|  |  |
| --- | --- |
| Name: | Dhwanit Pandya |
| Lab User ID: | 23SEK3324\_U34 |
| Date: | 09-01-2024 |
| Application Name: | **Damn-Vulnerable-Bank** |

**Follow the below guidelines:**





System Architecture:

(Understand the system and document the physical and logical architecture of the system, use the shapes and icons to capture the system architecture)



Hosted application using node on port: 8080

**Ubuntu 20.04** Created a Container with damnvulbanks as image

Define system’s normal behavior:

(Define the steady state of the system is defined, thereby defining some measurable outputs which can indicate the system’s normal behavior)

Basic functionality such as login, signup works perfectly. Along with these all the features for eg: Add beneficiary, view beneficiary, my balance, Transfer money, Transactions, Approve Benf, Currency Rates and Password reset

Hypothesis:

(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate for the action that will potentially have that effect on the system. For eg: "If one of our database servers fails, our service will automatically switch to a backup server, and users will not experience any downtime or data loss.")



**Known**

If there is an increase in traffic then we have logs to monitor but don’t know how will the functionality work

Every Functionality is working perfectly right from login

**Unknown**

**Unknown**

**Known**

If there is an increase in traffic then it would affect the functionality but don’t know how swiftly would it perform

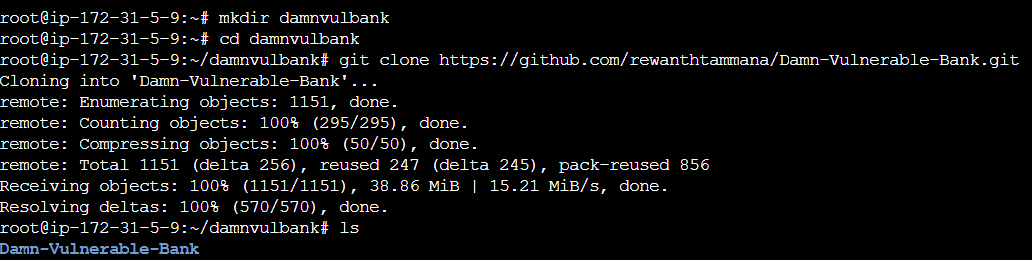
Any unknown vulnerabilities in code that might affect the availability of the application

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

Preparation:

* Launched EC2 instance on AWS with Ubuntu 20.4
* updated the machine and installed docker
* created a directory where I can clone the vulnerable repository and cloned the Damn-vulnearble-bank repo in that directory.



Implementation:

* Created a Dockerfile with contents as follows as per repo

FROM node:10.19.0

WORKDIR /home/node/dvba

COPY package\*.json ./

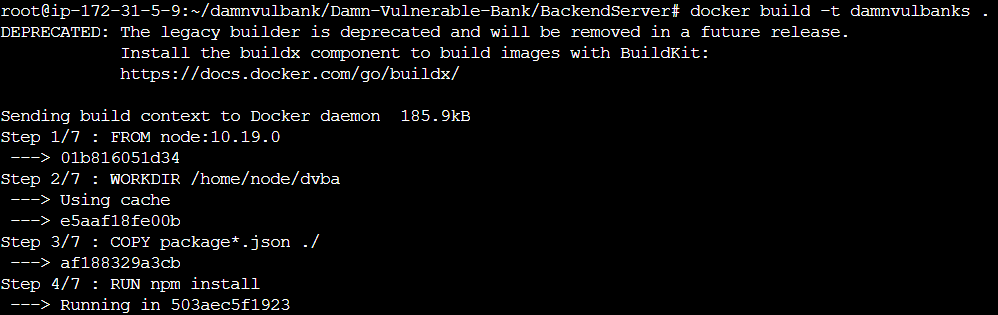
RUN npm install

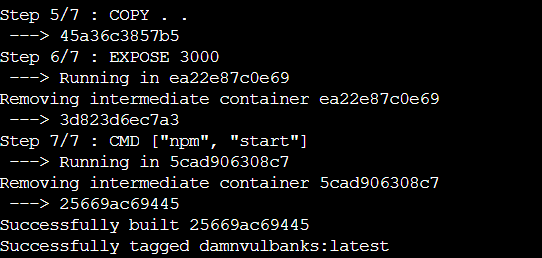
COPY . .

EXPOSE 3000

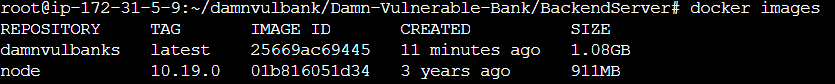
CMD ["npm", "start"]

* Executed the Dockerfile

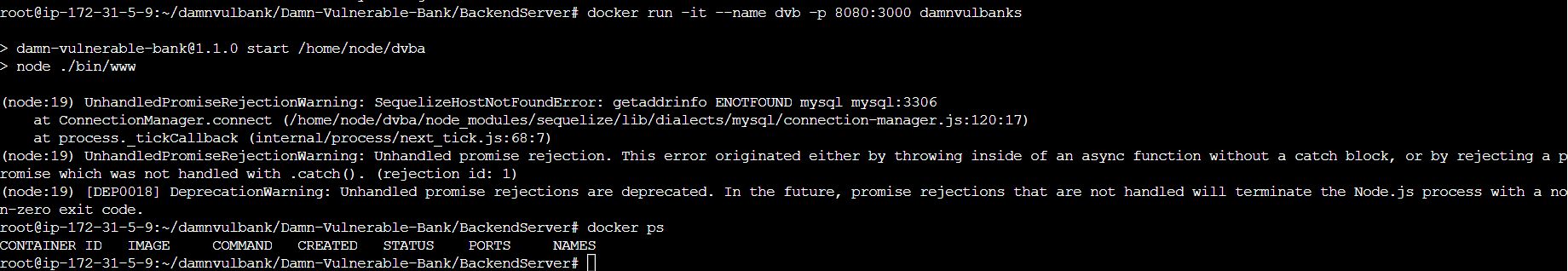




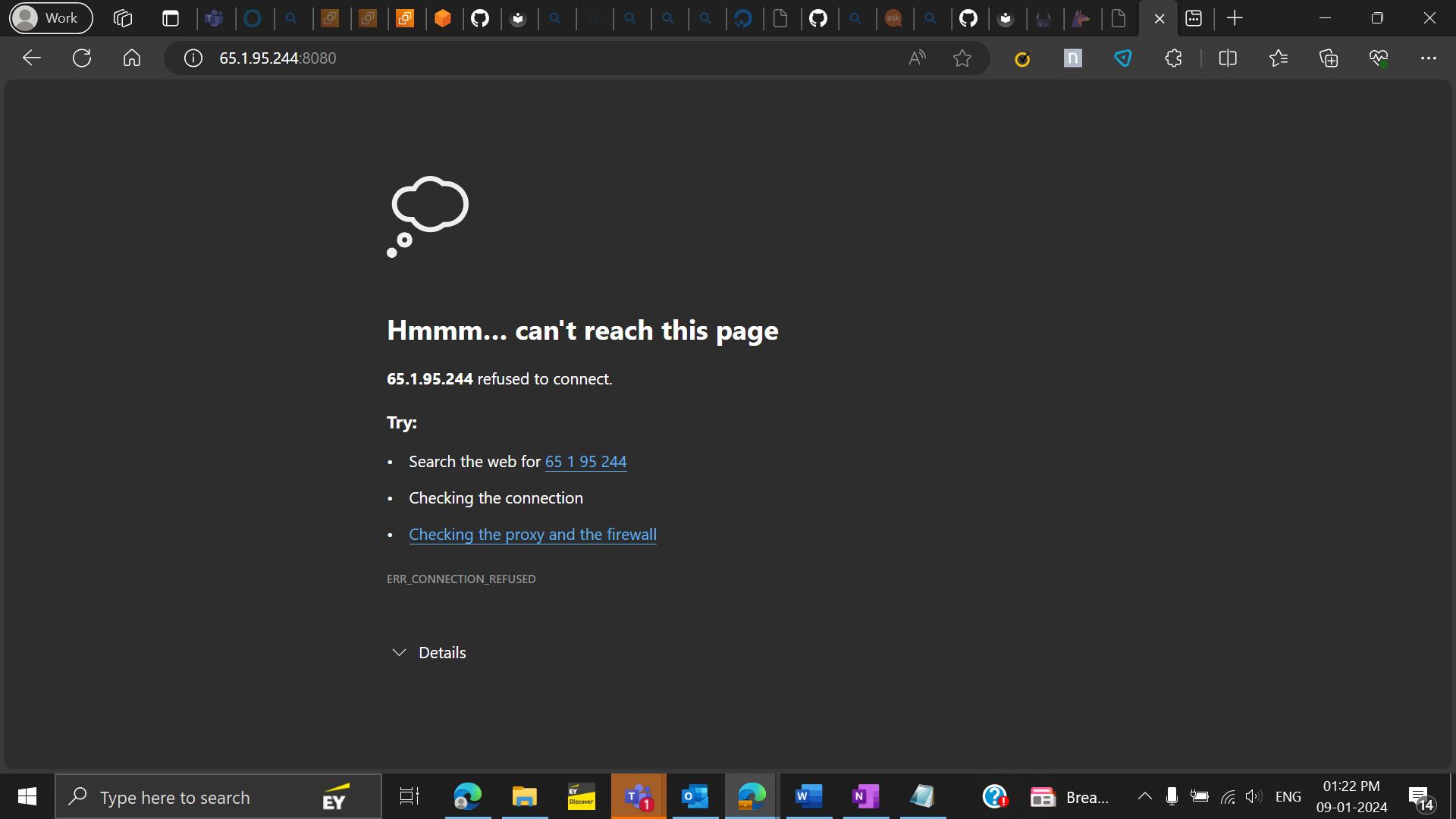
* Was able to create the image using Dockerfile



* Tried to create a container and expose it to port 8080 and using the above image ‘damnvulbanks’

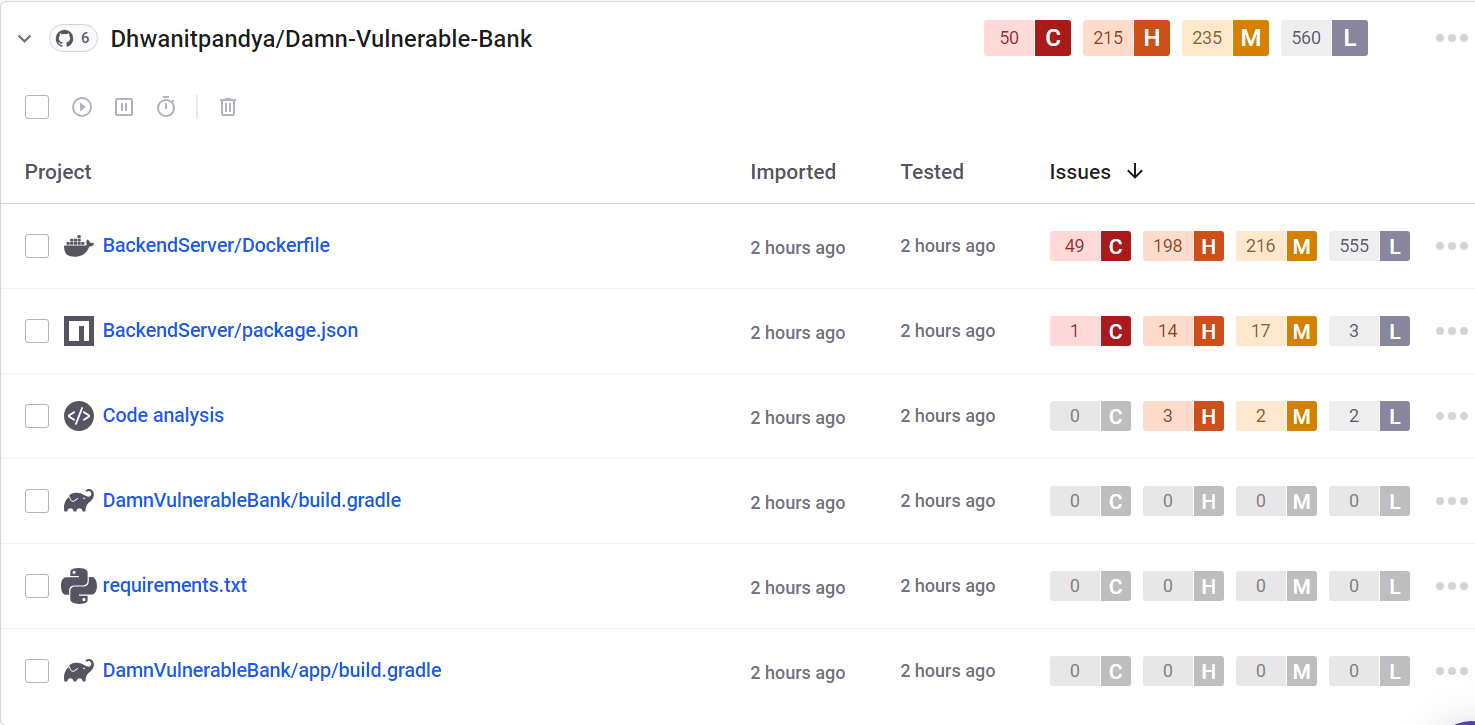


* But faced the above issue and wasn’t able to find anything on port 8080



Observation:

* Performed vulnerability analysis of the following repo using the snyk tool and found:



**Analysis:**

* Out of these Vulnerabilities some critical ones are and their fixes are mentioned below:

1. **Vul : Dpkg Directory traversal - Unintended disclosure of sensitive files**

**Impact**: By manipulating files with "dot-dot-slash (../)" sequences and its variations, or by using absolute file paths, it may be possible to access arbitrary files and directories stored on the filesystem; including application source code, configuration, and other critical system files.

**Fix**: Prevent the user-supplied directory from being higher up on the filesystem than the directory used to serve static content. Sanitize the input to prevent malicious payloads from tricking you through techniques such as URL encoding. Finally, instead of writing all the logic yourself, consider using popular open-source libraries which handle things for you.

1. **Vul : Null dereference-The dangers of accessing null objects**

**Impact:** it can cause the program to crash or behave unexpectedly, potentially leading to security issues.

**Fix**: have the requests.extract\_json().then().get() handle the JSON values instead of passing a pointer to that function.

1. **Vul : sequelize SQL Injection**

**Impact:** SQL injection allows the attacker to read, change, or delete sensitive data as well as execute administrative operations on the database.

**Fix**: Prevent user-supplied input from affecting the logic of the executed query. Using parameterized queries which almost all SQL libraries support. In a parameterized query, the input is "escaped" and handled literally.

1. **Vul : ansi-regex-Regular Expression Denial of Service (ReDoS)**

**Impact:** ReDoS attack attempts to slow down or even render an application unavailable. processing of the malicious string exhausts the computing power or memory available, thus impacting the application's performance and, in certain circumstances, causing a denial of service (or DoS).

**Fix**: Avoid using regex for user input validation. Closely review and analyze all patterns before implementation to ensure they do not contain any evil regex patterns.

1. **Vul : Improper Privilege Management**

**Impact:** threaten your organizational security, compromising sensitive data, granting privileges to individuals who can abuse them, and more.

**Fix**: giving users permissions to do specific things, and no more. Integrate security tools directly into your SDLC, read about vulnerabilities that could affect you, seek advice from security professionals (ideally an internal security team, but also outsourcing when necessary).