Mange tig der skal konfigurere når vi opsætter en applikation. Eks. Da vi skulle sætte vores droplet op med Tomcat og SSL. Man kan let komme til at efterlade bagdøre åbne for hackere at misbruge.

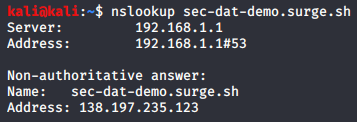
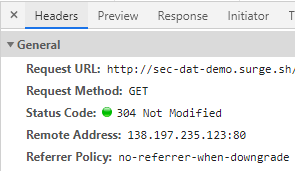
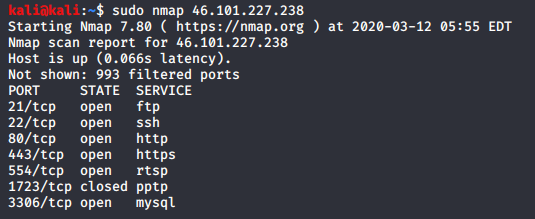
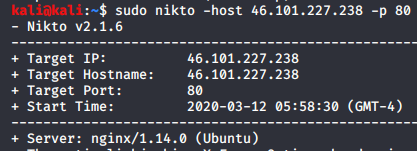
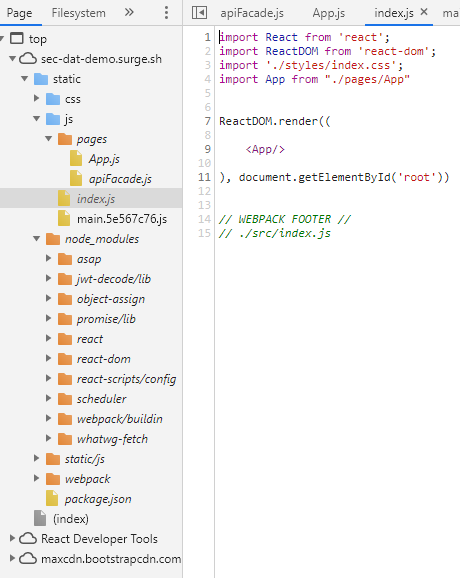
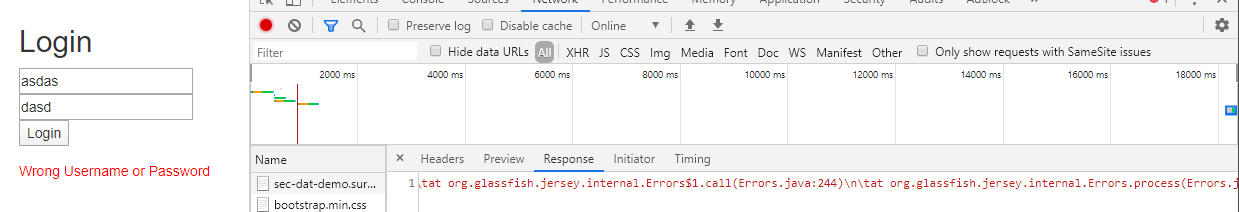
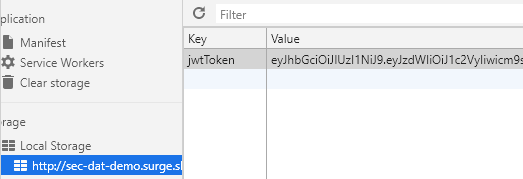
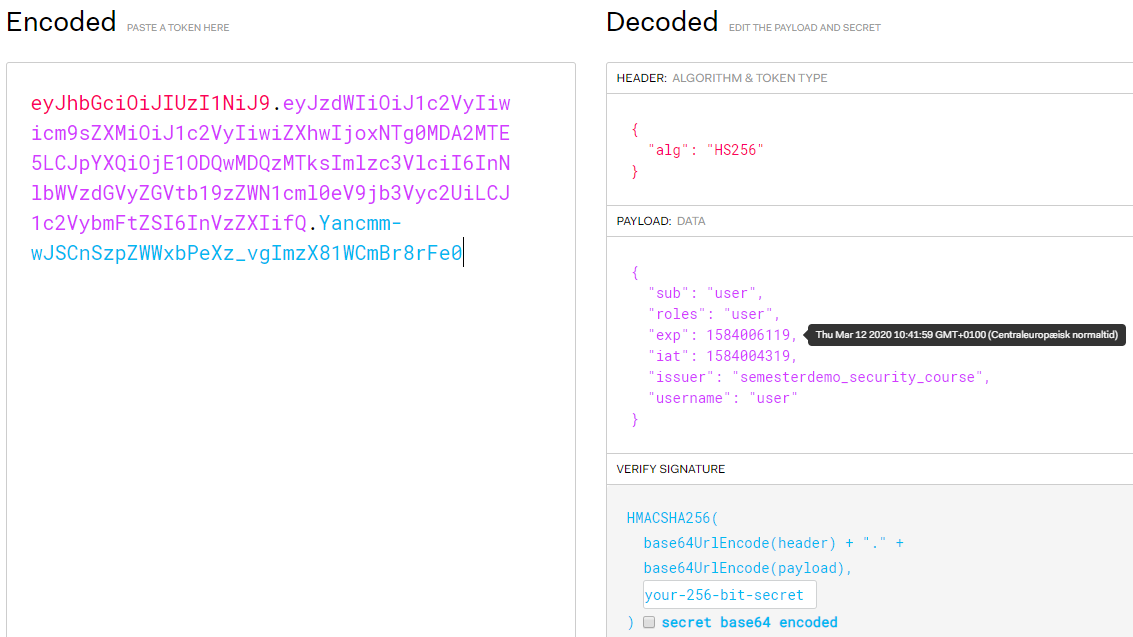
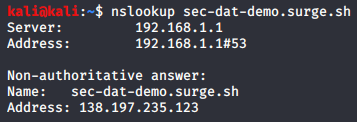
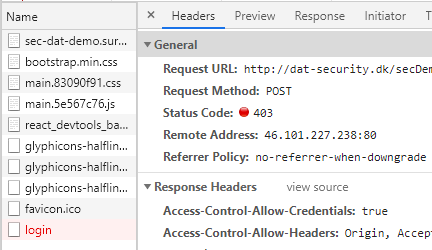
The application might be vulnerable if the application is:

* Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.
* Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
* Default accounts and their passwords still enabled and unchanged.
* Error handling reveals stack traces or other overly informative error messages to users.
* For upgraded systems, latest security features are disabled or not configured securely.
* The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
* The server does not send security headers or directives or they are not set to secure values.
  + Modern browser headers
  + Packet called helmet
* The software is out of date or vulnerable (see [A9:2017-Using Components with Known Vulnerabilities](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities)).
  + We have typically seen warnings on our GitHub repos last year when we used some of our start-code containing outdated dependencies.

1 A6, Using default settings

This [link points](http://sec-dat-demo.surge.sh/) to an extremely simple Single Page Application, providing a login page and, after a successful login, a page using a single protected REST-endpoint.

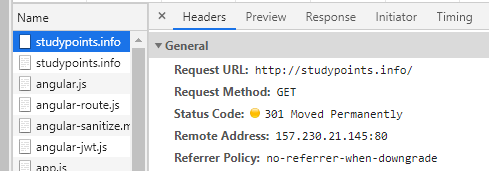
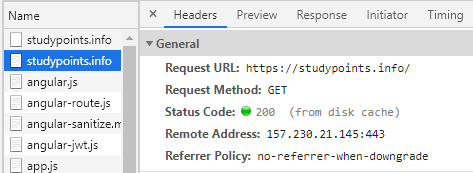
**a)** See whether you can discover the following properties of the application (not all are necessary security-problems). Use the GUI provided by the application (as a start), Postman, nmap and obviously your browser's Developer Tools, when probing the app:

* OS
  + Vi kender domæne navnet og vi kan bruge nslookup til at finde IP’  
      
    Eller i browseren:  
    
  + Responce header i browseren sider at den køre. Linux/Ubuntu  
    
* Server Architecture (Come up with a “guess” and provide arguments for your suggestion)
  + Vi kan se at frontenden ligger på en surge server og at der bruges NginX som reverce proxy
  + Lavede et forkert login og fik en tomcat fejl fra en backend server med addressen 46.101.227.238
  + Ved at scanne med nmap på backenden 46.101.227.238 kan vi se de porte der er åbne på serveren.  
    
  + Af dem er port 80 interessant da der her ligger noget vi kan scanne.
  + Vi kan med nikto se at serveren køre nginx 1.14 ubuntu på port 80  
    
  + På port 3306 køre der mysql
* Server(s)
  + - Lunux Ubuntu
    - Tomcat
    - Nginx
    - Mysql
* Programming Language
  + Frontend er lavet i React som en spa – JavaScript applikation. Man kan ikke skjule JavaScript i frontenden da det er det som selve browserne bruger.  
    
  + Backend er lavet i Java. Dette kan vi se ved at den fejl vi fik i browseren da vi prøvede at logge ind kom fra en GlassFish pakke med en række java exceptions.
  + 
* Important packages, classes used by the Programming Language
  + Vi kan se at applikationen bruger JAX-RS med restfull recources til at kommunikere sin front end med sin backend.
  + Ud over dette har vi observeret at der køres nogle Glassfish pakker backendens Java, og at der køre React i frontenden bruges JWT-tokens og React-Router-Dom.
  + 
* Can you see “what kind of pages” logged-in users will see, without having a way to log in?
  + Vi kan kigge i javascript på klienten.
  + Vi kan se at der også bruges React Router
* Can you discover the client technologies used
  + React -> node\_moduels
* Default users and Passwords = the ability to login
  + Der blev brugt en default user med user, test
* 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?
  + Vi kunne se at der bruges JWT til at store lokalt i klienten  
      
    Ved at logge ind med den default user user, test som vi fandt, kan vi se hvor denne token bliver gemt.  
    
  + Ved at decode vores token kan vi se hvor lang tid den der valid  
    Vi kan tage denne token fra local storrage, og copy paste den ind på siden jew.io der kan decode den fra base64. Hvis der er en hacker der får fat i denne token, kan personen bruge den ind til den timer ud. Han behøver ikke engang at bruge en klient, men kan i stedet bare bruge postman for eksempel. Man kan dog ikke ændre i denne token da serveren også holder styr på at det er den rigtige token gennem en HS256 hash algoritme som den sammenligner på.  
    
* How/where is the token stored by the client
  + The token is stored on the browser’s local storage.
  + The token is valid for 30 minutes.
* Can you determine/guess(must be qualified) whether front-end, REST back-end and Database is running on the same or on different servers?
  + Ved at bruge nslookup kan vi se hvem der ejer domænet og hvad IP-en er.  
    
  + Ved at skrive domænet fra front enden ind kan vi finde front end IP’en som er 138.197.235.123 og se at den er deployet på surge.
  + Hvis fejler i at logge ind kan vi se rød responce header der kommer fra backend serveren og derved finde IP’en til vores backend som er 46.101.227.238 .  
    
  + By looking at the first response from the frontend upon rendering the page, we can see the frontend IP is 138.197.235.123
  + By looking at the response of a failed login we can see that the response comes from a backed at the IP 46.101.227.238

*You are hereby granted permission to scan the server hosting the BACKEND*

* Can you determine which database is used by the backend?
  + Hvis vi burger nmap, kan vi se at der på port 3306 køre en Mysql server. Denne port burde være lukket for udefrakommende.
* Have you discovered any unnecessary features which are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges)
  + Mysql på port 3306 brudte for eksempel ikke være åben da den helst kun skulle kunne tilgås fra en bestemt applikation og måske endda kun same-origin.
* Who owns the domain used for the server?
  + Hvis vi Googler whois, kan vi finde ud af hvem der ejer serveren og hvem der hoster dat-security.dk.  
    
* Is the server hosted “privately”, by a cloud provider, or …..?
  + Som vi tidligere erfarede er frontenden hostet på Surge som provider.
  + Hvis vi kigger nærmere på whois kan vi se at navneserveren er fra digitalt ocean, som er dem der hoster denne server som provider.
* … Can you detect/discover more properties of the application than those suggested above?

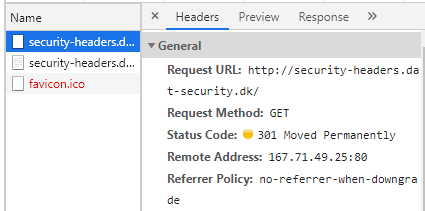
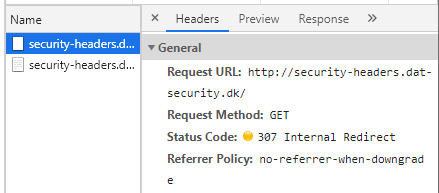
Extra (leading to the next exercise):

* Open developer tools, and the network-tab. Enter this URL (**exactly** as given) [http://studypoints.info](https://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)
  + We start by trying to access through http, and we are then redirected to https.  
    
  + Det er Nnginx der sørger for at redriecte os fra http til https.  
    
  + Overgangen mellem redirectet fra http til https kan dog blive misbrugt af hackere i det korte øjeblik inden der bliver redrected, da forbindelsen på dette tidspunkt ikke er sikker.

**b)** List all the things “done wrong” in this application

* Vi bliver ikke automatisk omdirigeret til https, men får i stedet lov til at fortsætte på en usikker http linje. Selv hvis vi blev redirectet gennem Nginx reverce proxyen, kunne en hacker stadig misbruge det mellemrum der ville være mellem redirectet fra http til https.
* Server arkitekturen er alt for åbenlys og vi kan se hvilke servere det køre, relativt nemt blandt andet Mysql.
* Vi kan få fat i port 3306 til MySQL som normalt altid brudte være lukket af, da den kan misbruges.
* Vi ved at serveren køre Java, og at der bruges Tomcat, hvilket vi kan finde ud af ved at se på de Glassfish pakker som kaster Java exceptions ud til responset når vi fejler i at logge ind. Normalt ville man ikke give hackere en ide om hvad man køre inde bag, bestemt ikke når det involvere pakker.
* Vi kan ved at kigge i frontendens kode meget nemt se hvilke sider vi har adgang til og hvilke sider der eller findes i applikationen. Dette kan ikke undgås, men kan gøres mine åbenlyst.
* Hvis vi logger ind, kan vi også meget let få fat i og misbruge den JWT-token ligger lokalt, og som vi let kan få oversat da den kun er base64 encoded. Hvis vi fik fat i denne udefra kunne vi i princippet tage kontrol over sessionen.

Exercise 2 Security Headers

* Hvis man tilgår siden første gang, får men en 301 moved permanently redirect fra http til https. Ved dette meget korte besøg på http han der nå at ske en masse snavs for hackerne at misbruge.  
  
* Hvis vi tilgår siden anden gang for vi en kode 307 internal redirect, hvor browseren overhoved ikke når at tilgå http, men bare videresendes til https internt i browseren. Man får her ikke lov til at være kortvarigt på http, men sendes med det samme til https.  
  

HTTP Strict Transport Security (HSTS)  
HSTS addresses the following threats:

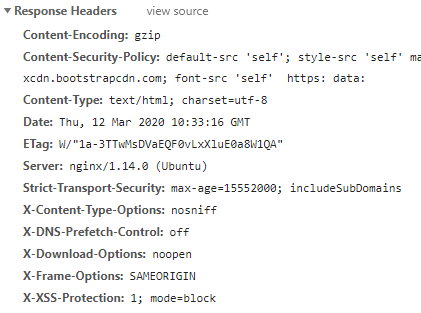
* User bookmarks or manually types <http://example.com>  and is subject to a man-in-the-middle attacker
  + HSTS automatically redirects HTTP requests to HTTPS for the target domain
* Web application that is intended to be purely HTTPS inadvertently contains HTTP links or serves content over HTTP
  + HSTS automatically redirects HTTP requests to HTTPS for the target domain
* A man-in-the-middle attacker attempts to intercept traffic from a victim user using an invalid certificate and hopes the user will accept the bad certificate
  + HSTS does not allow a user to override the invalid certificate message

Når man er blevet registered som at have besøgt en hjemmeside som https en gang kan vil man altid blive redirected til https siden frem for nogen http side.

Public Key Pinning Extension for HTTP (HPKP)  
Bruges ikke så meget da denne header typisk er til mere besvær end godt.  
Kort sagt:  
The HTTPS web server serves a list of public key hashes, and on subsequent connections clients expect that server to use one or more of those public keys in its certificate chain.

X-Frame-Options  
The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe> or <object>.

Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites.  
  
The added security is only provided if the user accessing the document is using a browser supporting X-Frame-Options.

X-Frame-Options kunne tidligere misbruges af hackere til at plasere et skjult vindue for at snyde brugeren til at tro at de navigere en anden side en de faktisk gør.  


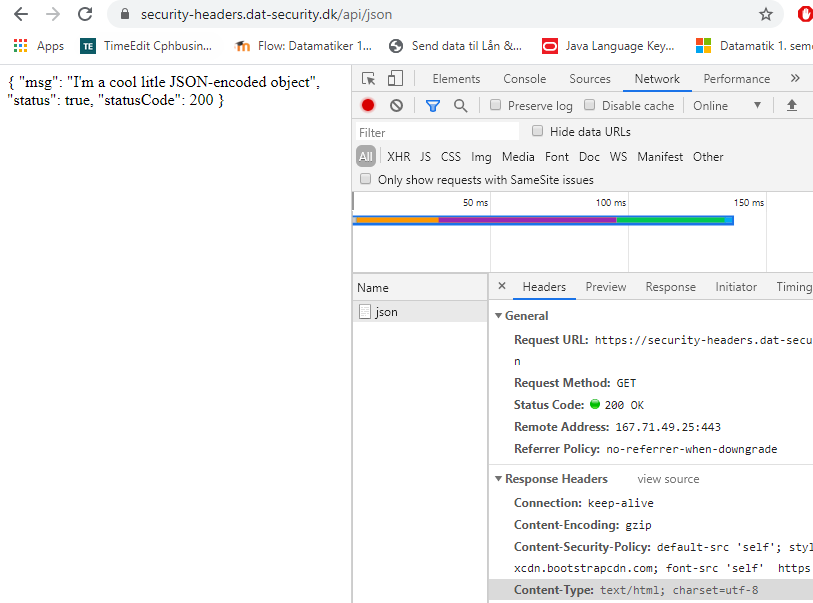
x-Frame-Options: SAMEORIGIN – sikre at man kun kan bruge en x-frame hvis man er fra samme server.

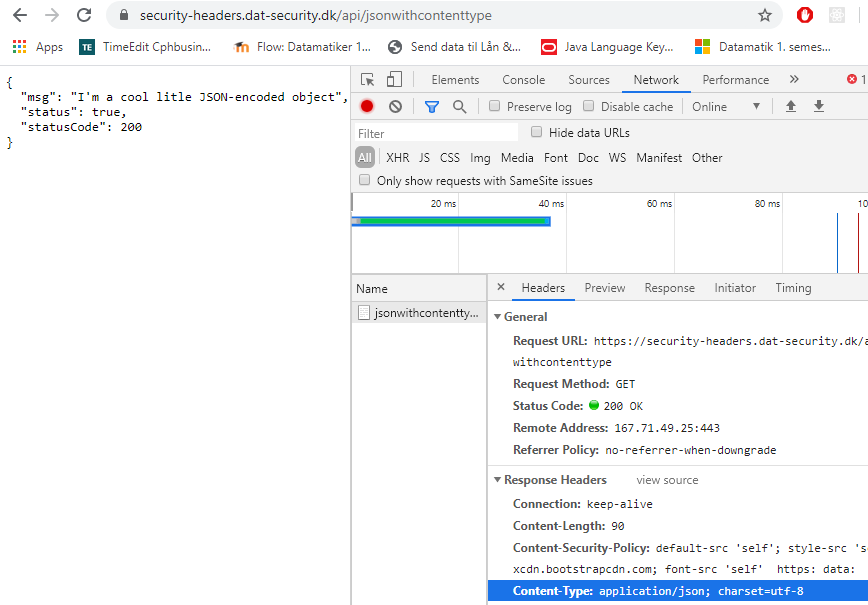
X-XSS-Protection  
The HTTP **X-XSS-Protection** response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks.

Although these protections are largely unnecessary in modern browsers when sites implement a strong **Content-Security-Policy (see later)** that disables the use of inline JavaScript ('unsafe-inline'), they can still provide protections for users of older web browsers that don't yet support CSP

X-Content-Type-Options  
A nifty attack known as **MIME type confusion** was the reason this header was created. Most of the browsers employ a technique called MIME sniffing, that consists on taking an educated guess at what the content type of the server response is, instead of trusting what the headers content type value says.

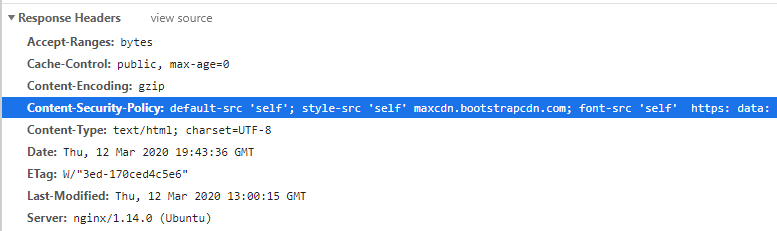
Under certain circumstances, browsers can be tricked into making the incorrect decision, allowing attackers to execute malicious code on victim’s browsers. +

Browsere fortolker …  


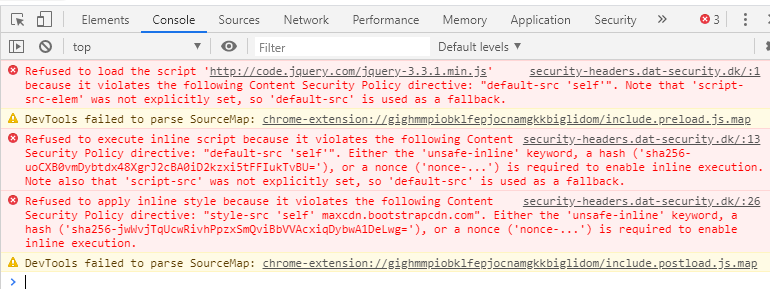
  
Hvis man glemmer at sætte content type, vil browsere ofte forsøge at gætte hvad content type er, hvilket hackere kan misbruge til at få sniffet noget ind som de ikke brudte. Her kan man ved brug af X-Content-Type-Options undgå dette ved at begrænse hvad der content type kan være.

Content-Security-Policy  
A **Content Security Policy (CSP)** requires careful tuning and precise definition of the policy. If enabled, CSP has significant impact on the way browsers render pages (e.g., inline JavaScript disabled by default and must be explicitly allowed in policy).   
  
CSP prevents a wide range of attacks, including Cross-site scripting and other cross-site injections

En header hvor man kan fine-tune hvad klienten egentlig må.



I tilfælde af test applikationen bruges der en Content Security Policy med default på self, hvilket diktere at der ikke må køres noget udefrakommende kode eller scripts på applikationens source kode. Dette kan vi se i aktion ved at iagttage konsollen, hvor både scripts og Bootstrap bliver refused ved render:



3 A9 Components with Known Vulnerabilities

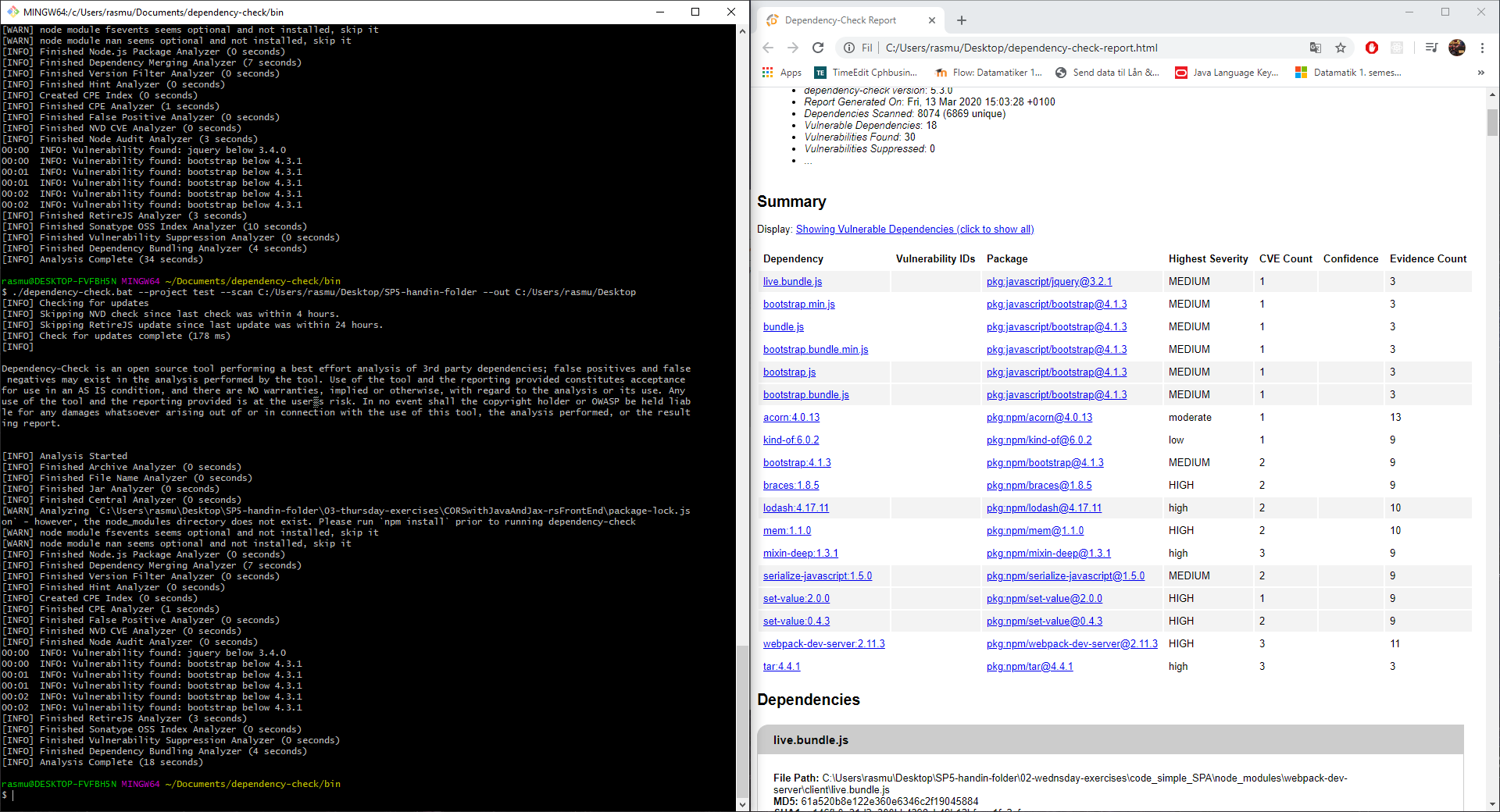
How can we (you) ensure that our maven dependencies do not contain Known Vulnerabilities?

* We can use dependency checking tool to ensure that are dependencies and packages are up to date and not vulnerable for exploits.

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

* <https://snyk.io/vuln/?packageManager=all> Commercial service that focuses on JavaScript npm dependencies.
* <https://www.owasp.org/index.php/OWASP_Dependency_Check> Dependency-check is an open-source command line tool from OWASP that is very well maintained.
* <https://nodesecurity.io/> Node.js modules and NPM dependencies.
* <http://retirejs.github.io/retire.js/> JavaScript-specific dependency checker.
* <https://ossindex.net/> Extracts dependency information from NPM, Nuget, Maven Central Repository, Bower, Chocolatey, and MSI (which means it's covering the JavaScript, .NET/C#, and Java
* GitHub also has its own building dependency checker, which we have used frequently in relation to prior projects.

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)

* Here is an example with the GitHub active dependency check (dependency-bot), where it has found an Cross-Site Scripting vulnerability in my SP5 assignment.  
  
* Here is an example using the OWASP Dependency-check command line tool on the very same SP5 assignment. By running ‘./dependency-check.bat --project test --scan C:/Users/rasmu/Desktop/SP5-handin-folder --out C:/Users/rasmu/Desktop’ OWASP Dependency-check will scan the project folder for vulnerable dependencies and provide detailed descriptions in a html file. If we look closely, OWASP Dependency-check will actually have discovered the same serialize-javascript:1.5.0 vulnerability as the GitHub dependency-bot, and a lot more. It found about 30 vulnerabilities.  
  

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

* We can use the OWASP Dependency-check tool on Python too since it has support across many languages and packages. We can scan with ‘./dependency-check.bat --project test --scan C:/Users/rasmu/Desktop/Python/Week-10/06-4\ Concurrency\ vs\ Paralelism --out C:/Users/rasmu/Desktop’. Since we are not using that many exploitable packages in out python classes, the scanner did not find and vulnerabilities.  
  