SECURIN ASSESSMENT

Name: Venkatakrishnan R

Date : 15-05-2024

PROBLEM STATEMENT

Keep track of the latest Common Vulnerabilities and Exposures (CVEs) listed on the National Vulnerability Database (NVD) using their API available at https://nvd.nist.gov/developers/vulnerabilities And offer a user-friendly interface to easily read and visualize the data related to these CVEs.

OVERVIEW

- The Project utilises Express.js as Web Server.
- Local MongoDB Container as Database with mongoose for connectivity.
- EJS for rendering HTML template to visualise the user interface.
- JEST for implementing Unit tests.

FUNCTIONALITIES IMPLEMENTED

1) The CVE data retrieval from the NVD API batchwise using **startIndex** and **resultsPerPage** as query parameters and stored in Mongo DB.
2) Periodically update CVE data from CVE HISTORY API to my local Mongo DB every 30 minutes.
3) Remove the **REJECTED** vulnerabilities from the Database
4) Present the CVE data in a tabular format.
5) Limit the number of rows displayed in table.(10, 50 ,100).
6) Pagination on the server side as well on client side is implemented to present the split the records into different pages.
7) Search Filtration of record using Multiple Parameters
    - Year
    - CVE id
    - Last Modified
    - Base Score(Greater than and Lesser than score)
8) Each of the row is linked to detailed description of the particular CVE.
9) Total Records in the database and range of records is displayed.
10) Sorting of dates in server side is implemented.
11) Unit tests for the API endpoints are implemented using JEST.

API ENDPOINTS

ENDPOINT **:** http://<domain>:<port>/

/ & /Cve/lists

- Lists all the CVE Data from the database

QUERY PARAMETERS

- ?page=
  -returns the contents of page requested
- ?limit=
  -limits the number of records
  -options:10 , 50 ,100
- ?year=
  -returns all the CVEs in that particular year
- ?lastmodified=
  -returns a sorted records of last modified data in descending order and limits is set to number of days given i.e N days .
- ?lt=
  -returns Base score value of CVE which is lesser than input given
- ?gt=
  -returns Base score value of CVE which is greater than input given
- ?id=

  -returns the records of CVE with ids with matching with input given partially or fully.

/cves/<CVE-ID>

- Return the detailed description of the particular CVE matching with the following CVE-ID.

OUTPUT

CVE LIST PAGE

CVE List

Total Records: 249967

CveId: Year: Lesser than score:

Greater Than Score: Last Modified:

Results Per Page 10 Search

CVE ID	Identifier	Published Date	Last Modified Date	Status
CVE-1999-0095	cve@mitre.org	30 Sep 1988	11 Jun 2019	Modified
CVE-1999-0082	cve@mitre.org	10 Nov 1988	09 Sep 2008	Analyzed
CVE-1999-1471	cve@mitre.org	31 Dec 1988	05 Sep 2008	Analyzed
CVE-1999-1122	cve@mitre.org	25 Jul 1989	02 May 2018	Modified
CVE-1999-1467	cve@mitre.org	25 Oct 1989	18 Dec 2017	Modified
CVE-1999-1506	cve@mitre.org	28 Jan 1990	05 Sep 2008	Analyzed
CVE-1999-0084	cve@mitre.org	30 Apr 1990	09 Oct 2017	Modified
CVE-2000-0388	cve@mitre.org	08 May 1990	10 Sep 2008	Analyzed
CVE-1999-0209	cve@mitre.org	13 Aug 1990	09 Sep 2008	Analyzed
CVE-1999-1198	cve@mitre.org	02 Oct 1990	05 Sep 2008	Analyzed

1 2 3 4 **1-10 of 249967 records**

CVE DETAILS PAGE

CVE-1999-0095

Description:

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

CVSS V2 Metrics:

Severity: HIGH **Score:** 10

Vector String: AV:N/AC:L/Au:N/C:C/I:C/A:C

Access Vector	Access Complexity	Authentication	Confidentiality Impact	Integrity Impact	Availability Impact
NETWORK	LOW	NONE	COMPLETE	COMPLETE	COMPLETE

Scores:

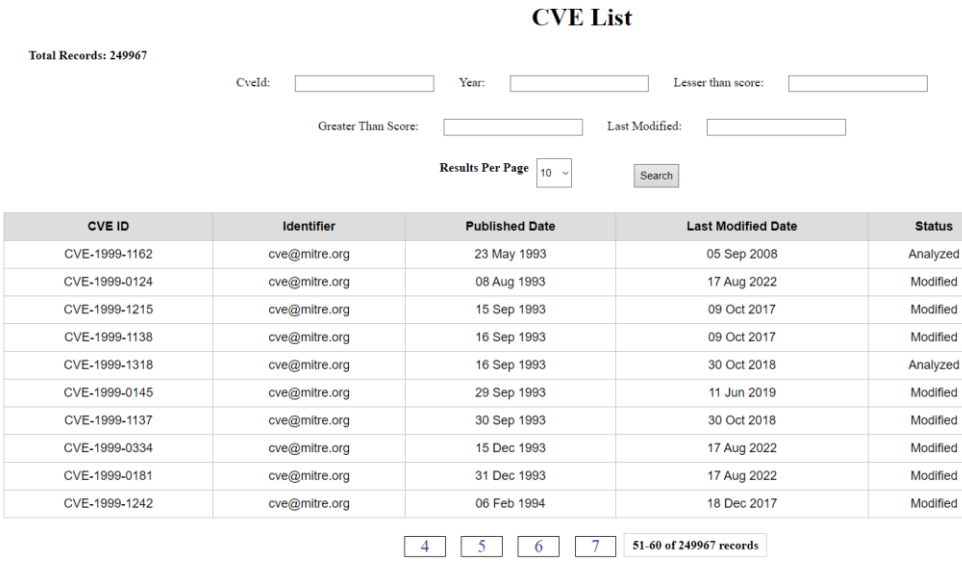
Exploitability Score: 10

Impact Score: 10

CPE:

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

PAGINATION EXAMPLE



CVE List

Total Records: 249967

CveId: Year: Lesser than score:

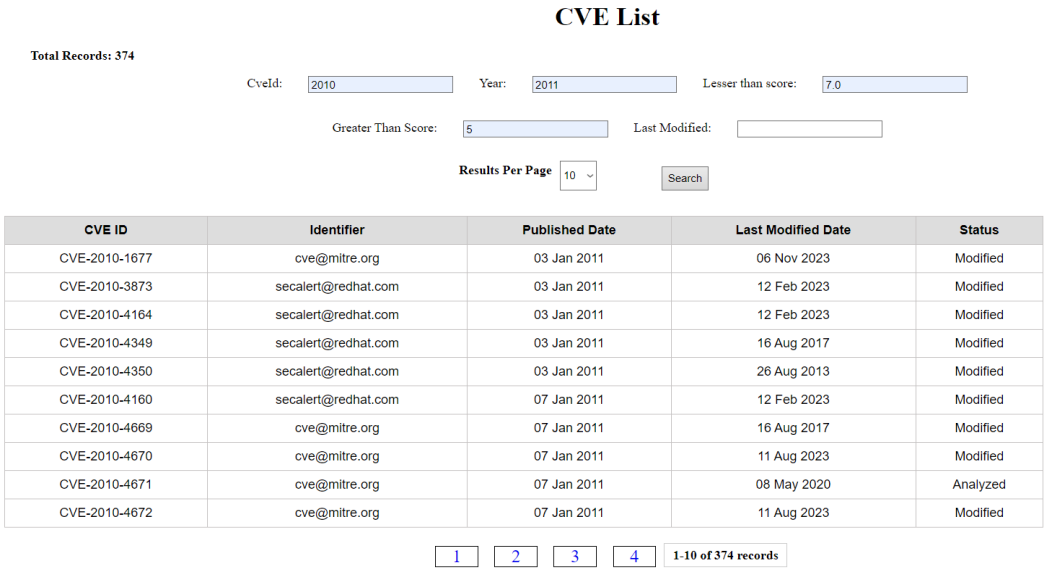
Greater Than Score: Last Modified:

Results Per Page 10 Search

CVE ID	Identifier	Published Date	Last Modified Date	Status
CVE-1999-1162	cve@mitre.org	23 May 1993	05 Sep 2008	Analyzed
CVE-1999-0124	cve@mitre.org	08 Aug 1993	17 Aug 2022	Modified
CVE-1999-1215	cve@mitre.org	15 Sep 1993	09 Oct 2017	Modified
CVE-1999-1138	cve@mitre.org	16 Sep 1993	09 Oct 2017	Modified
CVE-1999-1318	cve@mitre.org	16 Sep 1993	30 Oct 2018	Analyzed
CVE-1999-0145	cve@mitre.org	29 Sep 1993	11 Jun 2019	Modified
CVE-1999-1137	cve@mitre.org	30 Sep 1993	30 Oct 2018	Modified
CVE-1999-0334	cve@mitre.org	15 Dec 1993	17 Aug 2022	Modified
CVE-1999-0181	cve@mitre.org	31 Dec 1993	17 Aug 2022	Modified
CVE-1999-1242	cve@mitre.org	06 Feb 1994	18 Dec 2017	Modified

4 5 6 7 **51-60 of 249967 records**

CVE TABLES WITH FILTERS



CVE List

Total Records: 374

CveId: 2010 Year: 2011 Lesser than score: 7.0

Greater Than Score: 5 Last Modified:

Results Per Page 10 Search

CVE ID	Identifier	Published Date	Last Modified Date	Status
CVE-2010-1677	cve@mitre.org	03 Jan 2011	06 Nov 2023	Modified
CVE-2010-3873	secalert@redhat.com	03 Jan 2011	12 Feb 2023	Modified
CVE-2010-4164	secalert@redhat.com	03 Jan 2011	12 Feb 2023	Modified
CVE-2010-4349	secalert@redhat.com	03 Jan 2011	16 Aug 2017	Modified
CVE-2010-4350	secalert@redhat.com	03 Jan 2011	26 Aug 2013	Modified
CVE-2010-4160	secalert@redhat.com	07 Jan 2011	12 Feb 2023	Modified
CVE-2010-4669	cve@mitre.org	07 Jan 2011	16 Aug 2017	Modified
CVE-2010-4670	cve@mitre.org	07 Jan 2011	11 Aug 2023	Modified
CVE-2010-4671	cve@mitre.org	07 Jan 2011	08 May 2020	Analyzed
CVE-2010-4672	cve@mitre.org	07 Jan 2011	11 Aug 2023	Modified

1 2 3 4 **1-10 of 374 records**

WORKFLOW

Admin launches the NODE.JS server, connecting it to an empty MongoDB container initially, specified in the .env file's MONGO_URL. The server operates on the designated port specified in the .env file as PORT. To populate the local database with data from the NVD API, the admin starts the helper.js module of the application and the data is fetched in batches, with batch size determined by configurations, taking approximately 30 minutes to 1 hour depending on data volume.

For public access, two endpoints are available. Initially, users are directed to the CVE lists page via http://<domain>:<PORT>/cves/lists. The URL query parameters defaults

- page : 1,
- limit : 10,
- year : “ ” (i.e. empty string)
- lastModified: -1
- lt(lesser than score):10.0
- gt(greater than score):0.0
- id : “ ” (i.e. empty string)

Interaction with search parameter modifies these parameters, reflecting corresponding changes in displayed table.

Additionally, the server automatically checks for NVD updates from the https://services.nvd.nist.gov/rest/json/cvehistory/2.0 API every 30 minutes, updating the local database accordingly.

UNIT TESTS

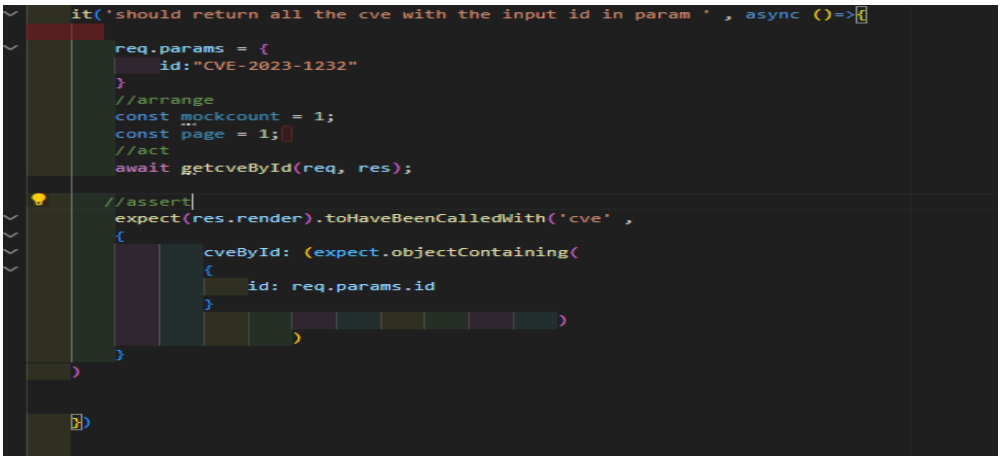
1) **Unit test case for /Cve/lists Endpoint**

This unit test validates the functionality of rendering CVE data for the input parameters passed in the CVE application. This test sets the request query parameters to include a page number, limit, year (set to "2022" in this case), lt (lesser than), gt (greater than), and the CVE ID ("CVE-2022-1234" in this case).

With the expected count of CVE entries being 1 and the count of pages being 1, the test verifies that the application renders the data appropriately. Upon executing the getListCve function asynchronously, the test verifies that the application renders the 'index' view with the expected parameters, including the count of CVEs, year, CVE ID, page number, limit, lt, gt, count of pages, and a success message.

2) **Unit test case for /cves/<CVE-ID>**

This unit test case verifies the functionality of fetching details for a specific CVE. It operates by providing the CVE ID as input through URL parameters. The expected outcome is that the returned CVE data count is 1, with only one page as result. These anticipated values serve as the basis for assertions. The test proceeds by sending a GET request to the /cves/<CVE-ID> endpoint. It then confirms that the application successfully renders the CVE data, ensuring that the displayed CVE ID corresponds accurately to the input provided endpoint.



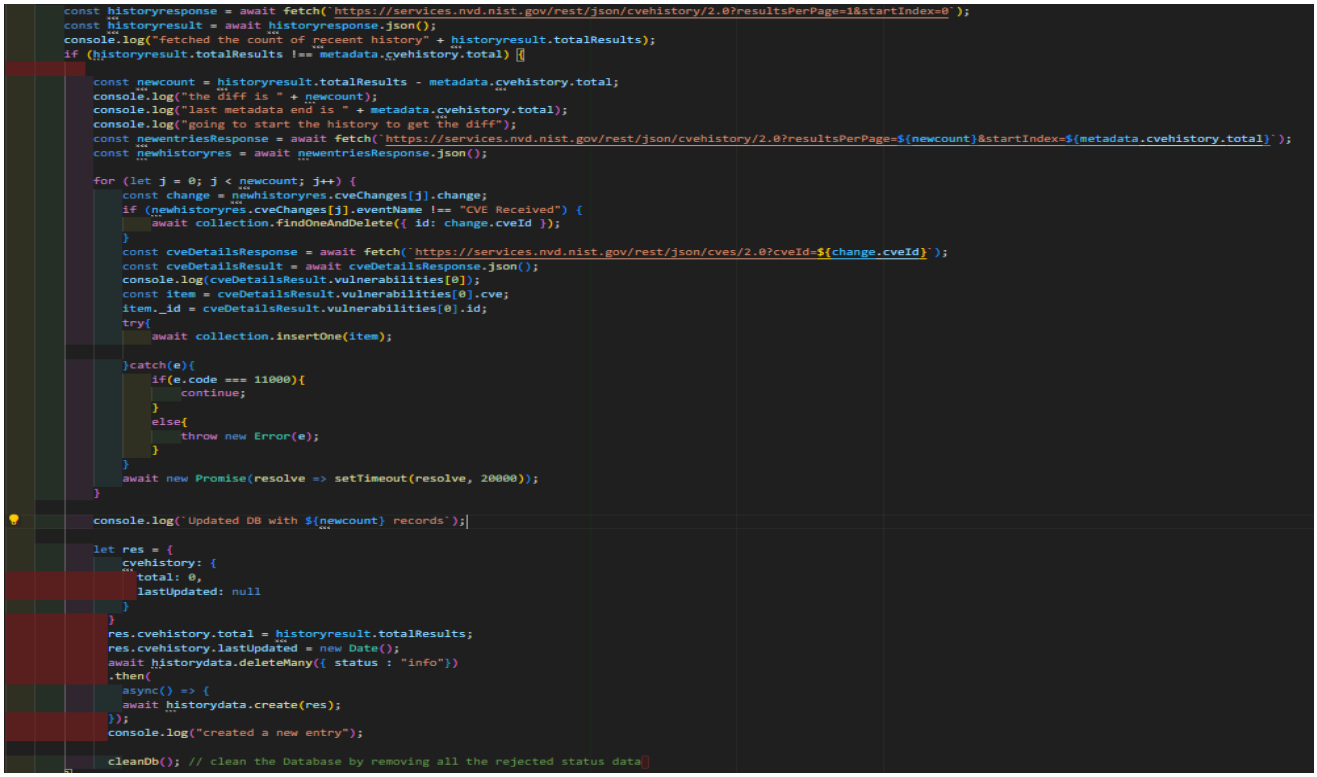
Periodic Update

The Function uses two API provided by NVD.

1) https://services.nvd.nist.gov/rest/json/cvehistory/2.0
2) https://services.nvd.nist.gov/rest/json/cves/2.0

How the update function works is explained below:

- First the total number of results returned by the history API are fetched by setting the results displayed to 1 and start index as 0.
- The fetched count is compared with the local count of the last metadata check from the history API.
- If they are equal , both local database and the API are in sync and the check is complete.
- If they are not equal , the difference between the local count and the actual count is calculated.
- Then the GET request is sent to history API to fetch the new or modified records by setting the parameters of resultsPerpage to difference value and startIndex is set to last local count.
- The new records are then iterated and corresponding cve data is extracted from the CVE API using modified CVE ID fetched from the history API and last local copy of CVE data is deleted if present and new CVE data is inserted.
- Once the record iterations are completed , DB and the API are in sync and wait for another 30 minutes.



```
const historyresponse = await fetch(`https://services.nvd.nist.gov/rest/json/cvehistory/2.0?resultsPerPage=1&startIndex=0`);
const historyresult = await historyresponse.json();
console.log("fetched the count of recent history" + historyresult.totalResults);
if (historyresult.totalResults !== metadata.cvehistory.total) {

    const newcount = historyresult.totalResults - metadata.cvehistory.total;
    console.log("the diff is " + newcount);
    console.log("last metadata end is " + metadata.cvehistory.total);
    console.log("going to start the history to get the diff");
    const newentriesResponse = await fetch(`https://services.nvd.nist.gov/rest/json/cvehistory/2.0?resultsPerPage=${newcount}&startIndex=${metadata.cvehistory.total}`);
    const newhistoryres = await newentriesResponse.json();

    for (let j = 0; j < newcount; j++) {
        const change = newhistoryres.cveChanges[j].change;
        if (newhistoryres.cveChanges[j].eventName !== "CVE Received") {
            await collection.findOneAndDelete({ id: change.cveId });
        }
        const cveDetailsResponse = await fetch(`https://services.nvd.nist.gov/rest/json/cves/2.0?cveId=${change.cveId}`);
        const cveDetailsResult = await cveDetailsResponse.json();
        console.log(cveDetailsResult.vulnerabilities[0]);
        const item = cveDetailsResult.vulnerabilities[0].cve;
        item._id = cveDetailsResult.vulnerabilities[0].id;
        try{
            await collection.insertOne(item);

        }catch(e){
            if(e.code === 11000){
                continue;
            }
            else{
                throw new Error(e);
            }
        }
        await new Promise(resolve => setTimeout(resolve, 20000));
    }

    console.log(`Updated DB with ${newcount} records`);

    let res = {
        cvehistory: {
          total: 0,
          lastUpdated: null
        }
      }
      res.cvehistory.total = historyresult.totalResults;
      res.cvehistory.lastUpdated = new Date();
      await historydata.deleteMany({ status : "info"})
      .then(
        async() => {
        await historydata.create(res);
      });
      console.log("created a new entry");

      cleanDb(); // clean the Database by removing all the rejected status data
```

FILTERING

The Search queries are passed as input to the getListCve function from the cveRouter , from the input is passed to the MongoDB query object which is passed as the input to the find function to filter the results based on search parameter passed and the return CVE data is sorted in the ascending order of the published date. All the sorting logic is done at the server side.

```
export const getListCve = async (req,res)=>{
    const client = new MongoClient(process.env.MONGO_URL)
    await client.connect();
    const database = client.db('Securin');
    const collection = database.collection('Cve_Securin');

    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    //search box queries
    const year = req.query.year || "";
    const lastModified = parseInt(req.query.lastmodified) || -1;
    const lt = parseFloat(req.query.lt) || 10.0;
    const gt = parseFloat(req.query.gt) || 0.0;
    const id = req.query.id || "";

    try {
        let query = {};
        if (year !== "" || lastModified !== -1 || lt < 10.0 || gt > 0.0 || id !== "") {
            query = {
                "published": { "$regex": year },
                "$or": [
                    { "metrics.cvssMetricV2.cvssData.baseScore": { "$gte": gt, "$lte": lt } },
                    { "metrics.cvssMetricV31.cvssData.baseScore": { "$gte": gt, "$lte": lt } }
                ],
                "id": { "$regex": id }
            };
        }

        let count = await collection.countDocuments(query);
        let countofpages = Math.ceil(count / limit);

        let Cve;
        if (lastModified !== -1) {
            Cve = await collection.find(query).limit(lastModified).sort({ lastModified: -1 }).toArray();
            count = Cve.length;

        } else {
            Cve = await collection.find(query).sort({published : 1 }).limit(limit).skip((page - 1) * limit).toArray();
        }


        res.render('index', { count,year,id, page,lastModified,lt,gt,limit, countofpages, Cve, message: "success" });
    } catch (err) {
        res.render('index' , "error occured")
```