguard.fortinet.com/api/threatmap/live/outbreak?outbreak_id=0> using the parameter outbreak\_id=0 to capture all threat types without category filters.

**Data Collection:**

In the Fortiguard data collection phase, threat intelligence data is received in a raw format. This data is typically embedded within a nested structure under keys like **ips** and includes metadata such as attack ID, vulnerability type, severity level, and geographic details like source and destination countries and their respective coordinates.   
  
During the Fortiguard data collection phase, the raw threat intelligence data is captured in a structured JSON format, where each entry represents a detected threat event. The data includes key metadata such as source and destination country code, geographic coordinates for both endpoints, attack name, the number of attacks observed, etc. Each record also lists the types of attacks detected and the corresponding timestamp. This concise and organized structure allows for efficient analysis and correlation of threat activity across different regions and attack types, supporting timely and informed security responses.

**Raw data:**

{"ips":{"1748951935958":[{"redis\_ms":"1748951936931-0","id":"40772","outbreak\_id":"0","vuln\_type":"ips","vuln\_name":"D-Link.Devices.HNAP.SOAPAction-Header.Command.Execution","severity":"High","dest\_country":"JP","dest\_city":"Higashiomi","dest\_lat":35.11246,"dest\_long":136.20776,"src\_country":"AR","src\_city":"Rosario","src\_lat":-32.94424,"src\_long":-60.65054,"hit":1,"replay":false}]}}

  
The raw input is first parsed into a clean and structured JSON format that the backend system can understand and work with.

**Structured Data:**

[

{

"Source Country Code": "AR",

"Source Country Name": "Argentina",

"Source Latitude": -32.94424,

"Source Longitude": -60.65054,

"Destination Country Code": "JP",

"Destination Country Name": "Japan",

"Destination Latitude": 35.11246,

"Destination Longitude": 136.20776,

"Attack Count": 1,

"Attack Types": [

"exploit"

],

"Timestamp": "2025-06-03T14:39:04.450045"

},

During this conversion, any entries lacking either source or destination country information are discarded. This validation is critical since both endpoints are necessary to visualize an attack route on the map. After validation, duplicates within the Fortiguard dataset are detected and removed, ensuring that only unique, actionable threat events proceed to the next stage.

**Aggregation:**Once Fortiguard’s threat data has been filtered and cleaned, it is temporarily stored as a standalone dataset. However, many attack events can appear multiple times or across different systems, so aggregation becomes essential. Fortiguard entries are held ready for cross-source aggregation where they will be merged with entries from other sources.  
  
**Processing:**  
Once Fortiguard’s threat data is aggregated, it enters the processing stage. Here, duplicate entries are ensuring that only unique threat records based on attack name and source/destination country codes—are retained. This includes data from Fortiguard. Then, individual threat events are grouped based on their country-to-country attack patterns. For instance, if multiple entries show attacks originating from Argentina and targeting Japan, they are grouped under one attack path: **AR - JP**. Instead of rendering each event separately on the map, this path is displayed as a **single visual element**, with an **attack count** indicating how many times that specific route has been detected. This grouping helps simplify the visualization while highlighting attack frequency.

* **Attack details**: vuln\_name, vuln\_type, severity, count
* **Geographic data**: src\_country, src\_lat, src\_long, dest\_country, dest\_lat, dest\_long
* **Metadata**: timestamps, IDs, hit counts

**2. Radware Attacks API**

**Collection:**

Similar to Fortiguard, Radware data also arrives in a raw form, though with a few key differences.   
**Raw data:**

[[{"sourceCountry":"NL","destinationCountry":"SE","millisecond":396,"type":"scanners","weight":"Light","attackTime":"2025-06-03T14:56:45.3966667"}]]

  
The raw input is then parsed into a clean and structured JSON format that the backend system can understand and work with.

[{

"Source Country Code": "NL",

"Source Country Name": "Netherlands",

"Source Latitude": 52.1326,

"Source Longitude": -74.9528,

"Destination Country Code": "SE",

"Destination Country Name": "Sweden",

"Destination Latitude": 40.0305,

"Destination Longitude": -74.9528,

"Attack Count": 1,

"Attack Types": [

"exploit"

],  
 "Timestamp": "2025-06-03T14:39:04.450045"}]

  
While Radware's entries do contain information about source and destination countries, they often lack geographical coordinates such as latitude and longitude. After the raw data is parsed and transformed into JSON, entries that do not contain both source and destination countries are removed to maintain consistency.

To compensate for the missing coordinates, the backend uses the **pycountry** Python library to map country names or codes to standardized forms. This enables consistent labeling and allows geographic mapping logic (like placing threats on a world map) to function correctly even without explicit coordinates. Similar to Fortiguard, Radware’s internal duplicates are also removed during this collection phase to reduce redundancy.

**Aggregation:**

Radware data undergoes a similar pre-aggregation cleanup. After filtering and enhancing the entries (like adding standardized country codes), the Radware threat records are stored separately. When it’s time to aggregate, Radware’s dataset is combined with Fortiguard’s dataset into a unified structure. Since some threats may be reported by both platforms, a **second round of deduplication** is performed across the combined dataset to eliminate any cross-source duplicates.

This aggregation step is crucial because it ensures that the threat intelligence data used in the threat map is not only comprehensive (covering multiple sources) but also **free of redundancies**. It creates a reliable, unified threat landscape view by merging and de-duplicating information from both Fortiguard and Radware.  
  
**Processing:**  
The processing logic for Radware data follows the same model. After aggregation, entries are grouped by their source and destination countries. All threats from one country to another are merged into a single unit with a cumulative count. Even if Radware didn't originally provide geographic coordinates, the use of standardized country data ensures that the grouping remains consistent with Fortiguard’s.

The grouped dataset from both sources is what ultimately powers the real-time visual elements of the cyber threat map, allowing viewers to see aggregated attack paths and their relative severity in a clean, simplified interface.

**Collection**

### 1. FortiGuard Outbreak API

The FortiGuard Outbreak API provides live, global threat intelligence focusing on malware outbreaks and attack events. Data is retrieved from:

The **NewsDataCollector** class asynchronously retrieves cybersecurity news articles from multiple RSS feeds, including:

* The Hacker News RSS ([**https://feeds.feedburner.com/TheHackersNews**](https://feeds.feedburner.com/TheHackersNews))
* Dark Reading RSS ([**https://www.darkreading.com/rss.xml**](https://www.darkreading.com/rss.xml))
* The 420 RSS ([**https://the420.in/feed**](https://the420.in/feed))

The collector parses feed entries to extract title, summary (with fallback to description), link, and publication timestamp. Articles undergo relevance filtering using comprehensive keyword matching on cybersecurity terms. The system employs a three-tier filtering approach with primary keywords, secondary keywords, and exclusion terms to ensure content relevance.

**Source Collection:** The **NewsDataCollector** asynchronously fetches articles from all configured RSS feeds using **feedparser** with a maximum of 3 retry attempts per source. The collector operates with zero interval delay, indicating non-streaming batch collection mode.

**Data Ingestion:** Each feed entry undergoes extraction of core fields:

* Title and summary/description text (with fallback mechanism)
* Publication link and timestamp
* Entries missing both title and link are automatically discarded

**Error Handling:** Feed parsing errors and network issues are logged without halting overall processing. Partial source failures do not prevent collection from other successful feeds.

**Normalization:** Timestamps default to current UTC time in ISO 8601 format when publication dates are unavailable. All text fields undergo whitespace trimming for consistency.

**Preprocessing**

**Keyword Filtering:** Articles must contain at least one primary keyword using regex word boundary matching for precision. The filtering system includes:

* **Primary Keywords:** ransomware, malware, exploit, vulnerability, breach, zero-day, attack, compromised, infected, stolen, hacked, leak, backdoor, trojan, rootkit, spyware, security, cyber
* **Secondary Keywords:** cybercrime, phishing, ddos, apt, hacking, credential, cyberattack, databreach, hack, payload, threat, botnet, mitigation, critical, authentication, attacker, command and control, lateral movement, exfiltration, intrusion, security flaw
* **Exclusion Terms:** webinar, workshop, training, course, certification, conference, roundtable, partner, sponsored, promotion, discount, offer, register now, sign up, earn, sale, subscription, tutorial, guide, how to, introduction to

**Filtering Logic:** Articles containing any exclusion term are immediately discarded. Only articles with at least one primary keyword pass the relevance filter. The system performs case-insensitive matching on combined title and summary text.

**Output Standardization:** The output consists of a filtered list of articles, each containing three standardized fields: **title**, **link**, and **timestamp**. The relevance matching function returns matched keywords but these are not included in the final output structure.

**Output Format**

[{

"title": "Android Trojan Crocodilus Now Active in 8 Countries, Targeting Banks and Crypto Wallets",

"link": "https://thehackernews.com/2025/06/android-trojan-crocodilus-now-active-in.html",

"timestamp": "Tue, 03 Jun 2025 15:04:00 +0530"

},

]

#### 

**Example:**

The MaliciousIPCollector class asynchronously retrieves threat intelligence data from multiple cybersecurity feeds, including:

AlienVault OTX Reputation Feed (<https://reputation.alienvault.com/reputation.unix>)

Binary Defense Banlist ([**https://www.binarydefense.com/banlist.txt**](https://www.binarydefense.com/banlist.txt))

FraudGuard Threat Intelligence (<https://api.fraudguard.io/landing-page-map>)

Cisco Talos Intelligence ([**https://talosintelligence.com/cloud\_intel/top\_senders\_list**](https://talosintelligence.com/cloud_intel/top_senders_list))

The collector performs IP address extraction, validation, and geolocation enrichment across all configured sources. Geographic positioning is achieved through GeoLite2 database integration, enabling precise threat visualization on interactive maps. The system employs regex-based IP extraction with comprehensive validation and deduplication mechanisms.

Source Collection: The MaliciousIPCollector asynchronously fetches threat data from all configured intelligence feeds with a maximum of 5 retry attempts per source. The collector operates with configurable interval delay, supporting both batch and continuous collection modes.

Data Ingestion: Each threat intelligence source undergoes specialized parsing:

IP address extraction using regex pattern **\b(?:\d{1,3}\.){3}\d{1,3}\b**

Source-specific format handling for different feed structures

Entries with invalid IP addresses are automatically discarded

**Error Handling:** Individual source failures do not impact overall collection operations. Network timeouts, parsing errors, and database lookup failures are logged without halting processing. Partial source failures do not prevent collection from other successful feeds.

Normalization: All extracted IP addresses undergo validation using Python's ipaddress module. Geographic coordinates default to null values when GeoLite2 database lookups fail. All IP addresses are sorted and deduplicated for consistency.

**Preprocessing:**

**IP Validation:** All extracted strings must pass IPv4 address validation using standard library functions. The system performs comprehensive validation to ensure only legitimate IP addresses enter the dataset.

**Geolocation Enhancement:** Each validated IP address undergoes geolocation lookup through the GeoLite2-City database. The system includes:

**Database Integration:** MaxMind GeoLite2-City.mmdb for coordinate resolution

**Coordinate Extraction**: Latitude and longitude decimal degree values

**Fallback Handling**: Graceful degradation to null coordinates for unresolvable addresses

**Type Classification:** Threat indicators are classified based on originating intelligence source:

**Malicious:** General threat indicators from AlienVault, Binary Defense, and FraudGuard

**Spam:** Email-specific threats from Cisco Talos spam sender intelligence

**Output Standardization:** The output consists of a deduplicated list of threat indicators, each containing four standardized fields: ip, latitude, longitude, and type. The geolocation coordinates enable direct integration with mapping visualization systems.

**Tab 3**