# Technical Documentation for CVE Collector and API Module

### 

### **Table of Contents**

**1. Introduction**

**2. System Architecture**

**3. Components**

**1. cve\_collector\_main.py**

**2. mongo\_connection.py**

**3. api\_main.py**

**4. Data Validation and Sanity Checks**

**5. Security Enhancements**

**6. Deployment**

**7. Usage**

**8. Other Approaches And Challenges Faced**

**9. Sample Configurations**

Introduction

The CVE Collector and API Module is designed to fetch CVE (Common Vulnerabilities and Exposures) data from the NVD (National Vulnerability Database) API, store it in a MongoDB database, and provide a RESTful API to query this data.

This module consists of three main components:

1. cve\_collector\_main.py: Handles the fetching and updating of CVE data from the NVD API.

2. mongo\_connection.py: Manages the MongoDB connection and data operations.

3. api\_main.py: Provides a FastAPI-based RESTful API for querying the CVE data.

### System Architecture

The system architecture consists of three layers:

1. Data Collection Layer : This layer is responsible for collecting data from the NVD API and storing it in the MongoDB database. It handles initial data collection as well as incremental updates.

2. Database Layer : This layer manages the MongoDB connection and provides functions for storing, updating, and retrieving data.

3. API Layer : This layer exposes a RESTful API using FastAPI for querying the CVE data stored in the MongoDB database.

### Components

###### cve\_collector\_main.py

This component is responsible for fetching CVE data from the NVD API and storing it in the MongoDB database. It provides both one-time data collection and incremental updates.

**Functions :**

- get\_cve\_data\_from\_api(session, params, retries=3): Fetches data from the NVD API with retry logic for handling transient errors.

- collect\_cve\_data\_at\_once(start\_index=0, results\_per\_page=Pagination): Collects all CVE data from the API starting from a specified index.

- update\_cve\_from\_api(start\_index=0, results\_per\_page=Pagination): Updates the CVE data in the database based on the last synchronization time.

- run\_one\_time\_scan(): Runs a one-time data sync.

- run\_incremental\_update(): Runs incremental data updates and schedules them to run every X days as configured.

###### mongo\_connection.py

This component manages the MongoDB connection and provides functions for storing, updating, and retrieving CVE data.

**Functions**

- \_\_init\_\_(): Initializes the MongoDB connection and sets up the collections.

- get\_mongo\_collection(db\_name, collection\_name): Retrieves a MongoDB collection.

- store\_cve\_data(cve\_data): Stores CVE data in the database using bulk write operations.

- get\_last\_sync\_time(): Retrieves the last synchronization time.

- update\_last\_sync\_time(last\_sync\_time): Updates the last synchronization time.

- add\_or\_update\_cve\_in\_mongo(data): Adds or updates CVE data in the database based on the last modified timestamp.

###### api\_main.py

This component provides a FastAPI-based RESTful API for querying the CVE data stored in the MongoDB database. It includes rate limiting to prevent abuse.

**Endpoints**

- GET /cve/cve\_id: Retrieves a CVE by its ID.

- GET /cve/score: Retrieves CVEs within a specified score range.

- GET /cve/modified: Retrieves CVEs modified within a specified number of days.

### Data Validation and Sanity Checks

To ensure the integrity and validity of the data before storing it in MongoDB, the following data validation and sanity checks are implemented:

1. Schema Validation: Use Pydantic models to validate the structure of the CVE data.

2. Type Checking: Ensure that the data types of all fields match the expected types.

3. Range Checks: Validate that numeric fields (e.g., scores) fall within acceptable ranges.

4. Required Fields: Ensure that all required fields are present in the data.

5. Exception Handling : Proper handling that ensures process flow won’t be impacted in case of any exception points

### Security Enhancements

To enhance the security of the module, the following measures are implemented:

1. Rate Limiting: Implement rate limiting using the SlowAPI library to prevent abuse of the API endpoints.

2. Error Handling: Implement comprehensive error handling to handle various HTTP errors and exceptions.

3. Exponential Backoff: Use exponential backoff for retrying API requests to avoid overloading the NVD API.

4. Indexing: Create appropriate indexes in MongoDB to optimize query performance and prevent potential denial-of-service attacks due to slow queries.

5. Secure Credentials: All credentials and configurations are kept secured in .env file

### Deployment

To deploy this module, follow these steps:

1. Install Dependencies: Install the required Python packages using pip.

*pip install -r requirements.txt*

2. Run Data Collector: Start the data collector to fetch and store CVE data.

*python cve\_collector\_main.py*

3. Run API Server: Start the FastAPI server to expose the RESTful API.

*python api\_main.py*

### Usage

###### Fetching CVE Data

1. Initial Data Collection: Run the one-time data sync to collect all CVE data.

1. Incremental Updates: Run the incremental update to keep the data up-to-date.

###### Querying CVE Data

1. Retrieve CVE by ID:

GET /cve/cve\_id?id=CVE-2021-34527

2. Retrieve CVEs by Score:

GET /cve/score?min\_score=9.5&max\_score=10.0

3. Retrieve CVEs by Modified Date:

GET /cve/modified?days=7

This documentation provides an overview of the CVE Collector and API Module, including its architecture, components, validation checks, security measures, and usage instructions. For detailed implementation and code, refer to the source files provided.

### Other Approaches And Challenges Faced

I have thought in multiple directions for completing this project. Briefing few of those approaches which tried below along with the disadvantages of using those and challenges faced.  
  
Data Collector Layer

Initially, these were normal function calls with direct invocations. To speed up the overall process and attain parallel processing, async functions has been used.  
  
Change histoty API(https://services.nvd.nist.gov/rest/json/cvehistory/2.0)  
Tried using CVE change history API initially for syncing data updation. On finding that it involves too much overheads and complexity, changed to current approach where normal CVE API is hit with additional *lastModStartDate* and *lastModEndDate* parameters.

Database Layer

Enhanced data validation, using pydantic arbitrary\_types\_allowed, dict validation and several other parameters have been tried out to ensure data sanity and validation. But these checks has been adversely affecting the performance of the system as it slows down entire process. So, it has been decided to follow basic data validation in code and data de-duplication and validation has been handled in mongo db directly.  
  
Also added indexing on all parameters which is accessed from mongo collection in the API. This given a notable increase in API response time.

###### API Layer

Initially used Flask API to get the process done. On finding that it was too slow, async functions has been used along with Flask to improve the speed. Small improvement was there but was not satisfactory.  
  
Finally, switched to Fast API and it brought a dramatic improvement in API performance and reponse time. Mongo db indexing for necessary columns was also a plus here.  
  
Implemented rate limiting and input validation for added security.

### Sample Configurations

Below are sample configurations from two mongo collections used.  
  
cve\_sync\_collection

{

"\_id" : "cve\_nvd\_data\_sync\_col",

"last\_sync\_time" : "2024-07-19T23:04:12.399877"

}

**cve\_details\_collection**

{

"\_id" : ObjectId("669a7871230ca01628a093d5"),

"cve" : {

"id" : "CVE-2000-0388",

"sourceIdentifier" : "cve@mitre.org",

"published" : "1990-05-09T04:00:00.000",

"lastModified" : "2008-09-10T19:04:33.930",

"vulnStatus" : "Analyzed",

"cveTags" : [],

"descriptions" : [

{

"lang" : "en",

"value" : "Buffer overflow in FreeBSD libmytinfo library allows local users to execute commands via a long TERMCAP environmental variable."

}

],

"metrics" : {

"cvssMetricV2" : [

{

"source" : "nvd@nist.gov",

"type" : "Primary",

"cvssData" : {

"version" : "2.0",

"vectorString" : "AV:N/AC:L/Au:N/C:P/I:P/A:P",

"accessVector" : "NETWORK",

"accessComplexity" : "LOW",

"authentication" : "NONE",

"confidentialityImpact" : "PARTIAL",

"integrityImpact" : "PARTIAL",

"availabilityImpact" : "PARTIAL",

"baseScore" : 7.5

},

"baseSeverity" : "HIGH",

"exploitabilityScore" : 10.0,

"impactScore" : 6.4,

"acInsufInfo" : false,

"obtainAllPrivilege" : false,

"obtainUserPrivilege" : true,

"obtainOtherPrivilege" : false,

"userInteractionRequired" : false

}

]

},

"weaknesses" : [

{

"source" : "nvd@nist.gov",

"type" : "Primary",

"description" : [

{

"lang" : "en",

"value" : "NVD-CWE-Other"

}

]

}

],

"configurations" : [

{

"nodes" : [

{

"operator" : "OR",

"negate" : false,

"cpeMatch" : [

{

"vulnerable" : true,

"criteria" : "cpe:2.3:o:freebsd:freebsd:3.0:\*:\*:\*:\*:\*:\*:\*",

"matchCriteriaId" : "EE38C50A-81FE-412E-9717-3672FAE6A6F4"

},

{

"vulnerable" : true,

"criteria" : "cpe:2.3:o:freebsd:freebsd:3.1:\*:\*:\*:\*:\*:\*:\*",

"matchCriteriaId" : "263F3734-7076-4EA8-B4C0-F37CFC4E979E"

},

{

"vulnerable" : true,

"criteria" : "cpe:2.3:o:freebsd:freebsd:3.2:\*:\*:\*:\*:\*:\*:\*",

"matchCriteriaId" : "0419DD66-FF66-48BC-AD3B-F6AFD0551E36"

},

{

"vulnerable" : true,

"criteria" : "cpe:2.3:o:freebsd:freebsd:3.3:\*:\*:\*:\*:\*:\*:\*",

"matchCriteriaId" : "C3518628-08E5-4AD7-AAF6-A4E38F1CDE2C"

},

{

"vulnerable" : true,

"criteria" : "cpe:2.3:o:freebsd:freebsd:3.4:\*:\*:\*:\*:\*:\*:\*",

"matchCriteriaId" : "B982342C-1981-4C55-8044-AFE4D87623DF"

}

]

}

]

}

],

"references" : [

{

"url" : "ftp://ftp.freebsd.org/pub/FreeBSD/CERT/advisories/FreeBSD-SA-00%3A17.libmytinfo.asc",

"source" : "cve@mitre.org"

},

{

"url" : "http://www.securityfocus.com/bid/1185",

"source" : "cve@mitre.org"

}

]

}

}