A6: Security Misconfiguration +
A9: Components with Known Vulnerabilities

1 A6, Using default settings

This link points to an extremely simple Single Page Application, providing a login page and, after a successful login, a page using a single protected REST-endpoint.

a) See whether you can discover the following properties of the application (not all are necessary security-problems). Use the GUI provided by the application (as a start), Postman, nmap and obviously your browser's Developer Tools, when probing the app:

- OS - Windows
- Server Architecture (Come up with a "guess" and provide arguments for your suggestion) - React back and frontend, analyze network in developer toold (react_devtools_backend.js file)
- Server(s) surge (deployed)
- Programming Language - Java-script
- Important packages, classes used by the Programming Language
- Can you see "what kind of pages" logged-in users will see, without having a way to log in?
- Can you discover the client technologies used Mozilla, Chrome
- Default users and Passwords = the ability to login[1] jwttoken
- If you can make a successful login, can you: discover the algorithm used to "protect" the token, the lifetime of the token, the role, assigned to you by the system? hashed
- How/where is the token stored by the client local storage - jwttoken
- Can you determine/guess(must be qualified) whether front-end, REST back-end and Database is running on the same or on different servers? Different, Surge dont handle backend or database. *You are hereby granted permission to scan the server hosting the BACKEND*
- Can you determine which database is used by the backend? mysql
- Have you discovered any unnecessary features which are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges)
- Who owns the domain used for the server? Lars (iplookup)
- Is the server hosted "privately", by a cloud provider, or .....? (digital ocean)
- ... Can you detect/discover more properties of the application than those suggested above?

Extra (leading to the next exercise):

- Open developer tools, and the network-tab. Enter this URL (**exactly** as given) http://studypoints.info
  Explain the first two requests, you monitor. Is this a problem, could this have been done better" (this probably require that you have read the suggested readings related to security-headers)

b) List all the things "done wrong" (misconfigured) in this application

[1] Assume that, using the initial observations, you could guess that this code was designed using a "known" seed (If not assume that you would always try with "easy" username, passwords)

2 A6, Security Headers

a) Enter this link http://securityheaders.mydemos.dk/ in a browser, and explain shortly the purpose of all response headers, related to security.
Note: This will be a class exercise, for those of you who attend the class ;-)

b) Setup a simple web server (Tomcat, Nginx, Express ....) which should set most of the security headers for **all** requests.

This is probably simpler than what you might think. With node + Express, you can create such a server in less than six lines of code ;-)
If you are using Tomcat (or Express), behind a Nginx-reverse proxy, you can do this either in your code (Tomcat, Express) or, let Nginx add the headers.

3 A9 Components with Known Vulnerabilities

How can we (you) ensure that our maven dependencies do not contain Known Vulnerabilities?

a) Google this topic an see what kind of tools you can find, the following are suggestions:

- https://snyk.io/vuln/?packageManager=all
- https://www.owasp.org/index.php/OWASP_Dependency_Check

b) Use one or more of the tools/strategies found above and use them to check some of your previous Java/Maven projects (for example the backend seed from your 3. Semester CA3)

c) If you are following Python or JavaScript come up with a similar strategy for Python/JavaScript dependencies