Token baseret IT-sikkerhed

SECURITY SPRING 2020

GitHub: https://github.com/cph-ms782/resetPasswordProject
Deployed: https://sec.sandersolutions.dk/

Dennis Hansen CPH-DH153
Johan Christian Ryge CPH-JR221
Martin Bøgh Sander-Thomsen CPH-MS782
Sercan Atici CPH-SA268

Indhold

Introduktion

I denne rapport adresseres tokens i forskellige former. Som første led i rapportens tilblivelse havde vi forberedt et oplæg til rapportens fokus som præsenterede en måde at bruge tokens til at gennemføre en specifik handling ud fra et sikkerhedsmæssigt synspunkt - det at få skiftet kodeord. Vi kom i denne forbindelse ind på, at vi ville gennemføre det manuelt ved selv at holde styr på alle aspekter i processen.

Dette fokus blev dog til en start ikke endegyldigt godkendt som et emne til projektet, men endte med at blive betinget godkendt, såfremt vi kunne inkludere en vinkel på emnet, der ikke bestod af et tredjepartsmodul, der gennemførte det hele. Eller på betingelse af vi kunne bruge et tredjepartsmodul til at holde en autorisering af en brugers gøren på en hjemmeside i gang, så længe vedkommende bruger siden. Altså at siden ikke udløber, mens brugeren foretager handlinger.

Herefter blev det diskuteret hvorvidt vi burde opgive reset password projektet, men her opnåede vi enighed om at vinklen vi havde identificeret, var brugbar, og at vi også ville lave en refresh token-del til at holde en session kørende. Dette bevirkede dog at fokus skiftede mod refresh token, og emner som “Webtokens”, ”prepared statements”, sikring af mysql, sikkerheds headers, og krypterede kodeord ikke kom med i samme form. Disse emner beskrives i kommende afsnit.

Web tokens er en del af fagets pensum, og man kan argumentere for at ordet passer til det som nævnes i rapporten, men vi mener det som at være JWT tokens, og derfor en inkludering af fagets pensum. I projektet benyttes tilfældige strenge af karakterer som tokens, udført af tredjeparts programmer. Disse er kun brugbare af vores server og klient.

Prepared statements er ikke inkluderet i indeværende rapport. Koden indeholder et sted, benævnt TODO, hvor det var muligt at inkludere ’prepared statements’, men da det ændrede fokus valgte vi ikke vi at gøre noget her. Man kan argumentere at React JS har indbyggede ’prepared statements’, men da vi ikke har testet dette, har det ikke været muligt at adressere det i rapporten.

Sikring af mysql er ikke inkluderet. Sikkerheds-headers er med i den form at tredjeparts middleware helmet er aktiveret. Derudover er brugerens bemyndigelse (authorization) af brug af en hjemmeside kørt udelukkende gennem respons headers.

Vedrørende krypterede kodeord er kryptering slået fra, så kodeordet kan aflæses i vores testbed. Det er beskrevet i koden, hvordan krypteringen skal laves, samtidig med det nævnes at det naturligvis er helt forkert ikke at kryptere.

Vi diskuterer mulighederne for at holde en session kørende, og i det fokus dykke ned i de forskellige typer af tokens der her benyttes. Det diskuteres om sikring af sine tokens og besværligheder forbundet hermed, og at det muligvis er en bedre metode at gemme en brugers token på en server, og om hvordan man manuelt kan holde styr på en brugers tokens (tokens er dog lavet af tredjeparts programmer). Sidstnævnte er dog kun beskrevet, som en del af processen under Brugseksempler og under konklusion.

Tokens

Indenfor IT-sikkerhed er det umuligt at undgå at adressere emnet om tokens. Tokens defineres som et objekt der repræsenterer retten til at udføre en given handling på et givent tidspunkt.
Et token kan være mange ting. Det kan eksempelvis være et invitations-token, en tilfældig række af tal og numre, der giver en bruger lov til at udføre en specifik handling. En token kan også være et såkaldt "sessions-token", et session ID eller ID Token, der gør det muligt at gennemføre handlinger, hvor det kræves, man er autentificeret. Slutteligt, kan tokens tage formen af et såkaldt "Access token" der kan indeholde sikkerhedsoplysningerne for en login-session og identificere brugeren, brugerens grupper og brugerens privilegier.
Grundlæggende er der to hovedtyper af tokens der er relateret til identitet: 1) ID Tokens og 2) Access Tokens. Begge vil gennemgås i de følgende afsnit.

ID Tokens

Et ID token er et JSON Web Token som applikationer benytter til at personliggøre brugeroplevelsen. Et sådan token bør kun indeholde ikke-følsomme data, som f.eks. brugernavn og billede, idet det er relativt ubesværet at få adgang til via 'middleman attacks' – et emne som bearbejdes senere i denne rapport. ID tokens bør heller ikke bruges til at få adgang til et API.
ID Tokens er JSON Webtokens (JWT'er), der udelukkende er beregnet til anvendelse af applikationen til at verificere brugeren. Hvis der eksempelvis er en applikation der benytter Google til at logge brugere på og til at synkronisere deres kalendere, sender Google et ID-token til den applikation der indeholder oplysninger om brugeren. Applikationen analyserer derefter tokenets indhold og bruger informationen, inklusive detaljer som brugernavn og profilbillede, til at tilpasse den samlede brugeroplevelse.

ID-tokens bør *ikke* bruges til at få adgang til et API. Enhver token indeholder information til det tilsigtede *audience*, aud, eksempelvis en klient applikation. Hvis dette ikke er tilfældet, skal man ikke stole på tokenet. Omvendt forventer et API, at et token med en aud værdi svarer til API'ets unikke ID. I henhold til OpenID Connect-specifikationen skal et *audience* til ID-token'et, som angivet af aud kravet, være client ID for den applikation der anmoder om godkendelse. Hvis dette ikke er tilfældet, skal man ikke stole på token'et. Omvendt forventer et API, at et token med aud værdien svarer til API'ens unikke identifikator. Derfor, medmindre man opretholder kontrol over både applikationen og API'et, vil afsendelse af et ID-token til et API normalt ikke fungere. Da ID-token'et ikke er underskrevet af API'et, vil API'et ikke kunne vide, hvorvidt applikationen har ændret token'et, f.eks. en tilføjelse af flere scopes, hvis det skulle acceptere ID-tokenet.

ID-tokens bruges i token-baseret godkendelse til at cache information om brugerprofiler og give dem til en klientapplikation. Programmet modtager et ID-token, efter at en bruger har godkendt godkendelsen. Derefter forbruges ID-tokenet og udtrækker brugeroplysninger fra det, som det derefter kan benytte til at tilpasse brugerens oplevelse.

Lad os for eksempel sige, at man har opbygget en almindelig webapplikation, registreret den med Auth0 og konfigureret den til at give en bruger mulighed for at logge ind ved hjælp af Google. Når en bruger logger ind på denne applikation, kan man bruge ID-token'et til at indsamle oplysninger, f.eks. brugernavn og e-mailadresse, som man herefter kan bruge til automatisk at generere og sende en personlig velkomst-e-mail.

ID-token sikkerhed

Som med alle andre JWT'er, bør man følge token best practice af Auth0, når ID-tokens bruges. Denne 'best practice' gennemgås i det følgende.

Token best practice

Først og fremmest, understreger 'best practice' elementerne, at man bør "holde det hemmeligt og sikkert". Underskrivelsesnøglen bør håndteres som enhver anden oplysning om legitimitet og kun afsløres for tjenester som behøver den. Herefter understreger 'best practice' at følsomme data ikke bør tilføres til nyttelasten. Ifølge Advodan[1] er 'følsomme oplysninger' bl.a. oplysninger vedrørende helbredstilstand, racemæssig eller etnisk baggrund og politiske, religiøse eller filosofiske overbevisninger. Dette element skyldes, at tokens er underskrevet for at beskytte mod manipulation og let afkodes, og det minimale antal krav til nyttelasten bør tilføjes for at optimere ydelse og den generelle sikkerhed.
Dernæst bør tokens tildeles en udløbsdato, idet et token – teknisk set, når det er underskrevet – er evigt gyldigt, medmindre signaturnøglen ændres, eller man eksplicit indstiller et specificeret udløb. Denne del af 'best practice' introducerer dog nogle mulige problematikker, hvorfor det er essentielt samtidig at formulere en strategi for valgt udløb eller tilbagekald af symboler. Tokens bør, desuden, kun sendes via HTTPS-forbindelser, idet ikke-HTTPS-forbindelser kan opfanges og dertilhørende tokens derfor risikerer at blive kompromitteret. Slutteligt, bør alle godkendelses use-cases overvejes. Tilføjelse af et sekundært verificeringssystem omkring tokens som sikrer at der genereres tokens fra serveren er ikke standard praksis, men det kan være nødvendigt for at imødekomme samtlige krav.

[1] https://www.advodan.dk/erhverv/persondata/hvad-er-foelsomme-oplysninger/

ID Token levetid

Generelt er et ID-token gyldigt i 36000 sekunder, svarende til 10 timer. Hvis der opstår sikkerhedsproblemer, kan man forkorte perioden før tokens udløber, og det er her vigtigt at huske, at et af formålene med tokens er at forbedre brugeroplevelsen ved at cache brugerinformation.

Nedenstående Billede 1 viser et eksempel på et ID-token.

```
{
  "iss": "http://YOUR_DOMAIN/",
  "sub": "auth0|123456",
  "aud": "YOUR_CLIENT_ID",
  "exp": 1311281970,
  "iat": 1311280970,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "gender": "female",
  "birthdate": "0000-10-31",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

Billede 1: Eksempel på ID-token

”Aud” skal indeholde client ID’et på den applikation som foretager autoriseringsforespørgslen. Hvis dette ikke er tilfældet for det pågældende token, bør det omtalte ID token ikke stoles på.

Access Tokens

Access Tokens bruges i token-baseret godkendelse for at give en applikation adgang til et APIs features. Applikationen modtager et access token efter en bruger har godkendt og autoriseret adgang, og derefter bruger access tokenet som en legitimationsoplysning, når den kalder det valgte API. Det vedlagte token informerer API'et om, at bæreren af tokenet er blevet autoriseret til at få adgang til API'et og kan udføre specifikke handlinger der er specificeret af det omfang, der blev tildelt under autorisationen.

Hvis man desuden har valgt at tillade brugere at logge ind via en Identity Provider (IdP), såsom Facebook, udsteder IdP'en sit eget access token for at give din applikation mulighed for at ringe til idP's API. For eksempel, hvis en bruger autentificerer ved hjælp af Facebook, kan access token udstedt af Facebook bruges til at kalde Facebook Graph API. Disse tokens styres af IdP og kan udstedes i ethvert format.

Opaque Access Tokens

Opaque Access Tokens er symboler i et proprietærformat, der typisk indeholder en identifikator til information i en servers vedvarende lager. For at validere et Opaque token, skal modtageren af token'et kalde den server, der har udstedt token'et. Opaque Access Tokens er symboler, hvis format man ikke har adgang til.

Bearer Token

Et Bearer Token er den mest almindelige form for Access Token. Den er en del af HTTP Authentication, hvor den mest basale er Basic, hvor kun bruger og kodeord gives med. Navnet *'Bearer'* henviser til, at man skal give "*Beareren*" adgang til sit endpoint.

Det er af typen Opaque, der ikke er beregnet til at have nogen betydning for klienter, der bruger den. Den skal gives i en Request header, skrevet som følgende:

Authorization: Bearer <token>

Nogle servere udsteder tokens der er en kort streng med hexadecimale tegn, mens andre muligvis bruger strukturerede tokens såsom JSON Web Tokens. I vores eksempel, længere nede i teksten, benyttes en string af tilfældige karakterer.

JSON-Web Token Access Token

JSON Web Token (JWT) Access Tokens overholder JSON Web Token standarden og indeholder oplysninger om en enhed i form af claims. De er uafhængige af at kalde en server for at validere et token. Vi bruger ikke JWT i vores test opstilling.

Access Token sikkerhed

Man skal følge tokens 'best practice', når man bruger Access Tokens, og for JWT'er, skal man sørge for at validere et Access Token, før det antages at dets indhold kan stoles på.

Tilpassede APIer

Som standard er et Access Token til et brugerdefineret API gyldig i 86400 sekunder, svarende til 24 timer. Vi vil anbefale, at gyldighedsperioden for dit token indstilles baseret på sikkerhedskravene i din API. For eksempel skal et access token, der får adgang til et bank-API, udløbe hurtigere end et der får adgang til et to-do-API. Angående udløbstid, så vises der brugseksempler i vores test opstilling. Her kan det ses hvordan man forlænger udløbstiden for access tokens.

Nedenfor på Billede 2 vises et eksempel på et Access Token.

```
{
  "iss": "https://YOUR_DOMAIN/",
  "sub": "auth0|123456",
  "aud": [
    "my-api-identifier",
    "https://YOUR_DOMAIN/userinfo"
  ],
  "azp": "YOUR_CLIENT_ID",
  "exp": 1489179954,
  "iat": 1489143954,
  "scope": "openid profile email address phone read:appointments"
}
```

Billede 2: Eksempel på et Access Token

Access Tokens har ingen information om brugeren udover brugerens ID (sub claim). Det indeholder derimod information om hvilke aktioner applikationen har lov til at udføre (scope claim), og derfor gør det Access Tokens brugbare til at sikre et API.

Refresh Tokens

Auth0 uddeler et Access Token eller et ID-token som respons på et autoriserings request. Som tidligere nævnt, bruges Access Tokens til at udforme autoriserede kald til et sikret API, og ID-tokens indeholder brugerinformation. Begge af disse tokens har en begrænset levetid (exp claim). Typisk skal en bruger bruge et ny Access Token når brugeren får adgang til resursen første gang, eller når det tidligere Access Token er udløbet.

Et Refresh Token er en speciel token som bruges til at få et fornyet Access Token. Det kan bruges indtil det er blacklistet. Refresh Tokens forbedrer brugeroplevelsen for native applikationer betydeligt, idet brugeren

kun skal autoriseres én gang gennem webautoriseringsprocessen. Efterfølgende re-autoriseringer kan finde sted uden brugerinteraktioner, ved at bruge et Refresh Token. Refresh Tokens skal opbevares sikkert, idet de kan tildele evig autorisering til brugeren.
Du kan øge sikkerheden ved at bruge Refresh Token-rotation, der udsteder et nyt Refresh Token og ugyldiggør forgængerens token med hver anmodning, på et nyt access token. Rotation af Refresh Token reducerer risikoen for et kompromitteret Refresh Token.

Refresh token rotation

Refresh Token-rotation er en teknik til at få nye Access Tokens ved hjælp af Refresh Tokens, der går ud over Silent Authentication, dette kan ske når brugeren ikke ser at vedkommende er godkendt, f.eks. ved at der ikke dukker et login vindue op. Refresh Tokens har typisk længere levetid og kan bruges til at anmode om nye Access Tokens, efter at de kortere-levende Access-tokens udløber. Refresh Tokens bruges ofte i native applikationer på mobile enheder sammen med kortvarige Access Tokens til at give problemfri UX uden at skulle udstede Access Tokens med lang levetid.

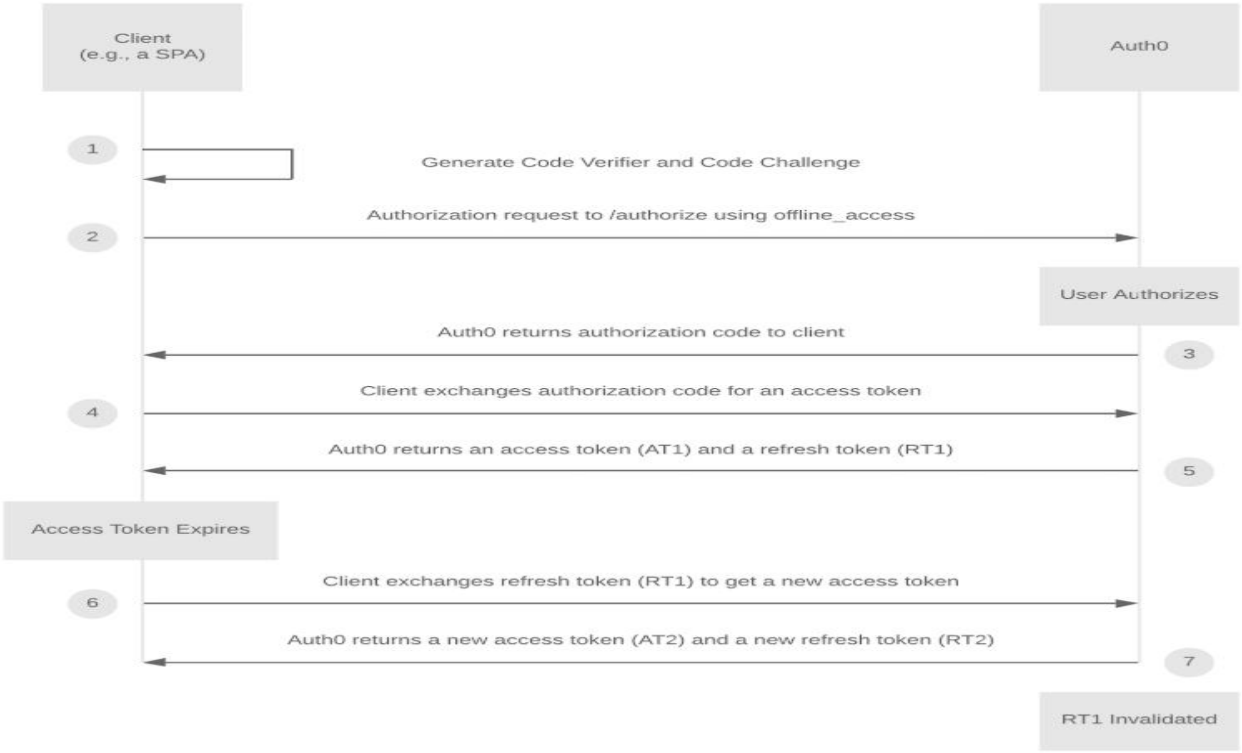
Ved refresh token rotation, returneres også et nyt refresh token, hver gang en klient udveksler et refresh token, for at få et nyt access token. Derfor har man ikke længere en langvarig Refresh Token, som, hvis den kompromitteres, kan give illegal adgang til ressourcer. Idet Refresh Tokens konstant udveksles og ugyldiggøres, reduceres truslen.

Vedligeholdelse af brugersessioner i SPAer

Desværre er langvarige Refresh Tokens ikke egnede til SPAer, fordi der ikke er nogen vedvarende opbevaringsmekanisme i en browser, der kun kan sikre adgang med den tilsigtede applikation. Da der er sårbarheder der kan udnyttes til at få disse artefakter af høj værdi og give ondsindede aktører adgang til beskyttede ressourcer, er brugen af Refresh Tokens i SPAs stærkt alarmerende.

Refresh Token-rotation tilbyder afhjælpning af slutbrugerens sessioner, der går tabt på grund af bivirkninger af browserens privatlivsmekanismer. Da Refresh Token-rotation ikke er afhængig af adgang til Auth0-session-cookien, påvirkes denne ikke af ITP eller lignende mekanismer.

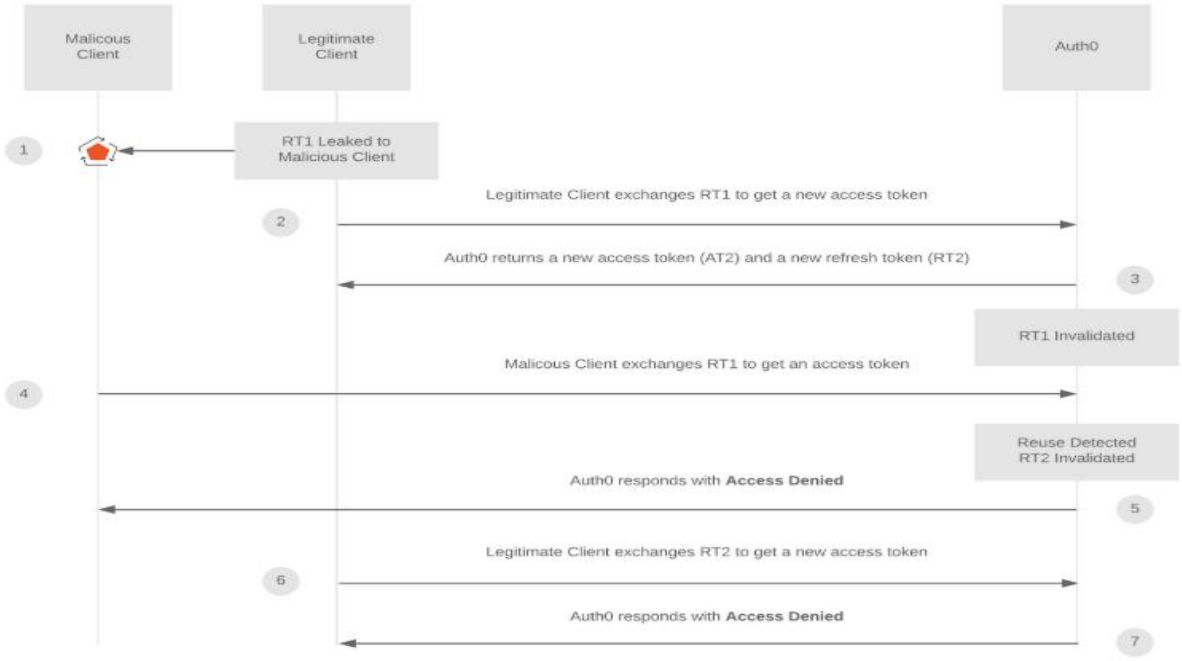
Som tidligere nævnt, bruges Refresh Tokens til at få et nyt Access Token efter det er udløbet, hvorfor et Refresh Token også har en længere levetid end et Access Token. For at minimere risikoen for at et Refresh Token kompromitteres, kan man gøre brug af Refresh Token-rotation. Dette betyder at hver gang en klient sender et Refresh Token, for at få et nyt Access Token, får klienten også udleveret et nyt Refresh Token, og på den måde slipper man for at have Refresh Tokens med lang levetid, som potentielt kan kompromitteres og give uberettiget adgang til resurser. Da Refresh Tokens kontinuerligt udskiftes og invalideres, reduceres risikoen for dette. Figur 1 nedenfor illustrerer hvordan Refresh Token-rotation kan se ud.



Figur 1: Eksempel på Refresh Token

Når en klient skal bruge et nyt Access Token, sender det et Refresh Token med et request til Aut0 om at få et nyt ”token-pair”. Så snart det nye Refresh Token er uddelt, invalideres det gamle. Dette sikrer applikationen imod ”replay-attacks”, som kunne være resultatet af et kompromitteret token.

Ved ikke at håndhæve ”sender-constraints” er det umuligt for autoriseringsserveren at vide om aktøren er legal eller ondsindet i forbindelse med et ”replay-attack”, og derfor er det vigtigt at invalidere nyligt uddelte Refresh Tokens når et tidligere brugt Refresh Token benyttes igen. Dette forhindrer hvilket som helst Refresh Token fra samme ”familie” i at blive brugt til at få et nyt Access Token. Man kan, eksempelvis, overveje scenariet illustreret i Figur 2 nedenfor. Elementerne i scenariet beskrives under figuren.



Figur 2: Refresh Token Scenarie

1: Legitim client har refresh token 1 (RT1), som bliver lækket eller stjålet af en ondsindet klient.

2: Legitim klient bruger RT1 til at få et nyt access token (AT2).

3: Auth0 returnere refresh token 2 (RT2) og access token 2(AT2).

4: Den ondsindede klient prøver nu at få et access token ved hjælp af RT1. Auth0 anerkender at RT1 bliver genbrugt og invaliderer dermed hele RT-familien, RT2 inkluderet.

5: Auth0 returnere ”Access Denied” til den ondsindede klient.

6: Access token 2 udløber og den legitime klient prøver at bruge RT2 til at få et nyt access token.
Auth0 returnere ”Access Denied” til den legitime klient.

7: Re-autorisering er påkrævet.

Denne beskyttelsesmekanisme fungerer uanset om den legitime klient eller den ondsindede klient er i stand til at udveksle Refresh Token 1 for et nyt token par før det andet. Så snart genbrug registreres nægtes alle efterfølgende anmodninger, indtil brugeren genautentiseres. Når genanvendelse registreres, indfanger Auth0 registrerede genanvendelsesbegivenheder (såsom *ferrt*, der indikerer en mislykket udveksling) i logfiler.

Et andet eksempel er, hvor den ondsindede klient stjæler Refresh Token 1 og med succes bruger det til at erhverve et Access Token, før den legitime klient forsøger at bruge Refresh Token 1. I dette tilfælde ville den ondsindede klients adgang være kortvarig, fordi Refresh Token 2 (eller eventuelle efterfølgende udstedte RTer) tilbagekaldes automatisk, når den legitime klient forsøger at bruge Refresh Token 1, som vist i forrige diagram.

Mobil/devices

Mange tror fejlagtigt, at en godkendelsesproces baseret på et brugernavn kombineret med en adgangskode er rigeligt sikker. Men en sådan proces identificerer kun 'hvem' der får adgang til API-serveren, men ikke 'hvad' der får adgang til den. Forskellene mellem disse elementer af 'hvem' og 'hvad' der opnår adgang til en API-server, illustreres i Billede 4 nedenfor.

Billede 3: Forskellen mellem 'hvem og 'hvad'

Den tilsigtede kommunikationskanal repræsenterer den mobile applikation der bruges som forventet, af en legitim bruger uden ondsindede intentioner, og ved hjælp af en uforfalsket version af mobil-applikationen og kommunikation direkte med API-serveren uden at blive udsat for et såkaldt "middleman attack".

Den egentlige kanal kan repræsentere flere forskellige scenarier. Eksempelvis kan en legitim bruger med ondsindede intentioner muligvis bruge en ompakket version af mobil applikationen, mens en hacker der benytter den ægte version af mobil applikationen kan udøve et "middleman attack", for bedre at forstå hvordan kommunikationen mellem mobil applikationen og API-serveren foregår for dermed at blive i stand til at automatisere angreb mod API. Mange andre scenarier er mulige, men vi vil ikke beskrive hver enkelt i denne rapport.

Brugeren af den mobile app kan autentificeres, autoriseres og identificeres på flere måder, såsom at bruge OpenID Connect eller OAUTH2-strømme.

Mens brugergodkendelse muligvis lader API-serveren vide, hvem der bruger API'et, kan den ikke garantere, at anmodningerne stammer fra forventelig kilde, altså den originale version af mobilappen.
Nu er der brug for en måde at identificere hvad der ringer til API-serveren, og her bliver opgaven vanskeliggjort yderligere, idet der skal identificeres, hvad der anmoder API-serveren. Det skal vurderes, om

det er en ægte udgave af mobilapplikationen, en 'bot', et automatiseret script eller en hacker som manuelt bearbejder API'en ved hjælp et værktøj som f.eks. Postman. Det kan dog vise sig at være en af de legitime brugere der benytter en ompakket version af mobilapplikationen eller et automatiseret script der forsøger at drage fordel af den service som mobilapplikationen leverer til brugeren. Normalt hardcoder udviklere en API-nøgle direkte ind i koden på mobilapplikationen, og nogle udviklere tager sikkerheden et skridt videre ved at inkludere API-nøglen i selve kørsel af mobilapplikationen, og gør det derved til en run-time-secret i modsætning til de tidligere praksisser omkring API-nøgler som implementerede en static secret i koden.
Udfordringen ved at gemme data på f.eks. en mobil, er, at selvom dataene er krypteret kan 'reverse engineere' foregå via programmer som Frida eller Xposed. Disse gennemgås kort i følgende afsnit ifølge programmernes egne beskrivelser.

Frida

Injicér dine egne scripts i black box-processer. Tilslut enhver funktion, spion på crypto APIer eller spor privat applikationskode, ingen kildekode er nødvendig. Rediger, tryk på gem, og se øjeblikkeligt resultaterne. Alt uden kompileringstrin eller program genstarter.

xPosed

Xposed er et framework for moduler, der kan ændre opførelsen af systemet og apps uden at røre nogen APKer. Det er fantastisk, fordi det betyder, at moduler kan arbejde til forskellige versioner og endda ROM uden ændringer.

Man in middle // Middleman

I en enhed kontrolleret af angriberen, kan man bruge en proxy til at udføre et "middleman attack" for at udtrække enhver information som kan bruges til at identificere 'hvem' eller 'hvad'. En bedre løsning kan bruges ved at bruge en Mobile App Attestation-løsning, der gør det muligt for API-serveren at sikre, at den kun modtager anmodninger fra ægte mobilapplikationer.

Attest for mobilapp

Brug af en Mobile App Attestation-løsning vil gøre det muligt for API-serveren at vide, hvad sender forespørgsler, og således tillades kun at svare på forespørgsler fra en ægte mobilapplikation, mens alle andre anmodninger fra usikre kilder vil blive afvist.
Formålet med at benytte en mobilapp attest-løsning er at garantere run-time, at din mobilapplikation ikke bliver manipuleret og ikke styres af programmer såsom xPosed eller Frida, ikke bliver MitM angrebet. Dette kan opnås ved at køre en SDK i baggrunden, hvilket er en tjeneste, som kører i 'skyen', der udfordrer

applikationen og baseret på modtagne svar attesteres integriteten af mobilapplikationen og enheden der kører, SDK vil aldrig være ansvarlig for beslutninger. Ved vellykket attestering af mobilapp-integriteten udstedes og signeres en kortvarig JWT-token med en hemmelighed, som kun API-serveren og Mobile App Attestation-tjenesten i skyen er opmærksom på. I tilfælde af fejl i mobilapp-attesten er JWT-tokenet underskrevet med en hemmelighed, som API-serveren ikke kender.

Herefter skal appen sendes med hvert API kald JWT-tokenet i overskrifterne til anmodningen. Dette tillader, at API-serveren kun godkender anmodninger når det er muligt for den at verificere signatur og udløbstid i JWT-tokenet, og nægter dem når en sådan verificering ikke kan gennemføres. Når den secret, der bruges af Mobile App Attestation-tjenesten, ikke er kendt af mobilapplikationen, er det ikke muligt at vende tilbage til det ved kørsel, selv når appen er manipuleret, kører i en rodet enhed eller kommunikerer via en forbindelse, der er mål for et ”middleman attack”.

AUTH-flow

De fleste indbyggede applikationer logger ind én gang og kun én gang. Ideen bag dette er, at Refresh Token aldrig udløber, og at det altid kan udveksles med en gyldig JWT. Problemet med et symbol, der aldrig udløber, er, at det aldrig udløber. Hvad kan man gøre, hvis man mister en enhed? Ved en mistet mobiltelefon skal det identificeres af brugeren på en eller anden måde, og applikationen skal give en måde at tilbagekalde adgangen. Vi besluttede at bruge enhedens navn, f.eks. "maryos iPad". Derefter kan brugeren gå til applikationen og tilbagekalde adgang til "maryos iPad".

En anden tilgang er at tilbagekalde Refresh Token’et ved specifikke begivenheder. En interessant begivenhed er at ændre adgangskoden (Se afsnittet ”Skift af kodeord” senere i denne rapport). Vi mener, at JWT ikke er nyttigt til disse use cases, men det er bedre at bruge en tilfældig genereret streng, da det alligevel er vores server der holder styr på den. Dette er gjort i vores praktiske eksempel - dog i form af en webapplikation, da vi vurderede, det var det mest simple eksempel.

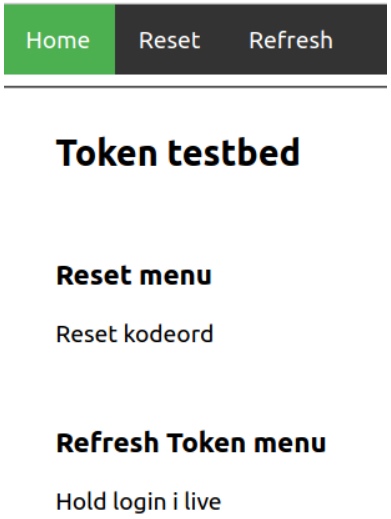
Web

Et godt mønster er at opdatere tokenet, før det udløber. Indstil token-udløbet til en uge og Refresh Token, hver gang brugeren åbner webapplikationen og hver time. Hvis en bruger ikke åbner applikationen i mere end en uge, skal de logge ind igen, og dette er acceptabelt webapplikations UX. For at forlænge en session indsendes det Refresh Token man fik udstedt sammen med Access Token'et. Herved får man et nyt, og hvis man forsøger at bruge det gamle, får man en fejl.

Brugseksempler

AUTH-flow - Refresh Token

Vi har sat et testbed op for at undersøge Refresh Token og reset password flowene, for at vise hvordan det kan foregå. Testbed kan ses her sec.sandersolutions.dk. For at se Refresh Token flowet, trykkes på Refresh menuen.



Funktionalitet:

Login

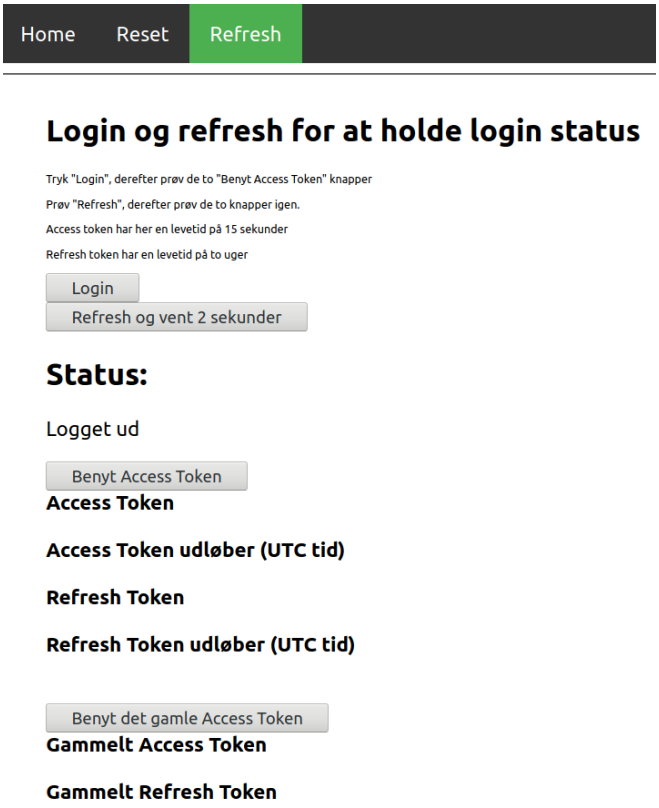
Her logges ind. Bruger og kodeord er hardcoded og kan ses af wireshark optagelser længere nede.

Refresh og vent

Her opnås et nyt Access Token, når det gamle er løbet ud eller man vil opretholde en tilstand at være "logget ind". Her er man "logget ind" så længe man bruger et brugbart Access Token.

Benyt Access Token

Anvender det aktive Access Token i forsøget på at få den hemmelige data. Er man succesfuld kommer der en tekst med den hemmelige data.



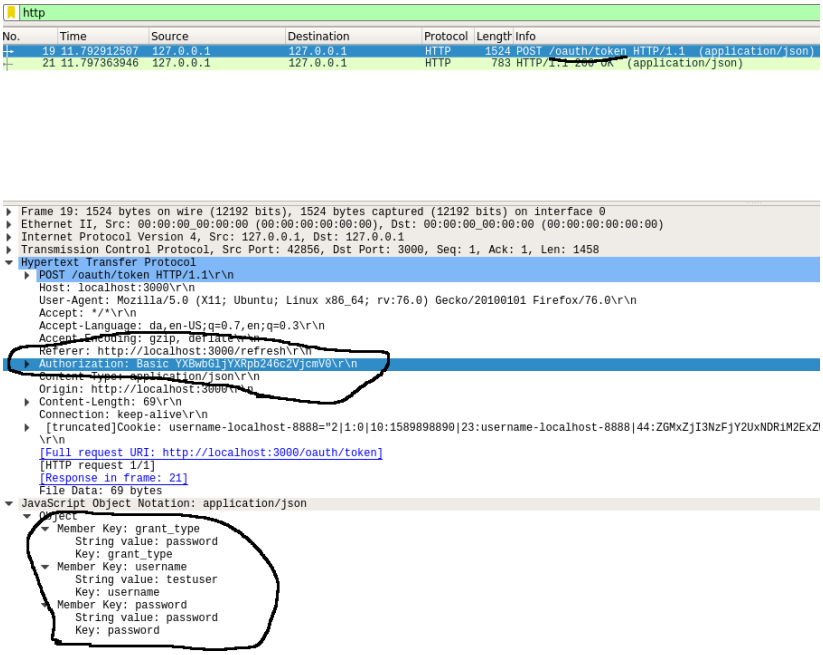
Benyt det gamle Access Token

Anvender et gammel udskiftet Access Token for at se hvad der sker.

Der mangler en knap hvor det forsøges at benytte det gamle Refresh Token. Hvis tiden tillader det, kommer det på inden eksamen. Ifølge dokumentationen for RFC-6749 burde der komme samme fejl som ved en udløbet access token, "invalid token". Link på dokumentationen ovenover: https://tools.ietf.org/html/rfc6749.html#section-4.2.2.1 Det tunge arbejde er udført af node.js middleware oauth2-server (se Appendix for links). Testbeden har: 1) et endpoint (*/oauth/token*), der håndterer tokens. Alt efter hvad man sender med i et request body, styrer hvad der kommer tilbage, 2) et endpoint (*/oauth/*), der kun viser data hvis det rigtige Access Token er sendt med i request header.

Login

Trykkes på Login knappen ansøges om at få et token fra serveren.
Ved at benytte simpel HTTP Authentication får man sendt et Access Token, en Refresh Token, og andre detaljer tilbage, som bla udløbstider. Man sætter grant_type til password og Oauth serveren sender tokens tilbage. Alle detaljer kan ses i wireshark billedet nedenunder.



Request:

POST

Header:

Authorization: Basic YXBwbGljYXRpb246c2VjcmV0 *(Base64 encoded "application:secret")*

Content-Type: application/x-www-form-urlencoded

Body:

grant_type=password

username=testuser

password=password

No.	Time	Source	Destination	Protocol	Length	Info
19	11.792912507	127.0.0.1	127.0.0.1	HTTP	1524	POST /oauth/token HTTP/
21	11.797363946	127.0.0.1	127.0.0.1	HTTP	783	HTTP/1.1 200 OK (appli

```
▸ Frame 21: 783 bytes on wire (6264 bits), 783 bytes captured (6264 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 3000, Dst Port: 42856, Seq: 1, Ack: 1459, Len: 717
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 200 OK\r\n
    Access-Control-Allow-Origin: *\r\n
    X-DNS-Prefetch-Control: off\r\n
    X-Frame-Options: SAMEORIGIN\r\n
    Strict-Transport-Security: max-age=15552000; includeSubDomains\r\n
    X-Download-Options: noopen\r\n
    X-Content-Type-Options: nosniff\r\n
    X-XSS-Protection: 1; mode=block\r\n
    Content-Type: application/json; charset=utf-8\r\n
  ▸ Content-Length: 278\r\n
    ETag: W/"116-eV8TgsaiqiyYp2irc66DaSbANRA"\r\n
    Date: Mon, 25 May 2020 15:06:21 GMT\r\n
    Connection: keep-alive\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.004451439 seconds]
    [Request in frame: 19]
    [Request URI: http://localhost:3000/oauth/token]
    File Data: 278 bytes
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: accessToken
        String value: 53787f01d7ebbb871c828d1758d9b8bc7548c45a
        Key: accessToken
    ▾ Member Key: accessTokenExpiresAt
        String value: 2020-05-25T15:06:36.575Z
        Key: accessTokenExpiresAt
    ▾ Member Key: refreshToken
        String value: a1f40ea7ad74d314f49bfe948dc1c414399017dd
        Key: refreshToken
    ▾ Member Key: refreshTokenExpiresAt
        String value: 2020-06-08T15:06:21.575Z
        Key: refreshTokenExpiresAt
    ▾ Member Key: client
      ▾ Object
        ▾ Member Key: id
            String value: application
            Key: id
        Key: client
    ▾ Member Key: user
```

Response (obs originale tokens i blå. Tokens er ikke JWT men bare en tilfældig række karakterer):

{

"accessToken":"53787f01d7ebbb871c828d1758d9b8bc7548c45a",

"accessTokenExpiresAt":"2020-05-25T15:06:36.575Z",

"refreshToken":"a1f40ea7ad74d314f49bfe948dc1c414399017dd",

"refreshTokenExpiresAt":"2020-06-08T15:06:21.575Z",

"client":

{

"Id":"application"

},

"user":{

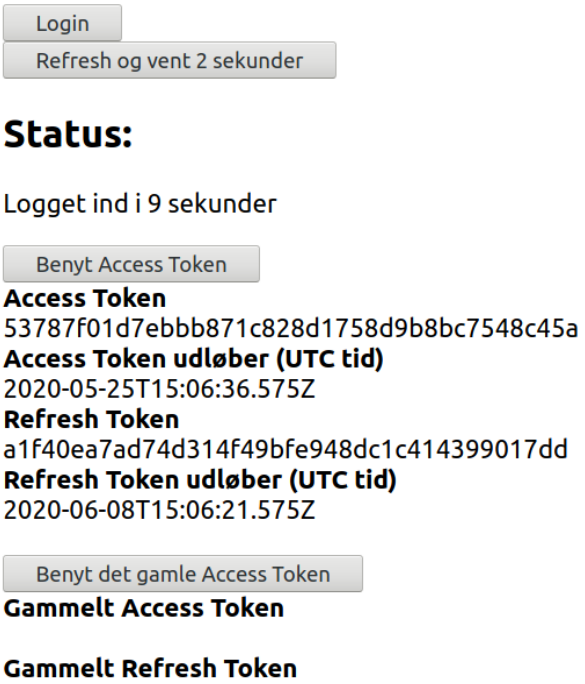
"Username":"testuser"

}

}

Resultatet ses her:

Siden viser at man har et brugbart Access Token (med 9 sekunder tilbage), og den viser data omkring de to tokens.



Client Credentials

Hvis man vil have en klient til at få oplysninger, som ikke beror på en bruger, så sætter man i request body, *grant_type = client_credentials*. Er ikke med i optagelser eller testbed.

Refresh Token

Når en klient har brug for at få et nyt Access Token, kan man trykke på refresh knappen. Derefter sendes nedenstående request til serveren. Udeladt er headers cache-control: no-store og Pragma: no-cache. Disse kan bruges til at fortælle serveren at den ikke skal gemme tokenet:

Request

POST

Header:

Authorization: Basic YXBwbGljYXRpb246c2VjcmV0 *(Base64 encoded "application:secret")*

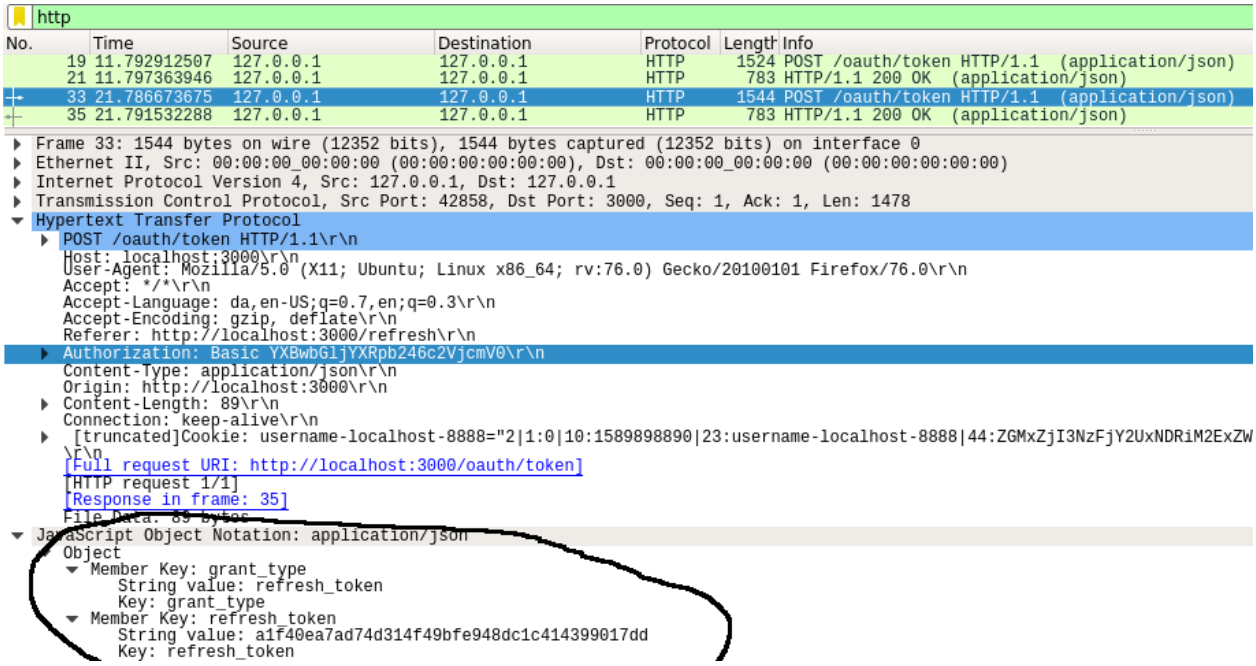
Content-Type: application/x-www-form-urlencoded

Body

(obs det originale Refresh Token i blåt):

grant_type=refresh_token

refresh_token=a1f40ea7ad74d314f49bfe948dc1c414399017dd



Response (obs nye tokens i grønt):

{

"accessToken":"b2b84bbcb96250e0743127d576b4a9b5c04bcee6",

"accessTokenExpiresAt":"2020-05-25T15:06:46.569Z",

"refreshToken":"8acbbeaa95ac8096a27bfe99f8eb4362802a5134",

"refreshTokenExpiresAt":"2020-06-08T15:06:31.569Z",

"client":

{

"Id":"application"

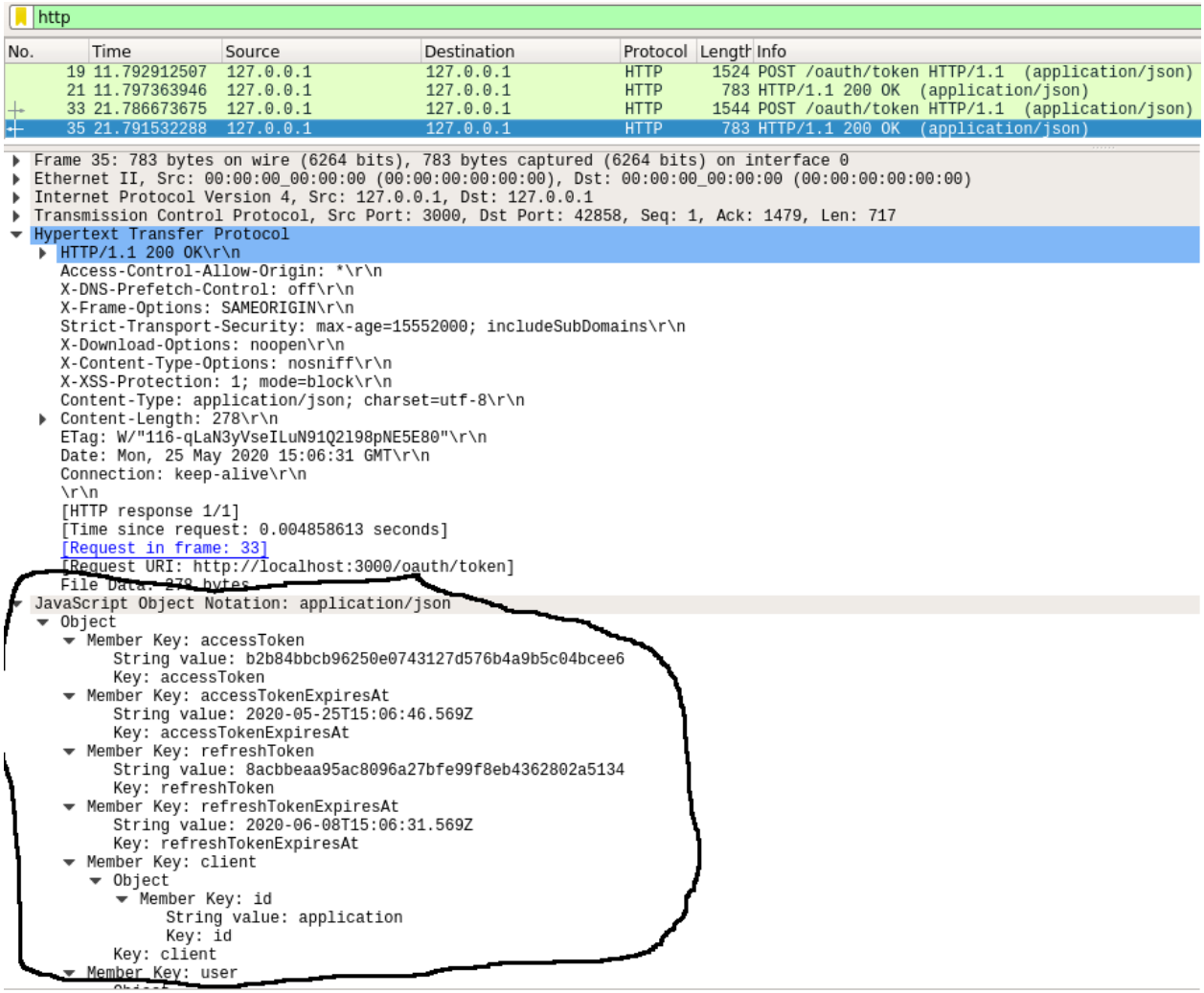
},

"user":{

"Username":"testuser"

}

}



Nu er der kommet et nyt Access Token og en ny Refresh Token. Klienten gemmer dog de gamle for at vise at de ikke dur mere.

Resultatet er dette: Vi har vundet nogle flere sekunder, og selvom udløbstiden er 15 sekunder, går nogle af sekunderne på at klienten venter med at vise resultatet - og på at "fotografen" tager et snapshot.

Status:

Logget ind i 9 sekunder

Benyt Access Token

Access Token
b2b84bbcb96250e0743127d576b4a9b5c04bcee6
Access Token udløber (UTC tid)
2020-05-25T15:06:46.569Z
Refresh Token
8acbbeaa95ac8096a27bfe99f8eb4362802a5134
Refresh Token udløber (UTC tid)
2020-06-08T15:06:31.569Z

Benyt det gamle Access Token

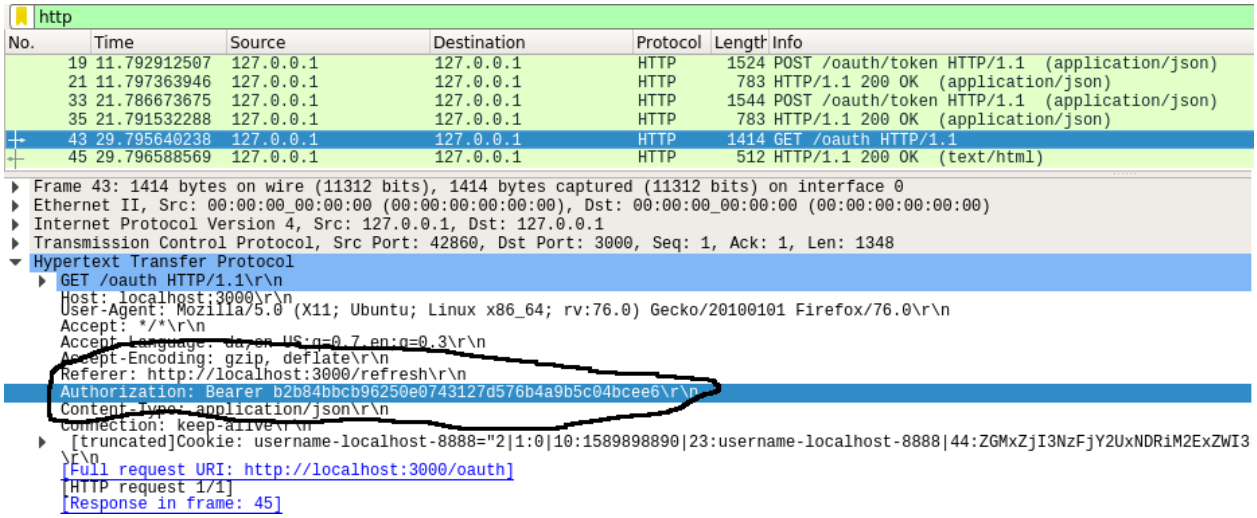
Gammelt Access Token
53787f01d7ebbb871c828d1758d9b8bc7548c45a
Gammelt Refresh Token
a1f40ea7ad74d314f49bfe948dc1c414399017dd

Succes:

Herefter kan man bruge sit Access Token i headeren ved hver brug af endpointet. Observer at her bruges HTTP Authentication **Bearer** token (læs om den længere oppe i teksten):

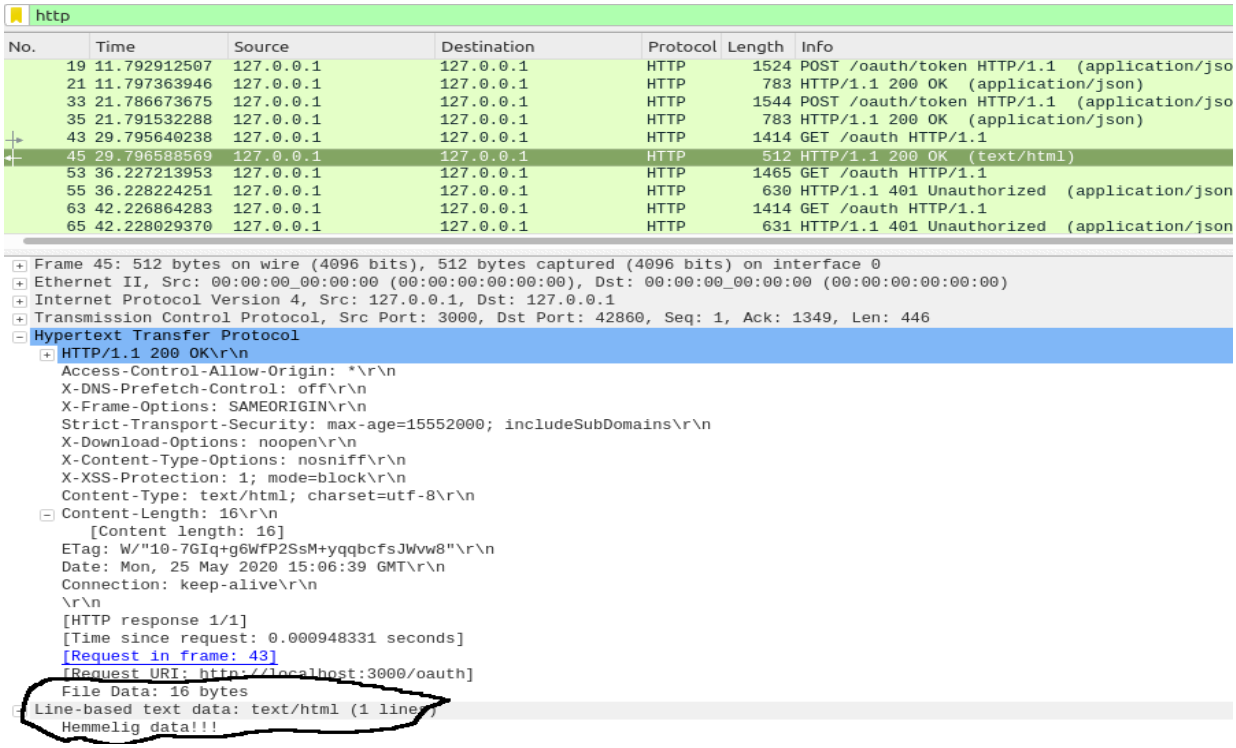
Request: GET

Header: Authorization: Bearer b2b84bbcb96250e0743127d576b4a9b5c04bcee6

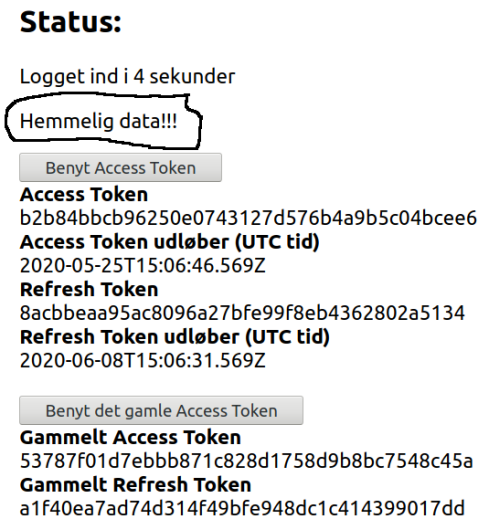


Response:

Den hemmelige data



Klienten viser den hemmelige data på skærmen:



Invalid:

Hvis det brugte Access Token, er for gammelt, er blevet revoked eller der er andet galt kommer der en fejl:

Request: GET

Header (obs originale, nu forældede Access Token): Authorization: Bearer

b2b84bbcb96250e0743127d576b4a9b5c04bcee6

Response:

{

"statusCode":401,

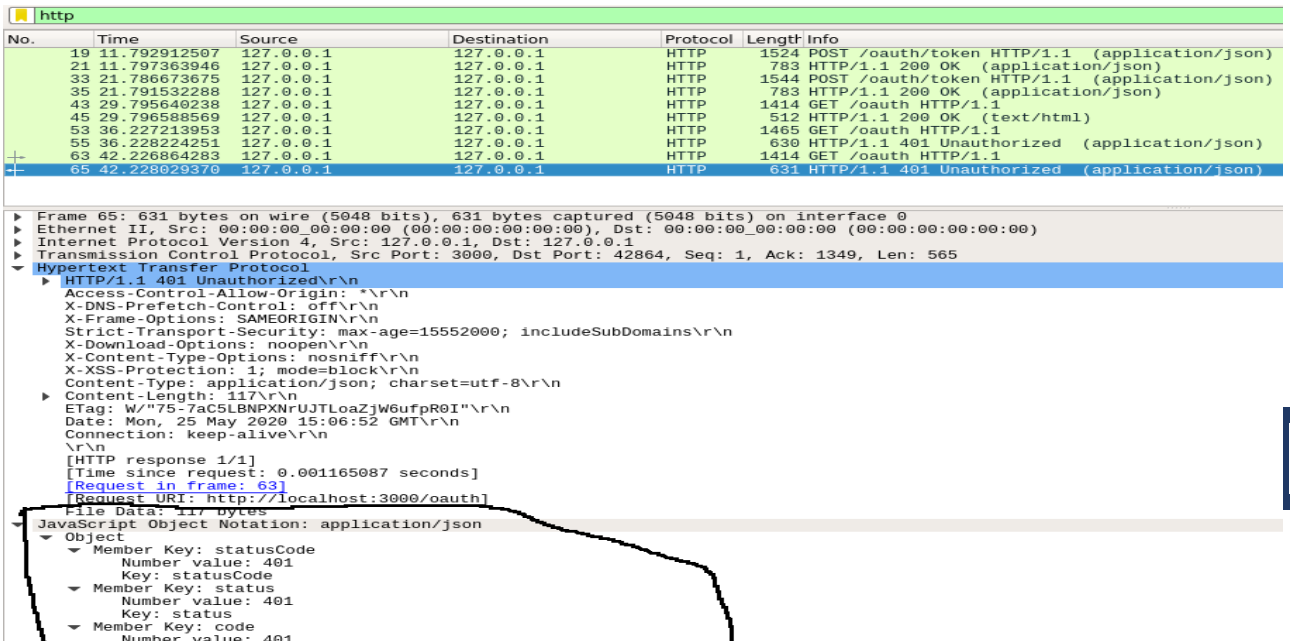
"status":401,

"code":401,

"message":"Invalid token: access token is has expired",

"name":"invalid_token"

}

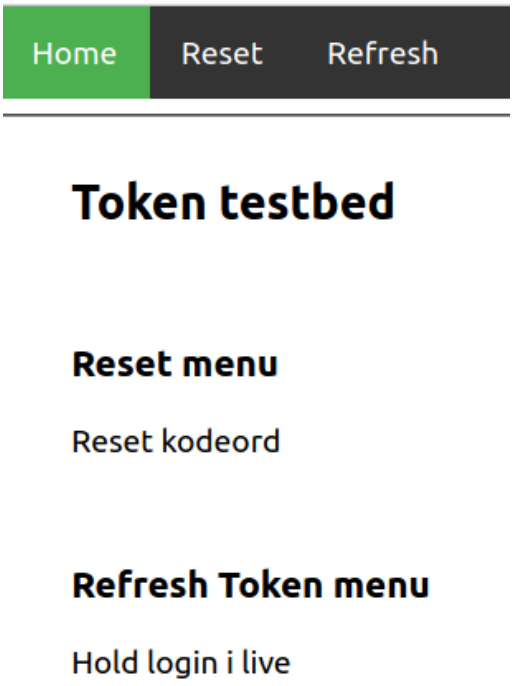


Så når man får et nyt Access Token ses det tydeligt at OAUTH2-serveren har fjernet det gamle tokens rettighed til de hemmelige data. Hvis man forsøger med alt andet en det korrekte Access Token, kommer følgende respons *Access token is invalid*:

```
▼ JavaScript Object Notation: application/json
  ▼ Object
    ▼ Member Key: statusCode
        Number value: 401
        Key: statusCode
    ▼ Member Key: status
        Number value: 401
        Key: status
    ▼ Member Key: code
        Number value: 401
        Key: code
    ▼ Member Key: message
        String value: Invalid token: access token is invalid
        Key: message
    ▼ Member Key: name
        String value: invalid_token
        Key: name
```

AUTH-flow - Skift af kodeord

I testbeden på sec.sandersolutions.dk er det muligt at afprøve den manuelle metode til at skifte kodeord på.

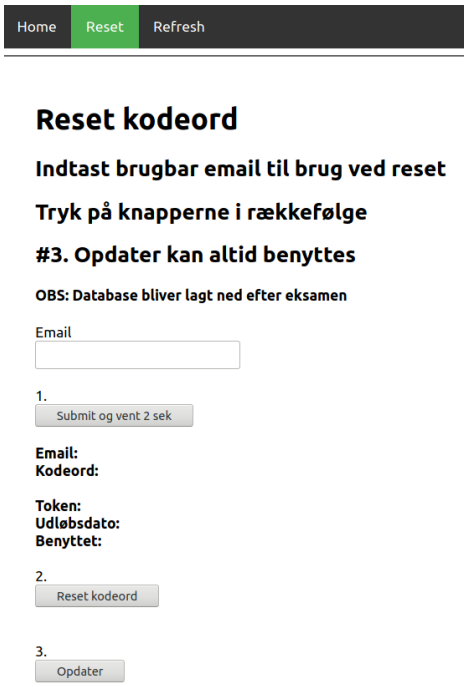


Vælg menupunktet **Reset**

For at teste skal man indtaste en gyldig e-mailadresse, som man kan tilgå undervejs. Denne e-mail bliver gemt i en mysql database. Man kunne også lige så godt have brugt en mongoDB eller andet. Det er blevet argumenteret at de indbyggede muligheder for at automatisk at slette gamle token-indlæg i en mongoDB gør den velegnet til formålet, men mysql er brugt, da vi havde en let mulighed for at sætte den op og vise auth flowet på. Data skal i denne database slettes af koden selv eller et automatiseret job i styresystemet.

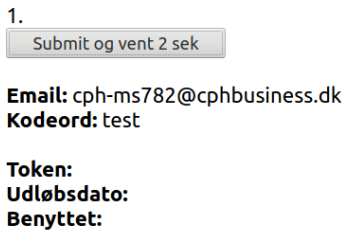
På nedenstående billede er der et input felt til den gyldige e-mailadresse, og tre knapper man kan styre reset flowet med.

1. **Submit og vent**. sender e-mailadressen op til databasen, og hvis den er der i forvejen, får man vist kodeord (og som bonus også tidligere tokens fra et tidligere reset). Det er naturligvis helt hul i hovedet sikkerhedsmæssigt, og er kun for testing.
2. **Reset kodeord.** Sender en e-mail til den aktuelle adresse med et link til at resette med. Når man klikker på linket, kommer man tilbage til domænet men ikke samme side, og flowet her er lidt manuelt, hvor man selv skal skifte mellem sider.
3. **Opdater**. Er til når man kommer tilbage efter at have resat sit kodeord.



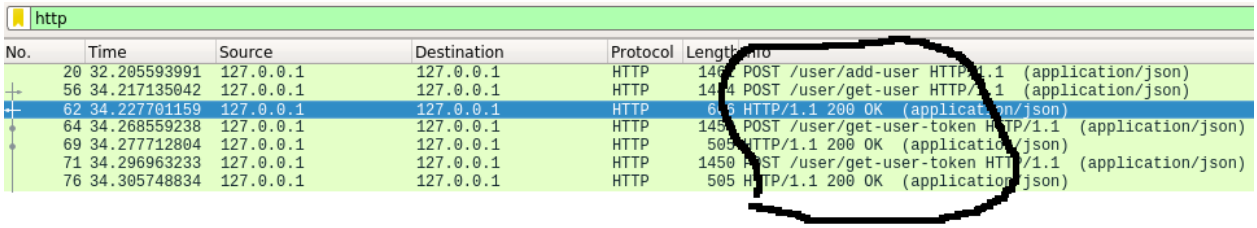
Lad os se hvad der sker i baggrunden.

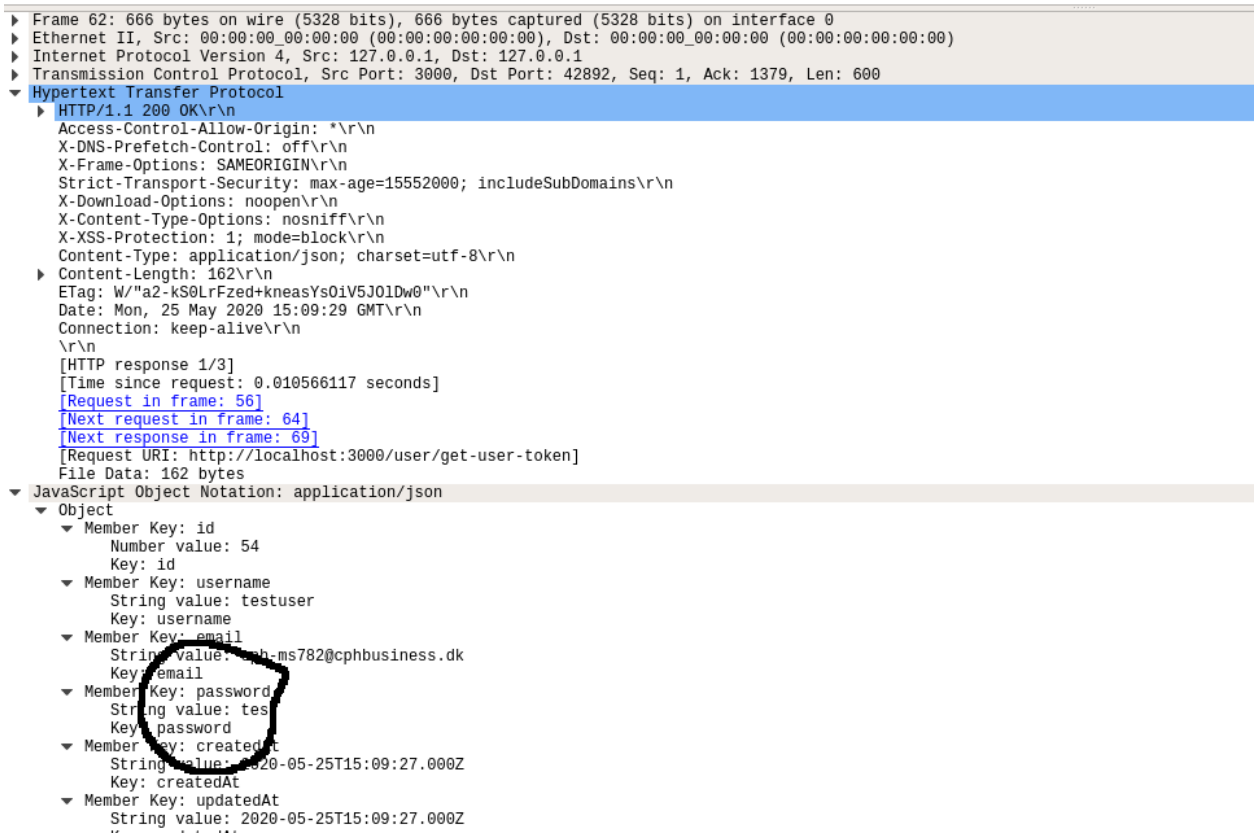
Når man sender sin e-mailadresse, kommunikerer siden med serveren. Var det en produktions side, ville man ikke få noget svar på om e-mailadressen eksisterer. Her ville man bare få et ok, lige meget hvad man skriver. For at teste skal det selvfølgelig være en gyldig adresse.



Som det ses på billedet til højre, er kodeordet dukket op (”test”), men ikke noget token, så den har ikke været brugt før eller alt data er blevet slettet i databasen.

Der er 6-7 http pakker fanget i wireshark (se nedenunder). Wireshark optagelserne er tilgængelige her (se Appendix hvis link ikke virker). Man kan se på endpoints hvad der søges gjort - add-user (gemmer evt. e-mail), get-user (henter bruger - kunne måske slås sammen i en), get-user-token (henter token for bruger hvis den er der). Nedenstående pakke er den mest interessante, da den sender kodeordet med.





På nedenstående billede er reset mailen sendt, og et token er blevet oprettet og gemt i databasen. Derudover er en udløbsdato opretter og en celle der holder øje med om den har været brugt før.

Det token, der bliver oprettet, er sikker at sende med en e-mail (URLsafe), da den er Opaque og har ingen værdi/information i sig selv. Derudover fungerer det som en ekstra sikkerhed at der bliver holdt øje med om et token er brugt før. Det benyttes længere nede i flowet.

Email: cph-ms782@cphbusiness.dk
Kodeord: test

Token: ggZn/BYowa6EwS2iCkfvmVqQCBnBPXIENN5e8M7NTp+VXKwxuULaAk2Mt9RsSuPPKr5caHWLFC19IMHc/uIeWQ==
Udløbsdato: 2020-05-25T15:11:05.000Z
Benyttet: 0

2.
Reset kodeord

Email Sendt. Check email adresse.

OBS bemærk token info ovenover

Tiden er UTC

Mailen, der bliver sendt, indeholder det URL sikre token og e-mailadressen. (Den er her sendt via localhost for at wireshark kan optage pakkerne).

For at nulstille dit kodeord, tryk på linket her nedendunder.

http://localhost:3000/user/reset-password?token=ggZn%2FBYowa6EwS2iCkfvmVqQCBnBPXIENN5e8M7NTp%2BVXKwxuULaAk2Mt9RsSuPPKr5caHWLFC19IMHc%2FuIeWQ%3D%3D&email=cph-ms782@cphbusiness.dk

Trykker man på linket kommer følgende side. Her kan man skrive sit nye kodeord.

Der er selvfølgelig alt for mange oplysninger på siden, og var det en produktionsmaskine, ville man ikke kunne se kodeordene blive indtastet.

På billedet til højre er det nye kodeord "12345" indtastet.

Reset password testbed

Indtast dit nye kodeord (synligt pga testing).

Nyt kodeord:

12345

Pga test er der ingen mindre grænse for hvor langt kodeordet er.

Bekræft nyt kodeord:

12345

Kodeord skal være ens.

Reset password og gå derefter tilbage til reset side

Resultatet af Reset password knapper er følgende:

Kodeord er nulstillet. Benyt det nye kodeord fremover. Gå tilbage til reset side og tryk på Opdater

Forsøger man igen (eller hvis en hacker har opsnappet e-mailen, og brugt linket før), kommer dette:

Token er udløbet. Prøv kodeords nulstilling igen.

Meddelelsen er den samme, uanset om token'et udløber eller er blevet brugt før. Udløbstiden er som skrevet længere oppe ikke en del af token'et, da den ikke indeholder data, men er styret af den tid der er skrevet i databasen.

Når man kommer tilbage til Reset-siden, kan man trykke på Opdater for at se resultatet som er følgende:

Email: cph-ms782@cphbusiness.dk
Kodeord: 12345

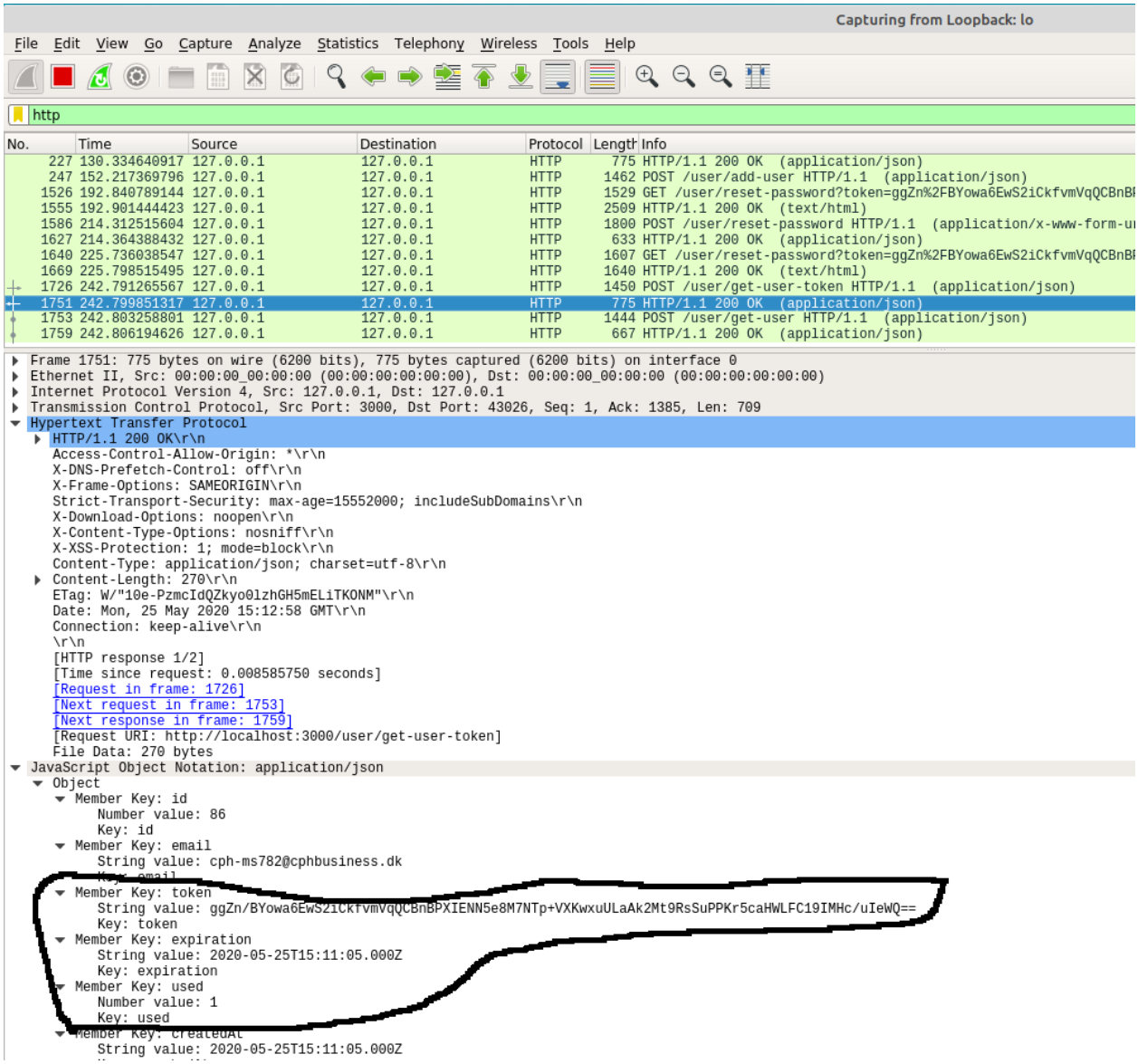
Token: ggZn/BYowa6EwS2iCkfvmVqQCBnBPXIENN5e8M7NTp+VXKwxuULaAk2Mt9RsSuPPKr5caHWLFC19IMHc/uIeWQ==
Udløbsdato: 2020-05-25T15:11:05.000Z
Benyttet: 1

2.
Reset kodeord

3.
Opdater

Som det ses på ovenstående, er det pågældende token nu benyttet en gang og kodeordet er udskiftet til **12345**.

Wireshark sladrer om detaljerne. Her er svaret på get-user-token. Used er nu beskrevet som 1, så selvom Expiration ikke er opnået endnu, så vil koden ikke accepterer flere forespørgsler på dette token.


Capturing from Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
No. Time Source Destination Protocol Length Info
227 130.334640917 127.0.0.1 127.0.0.1 HTTP 775 HTTP/1.1 200 OK (application/json)
247 152.217369796 127.0.0.1 127.0.0.1 HTTP 1462 POST /user/add-user HTTP/1.1 (application/json)
1526 192.840789144 127.0.0.1 127.0.0.1 HTTP 1529 GET /user/reset-password?token=ggZn%2FBYowa6EwS2iCkfvmVqQCBnBP
1555 192.901444423 127.0.0.1 127.0.0.1 HTTP 2509 HTTP/1.1 200 OK (text/html)
1586 214.312515604 127.0.0.1 127.0.0.1 HTTP 1800 POST /user/reset-password HTTP/1.1 (application/x-www-form-u
1627 214.364388432 127.0.0.1 127.0.0.1 HTTP 633 HTTP/1.1 200 OK (application/json)
1640 225.736038547 127.0.0.1 127.0.0.1 HTTP 1607 GET /user/reset-password?token=ggZn%2FBYowa6EwS2iCkfvmVqQCBnBP
1669 225.798515495 127.0.0.1 127.0.0.1 HTTP 1640 HTTP/1.1 200 OK (text/html)
1726 242.791265567 127.0.0.1 127.0.0.1 HTTP 1450 POST /user/get-user-token HTTP/1.1 (application/json)
1751 242.799851317 127.0.0.1 127.0.0.1 HTTP 775 HTTP/1.1 200 OK (application/json)
1753 242.803258801 127.0.0.1 127.0.0.1 HTTP 1444 POST /user/get-user HTTP/1.1 (application/json)
1759 242.806194626 127.0.0.1 127.0.0.1 HTTP 667 HTTP/1.1 200 OK (application/json)
Frame 1751: 775 bytes on wire (6200 bits), 775 bytes captured (6200 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 3000, Dst Port: 43026, Seq: 1, Ack: 1385, Len: 709
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Access-Control-Allow-Origin: *\r\n
X-DNS-Prefetch-Control: off\r\n
X-Frame-Options: SAMEORIGIN\r\n
Strict-Transport-Security: max-age=15552000; includeSubDomains\r\n
X-Download-Options: noopen\r\n
X-Content-Type-Options: nosniff\r\n
X-XSS-Protection: 1; mode=block\r\n
Content-Type: application/json; charset=utf-8\r\n
Content-Length: 270\r\n
ETag: W/"10e-PzmcIdQZkyo0lzhGH5mELiTKONM"\r\n
Date: Mon, 25 May 2020 15:12:58 GMT\r\n
Connection: keep-alive\r\n
\r\n
[HTTP response 1/2]
[Time since request: 0.008585750 seconds]
[Request in frame: 1726]
[Next request in frame: 1753]
[Next response in frame: 1759]
[Request URI: http://localhost:3000/user/get-user-token]
File Data: 270 bytes
JavaScript Object Notation: application/json
Object
Member Key: id
Number value: 86
Key: id
Member Key: email
String value: cph-ms782@cphbusiness.dk
Member Key: token
String value: ggZn/BYowa6EwS2iCkfvmVqQCBnBPXIENN5e8M7NTp+VXKwxuULaAk2Mt9RsSuPPKr5caHWLFC19IMHc/uIeWQ==
Key: token
Member Key: expiration
String value: 2020-05-25T15:11:05.000Z
Key: expiration
Member Key: used
Number value: 1
Key: used
Member Key: createdAt
String value: 2020-05-25T15:11:05.000Z

Konklusion

Vi kan konkludere at tokens, i sig selv, ikke er sikre nok da de kan opsnappes og bruges af ondsindede individer. Dog kan man reducere risici ved hjælp af Auth0's token rotation, idet man i høj grad forbedrer sikkerheden, da man sikrer at tokens ikke kan genbruges, og dermed sørger for at en klient - ondsindet eller ej - der genbruger en token skal gen-autentificere sig selv.

Det er heller ikke nødvendigvis særligt sikkert at gemme sine tokens på en mobiltelefon eller anden device, da disse ifølge nævnte værktøjer kan hives frem, selvom de er krypterede. Der er ikke noget der hedder '100% sikkerhed', og det er kun et spørgsmål om hvor meget man vil gøre, og hvor meget en bruger vil finde sig i, før vedkommende finder det umuligt at bruge en funktionalitet. Skal man f.eks. logge ind hele tiden, kan virke overskueligt for en ung bruger der har en anden og bedre forståelse for teknologi, idet det har været "tæt på kroppen" hele livet. Men for ældre personer kan det at "være logget ind" være ekstremt svært at forholde sig til. Skal man i et sådan tilfælde give ældre et uendeligt access token, som de altid kan være logget ind med? Skal man give dem et refresh token med samme udløbsdato, eller mangel på uendeligt access token? Set fra et IT-sikkerhedsmæssigt perspektiv, giver sådanne tiltag ingen mening.

Derfor er der nogle punkter hvor det kan siges, at man slækker lidt på sikkerheden for at styrke funktionaliteten. Her oplever vi, at refresh token rotation er er meget godt eksempel, idet dette token udskiftes ved brug, og derfor mindskes risikoen for misbrug. Access tokens har allerede så kort en udløbstid (ideelt), at sikkerhedsproblemerne er til at overse.

I gamle dage var der to muligheder for dit kodeord. Enten gemmes din adgangskode i almindelig tekst, eller også er som krypteret tekst, med mulighed for at dekryptere den. Det er i stedet for at have den meget stærkere og mere sikre envejskryptering. Fordi (sikre) adgangskoder ikke burde kunne dekrypteres, giver det os et af to valg, når en bruger glemmer deres adgangskode: 1) Lav en ny, midlertidig adgangskode, og send den via e-mail, eller 2) Generer en e-mail, der indeholder et engangslink i indholdet af e-mailen, der fører brugeren til en side, hvor de kan indtaste en ny sikker adgangskode.

Begge indstillinger sender en e-mail, som på længere sigt ikke bør betragtes som et sikkert lagringsmedium. Med den første mulighed vises adgangskoden i almindelig tekst. Hvis brugeren skulle gemme denne e-mail i deres indbakke som deres metode til at huske deres adgangskode (især fordi de ikke valgte den), ville det være næsten lige så usikkert som at skrive deres adgangskode på en note og efterlade den ved siden af deres computer.

Et andet problem med den første mulighed er, at en ondsindet bruger, der kender deres e-mailadresse, let vil kunne låse en bruger ud fra webstedet ved at nulstille deres adgangskode. Hvis den ondsindede bruger gentog dette igen og igen, ville det gøre det næsten umuligt for brugeren nogensinde at logge ind igen, fordi deres adgangskode aldrig ville forblive den samme.

Derfor anbefaler vi at man vælger mulighed nr. 2 (som vi har gjort i reset password koden). Token er sendt med et URL sikkert opaque token, og det tjekkes via serveren.

Et forholdsvist sikkert setup kunne være at man altid bad om at skifte kodeord, får tilsendt et nyt kodeord, benytter det, og i samme proces får et access token/refresh token der kun er gyldigt i kort tid. Dette kræver imidlertid at man har adgang til sin e-mail, med alt det som det kan medbringe af udfordringer. Men det er indhold til en anden fremtidig rapport.

Appendix

Arbejdsfordeling

Vi har i løbet af dette projekt forsøgt at arbejde så dynamisk som muligt, på denne led har alle fået en indsigt og indflydelse på samtlige dele af rapporten.

Vi har arbejdet på denne led igennem hele projektperioden med få undtagelser, da Martin og Sercan har haft Full-stack JavaScript har de haft en væsentligt bedre forståelse for netop JavaScript.

Martin har næsten på egen hånd udviklet hele test koden, med en smule hjælp fra Sercan. Martin og Sercan har på bedst mulig måde sat Christian og Dennis ind i JavaScript delen og har på denne led øget den samlede forståelse for hele projektet for alle medlemmer.

Dennis og Christian har primært fundet information om emnet og renskrevet samt formateret rapporten.

Slutteligt har vi alle lyttet til kommunikationen på testserveren igennem Wireshark.

Vi føler at vi ved at uddelegere arbejdet på denne led har udnyttet gruppens kompetencer bedst muligt og at det har rustet os markant i forhold til vores kommende praktikperiode, ved at give os den fornødne erfaring til at indgå i et projekt.

Kilder

React klienten på sec.sandersolutions.dk er hjemmelavet. Al kode for både server og klient (der ligger inde i serveren i webclient folderen) kan findes her:
https://github.com/cph-ms782/securityProject

Live testbed serveren er sat sammen med hjælp fra disse links

Refresh Token del:
https://github.com/pedroetb/node-oauth2-server-example

Reset password del:
https://www.smashingmagazine.com/2020/03/creating-secure-password-flows-nodejs-mysql/

Wireshark optagelser

Brug http som filter
Reset password
Refresh Token

Virker linket af en eller anden grund ikke, så skriv til en fra gruppen, og vi sender dem med mail eller giver en nyt link.

Node.js middleware:

oath2-server: https://www.npmjs.com/package/oauth2-server