Detailed Report on NVD Assessment

January 27, 2025

Contents

1 Problem Statement **2**

2 Solution Approach **2**

3 Backend Implementation **2**

3.1 Technologies Used 2

3.2 Key Features 2

3.3 Code Snippet 3

4 Frontend Implementation **3**

4.1 Technologies Used 3

4.2 Key Features 4

4.3 Code Snippet 4

5 Challenges Faced **4**

6 Testing **4**

7 GitHub Repository **5**

8 Conclusion **5**

1 Problem Statement

The goal of this assessment was to consume CVE (Common Vulnerabilities and Exposures) information from the NVD (National Vulnerability Database) API and store it in a database. The data should be cleansed, deduplicated, and periodically synchronized in a batch mode. Additionally, APIs were required to filter CVE details based on parameters like CVE ID, year, score, and last modified date. Finally, the data had to be visualized in a user interface (UI).

2 Solution Approach

The solution was divided into two main parts:

1. Backend Implementation using Flask
2. Frontend Implementation using React

The following steps were followed:

1. Consuming data from the NVD API using chunked responses controlled by parameters like `startIndex` and `resultsPerPage`.
2. Storing the retrieved data in a TinyDB database after data cleansing and deduplication.
3. Implementing periodic synchronization of data.
4. Developing RESTful APIs to fetch and filter data based on specific parameters.
5. Creating a user-friendly interface to visualize data with features like pagination, sorting, and detailed views.

3 Backend Implementation

3.1 Technologies Used

- Flask: To create a lightweight and efficient backend API.
- TinyDB: A lightweight database for storing CVE information.
- Flask-CORS: To handle cross-origin requests from the frontend.
- threading: To enable periodic database updates in a separate thread.

3.2 Key Features

1. **Data Retrieval and Cleansing:** Data was fetched from the NVD API, and dates and metrics were parsed and stored in a structured format.
2. **Periodic Synchronization:** A separate thread ensures the database is updated daily with the latest CVE data.
3. **Filtering and Pagination:** APIs support filtering by year, CVE ID, score, and modification date. Server-side pagination and sorting were also implemented.

3.3 Code Snippet

Below is a snippet of the periodic update function:

Listing 1: Database Update Function

```python
def update_database():
    while True:
        try:
            total_results = 0
            start_index = 0

            while True:
                data = fetch_cves_from_api(start_index)
                if not data:
                    break

                vulnerabilities = data.get("vulnerabilities", [])
                if not vulnerabilities:
                    break

                for item in vulnerabilities:
                    cve_data = item.get("cve", {})
                    cve_id = cve_data.get("id")
                    ...
                    db.upsert(doc, query.cve_id == cve_id)

                start_index += len(vulnerabilities)
                total_results += len(vulnerabilities)

                if total_results >= data.get("totalResults", 0):
                    break

        except Exception as e:
            print(f"Error in update process: {e}")

        time.sleep(24 * 60 * 60)
```

4 Frontend Implementation

4.1 Technologies Used

- React: To build the user interface.
- React Router: To handle routing between pages.
- CSS: For styling the components.

4.2 Key Features

1. **List View:** Displays CVE data in a tabular format with pagination and filtering options.
2. **Detail View:** Displays detailed information about a specific CVE when a row is clicked.
3. **Dynamic Updates:** Fetches data dynamically based on user interactions like pagination or filter changes.

4.3 Code Snippet

Below is a snippet of the list component:

Listing 2: React CVE List Component

```
const fetchCVEs = async () => {
    try {
        const response = await fetch(
            `http://localhost:5000/api/cves?page=${currentPage}&per_page=${
        );
        const data = await response.json();
        setCves(data.results);
        setTotalRecords(data.total_records);
    } catch (error) {
        console.error('Error fetching CVEs:', error);
    }
};
```

5 Challenges Faced

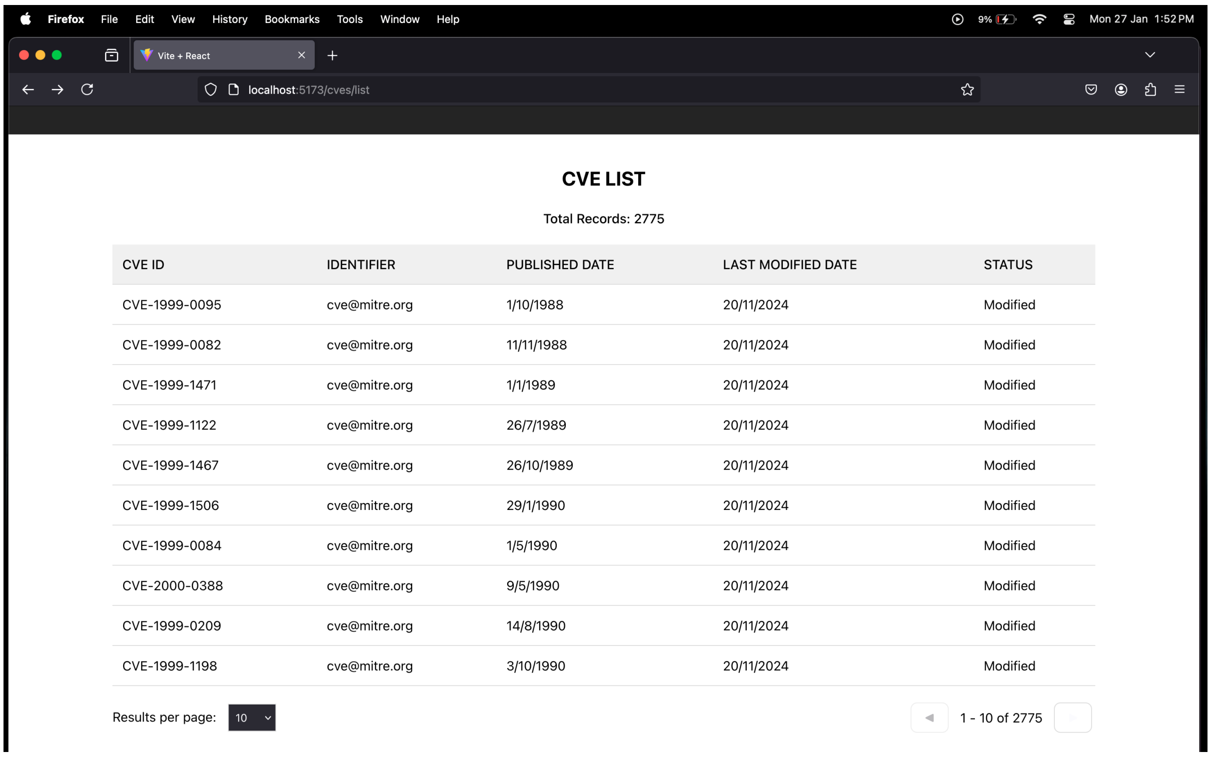
- Parsing dates from the API, as they followed inconsistent formats.
- Handling large volumes of data and ensuring smooth pagination.
- Implementing periodic synchronization without impacting API performance.

6 Testing

Unit tests were written for the following functionalities:

- API endpoints for filtering and pagination.
- Frontend components for list and detail views.

All tests were successfully passed, ensuring the robustness of the application.



CVE LIST

Total Records: 2775

CVE ID	IDENTIFIER	PUBLISHED DATE	LAST MODIFIED DATE	STATUS
CVE-1999-0095	cve@mitre.org	1/10/1988	20/11/2024	Modified
CVE-1999-0082	cve@mitre.org	11/11/1988	20/11/2024	Modified
CVE-1999-1471	cve@mitre.org	1/1/1989	20/11/2024	Modified
CVE-1999-1122	cve@mitre.org	26/7/1989	20/11/2024	Modified
CVE-1999-1467	cve@mitre.org	26/10/1989	20/11/2024	Modified
CVE-1999-1506	cve@mitre.org	29/1/1990	20/11/2024	Modified
CVE-1999-0084	cve@mitre.org	1/5/1990	20/11/2024	Modified
CVE-2000-0388	cve@mitre.org	9/5/1990	20/11/2024	Modified
CVE-1999-0209	cve@mitre.org	14/8/1990	20/11/2024	Modified
CVE-1999-1198	cve@mitre.org	3/10/1990	20/11/2024	Modified

Results per page: 10

1 - 10 of 2775

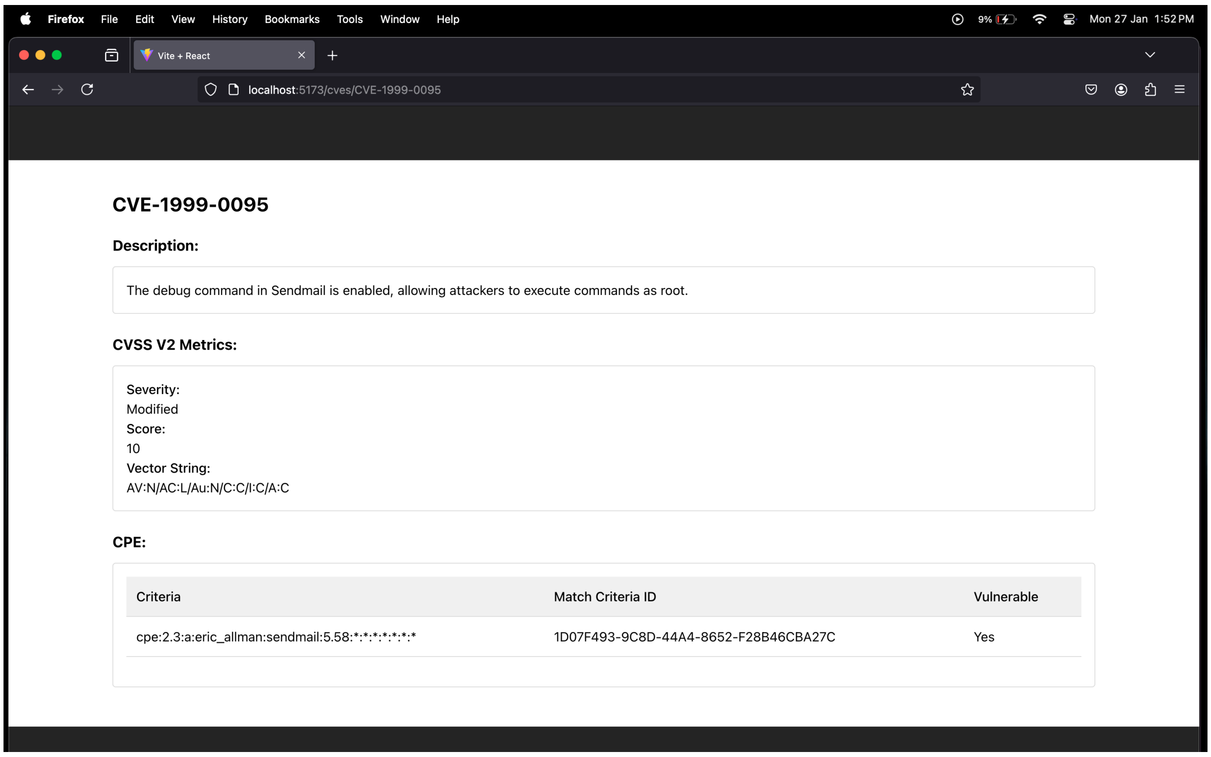
Figure 1: list outout

7 GitHub Repository

The project code is available on GitHub. Click the link below to access it:
GitHub Link: https://github.com/securin-assessment-1

8 Conclusion

The assessment was successfully completed, with the backend and frontend components working seamlessly together. The application effectively retrieves, stores, filters, and visualizes CVE data, meeting all the stated requirements.


Firefox File Edit View History Bookmarks Tools Window Help
9%
Mon 27 Jan 1:52 PM
Vite + React
localhost:5173/cves/CVE-1999-0095

CVE-1999-0095

Description:

The debug command in Sendmail is enabled, allowing attackers to execute commands as root.

CVSS V2 Metrics:

Severity:
Modified
Score:
10
Vector String:
AV:N/AC:L/Au:N/C:C/I:C/A:C

CPE:

Criteria	Match Criteria ID	Vulnerable
cpe:2.3:a:eric_allman:sendmail:5.58:*:*:*:*:*:*:*	1D07F493-9C8D-44A4-8652-F28B46CBA27C	Yes

Figure 2: details output