LINK: https://github.com/wald3r/cybersecurity_project

Installation:
1) Clone git repository

1) Install nodejs
2) Change to /app
3) Execute „npm install“
4) Execute „npm update“
5) Execute „npm start“
6) Call app via http://localhost:3003
7) username: „test“ and password: „test“

About the Application:
The application is called SecureBox and is supposed to imitate Dropbox in a simple way. It is possible to upload/download files, register and login user, edit profiles and so…
For the sake of security, I have removed the code with the upload function.
In the repository, there can also be found screenshots, which provide solutions for the vulnerabilities.

FLAW 1: Brute-force attack (A2)
The application is vulnerable towards brute-force attacks. When you intercept the login request with, for example, theZAP-Tool from OWASP it is easy to figure out that for the user: „test“ the password is „test“.
To secure the application there are several approaches possible offered by the nodejs community. So there is no need to invent something on your own. After trying several packets, the most simple and easiest is the „express-rate-limit“ packet, which does nothing else than limiting the number of requests per ip for a certain amount of time. So it is possible to limit requests in general per ip or for specific routes. Which means, that the address /*api*/login must get limited to fix this flaw.
If we do this, we can also limit the requests for /*api*/registration to avoid that people register too many users in a row.

FLAW 2: Role Management - A5
Currently, it is possible for every user which is logged in, for example, with user „test“ and pw „test“, that they can see the admin area. Reachable via „localhost:3003/admin“. The console contains information about all registered user, so loads of private information which are not meant for everybody. To fix this vulnerability a proper user role management is needed. A simple way to do this, is to add to the user model an extra field and always check for it in the frontend, whenever you want to access critical data. Although this can be easily disabled, so it is important to verify the role of the user in the backend.

FLAW 3: Encryption - A3
We know that this app is meant to be some sort of dropbox clone. So people will upload sooner or later files which contain confidential information. It is a necessity that the backend secures such files with an encryption algorithm and only the user which have uploaded the file, can decrypt it with a certain password. So the idea is to implement a symmetric cryptosystem.

The ‚crypto' and ‚fs' packets can be used to fix this flaw. With the ‚fs' packet, it is possible to operate with files and create read and write streams. With such streams, we can encrypt/decrypt every single byte of a file. As encryption algorithm ‚crypto' offers a wide range of algorithms. For testing, I used the „aes256" - algorithm, which is the advanced encryption algorithm with a 256-bit key. To make sure that the password is always 256-bit long, a 256-bit hash creator will do the job.
The only problem with this solution is that the uploaded file has to be stored first unencrypted so that it is possible to create read and write streams. After encryption, the unencrypted file can be deleted. Which is the reason why there might be a security leak regarding the problem with data life time. The unencrypted file might be still somewhere for a certain amount of time. Although after loads of research there are not many encryption/decryption examples for files found online. So for now this solution has to be sufficient.

FLAW 4: Authentication - A5
Another problem with this application is that you can enter some of the URLs directly and through that, you can simply skip authentication. For example:

http://localhost:3003/api/users

*api/*users is the backend and its response contains all registered users.
Best way to fix this vulnerability is to generate a token, which gets transported from the frontend to the backend with every request. The backend then verifies every request. If there is no token, no data gets transmitted to the frontend. Through this, it is possible to secure data from users without authentication.
A big problem is now, that it is highly insecure to transport the token via HTTP requests. To use solely HTTPs would get mandatory. Although to achieve this is a bit more complicated. An easy workaround is to deploy the app only to Heroku, which redirects everything from HTTP to HTTPs automatically.

The link http://localhost:3003/api/files can serve as an example for a secured request.

FLAW 5: Proper logging - A10
A common problem is insufficient logging. In this case, the application barely logs anything. There are a few lines of code where some sort of logging happens via „console.log(message)" but nothing consistent, which is clear and easy to understand. With such kind of logging, it is easy to overlook attacks and flaws.
Luckily nodejs offers many different kinds of packets to implement a logger. So there is no need to build your own. For example, the „simple-node-logger" is easy to use and to customize.
To start it is wise to log all HTTP requests, which the backend receives. More importantly every login attempt, to spot brute force attacks, and registrations. All database operations, no matter if they fail or not are important to get logged. Distribute the logger over the backend and frontend to achieve a consistent logging.