Komposition – Authentication

Forstå, vælg og brug forskellige login strategier

**Agenda:**

Gennemgang af login strategier:

1. JAVA
2. Form-based
3. Token-based

**Java’s declarative authentication and authorization features**

I en ”*declarative security model*”, udtrykkes applikationens sikkerhed gennem regler og tilladelser, specifikt for J2EE applikationer. Reglerne og tilladelserne bliver defineret i en ”*deployment descriptor document*” (web.xml fil) og gemmes i en mappe WEB-INF. I web.xml angives applikationens sikrede komponenter, som ”security-constraints”, og en authentication-mekanisme (som f.eks. BASIC eller FORM). I en anden mappe, kaldet META-INF findes der en fil, context.xml, hvori der defineres de roller som skal indgå i authentication af en bruger. Den indeholder også forbindelsen til databasen.

Når det er muligt, anbefales det at bruge eksisterende authentication framework, i stedet for at definere sin egen. Det er Java der håndterer denne authentication og ikke os, som programmør udvikler.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Basic HTTP-authentication**

Ved basic HTTP, kan en klient undgå login prompt, ved at skrive brugernavn:adgangskode i url’en med en sikker HTTPS-forbindelse. F.eks. *https://Aladdin:OpenSesame@www.example.com/index.html*

1. En klient sender en forespørgsel, for at få adgang en sikker ressource.
2. Webserveren sender en retur dialog-boks med en forespørgsel af brugernavn og adgangskode.
3. Klienten sender sit brugernavn og adgangskode tilbage til serveren.
4. Et billede, der indeholder skærmbillede

   Automatisk genereret beskrivelseServeren godkender brugeren, hvis det indtastede er korrekt og sender den anmodede ressource tilbage til klienten.

**Eksempel på Basic Authentication**

Ved Basic Authentication angives der ikke en authentication-mekanisme i web.xml, da BASIC er sat som default. Det angives at data er ”CONFIDENTIAL”, hvilket vil sige at den data der sendes, ikke kan ses af andre brugere/bot under transmissionen af dataen.

I stedet for denne, kan man også angive ”INTEGRAL”, hvilket vil sige at data der sendes mellem klient og server, ikke kan ændre under transit.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelseTil sidst, kan man angive ”NONE”, hvilket vil sige at der accepteres alle request i enhver forbindelse, også http.

Derudover er der angivet en JDBCRealm, som forbinder til databasen.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Fordele:**

1. Den er simpel, fordi den ikke påkræver cookies, session eller loginsider.
2. Det er muligt at ”clear cache”, da man ikke kan logge ud direkte.
3. Man skal ikke logge ind hver gang man forsøger at tilgå en sikker ressource, da ens loginoplysninger bliver gemt i et state hos klienten. Dette bliver sendt med i request-header.

**Ulemper:**

1. Man får ikke direkte mulighed for at logge ud.
2. Brugernavn og adgangskode er eksponeret, da det bliver sendt med ved hver forespørgsel. Dette er uanset om forbindelsen er sikker eller ej.
3. Man skal acceptere browserens ”grimme” loginvindue.
4. Loginoplysninger er ikke krypteret, eller hashet, men base64 kodet.
5. Der kan ikke inkluderes andre informationer i sin forespørgsel (udover brugernavn og adgangskode).

**Form-based authentication**

1. Klienten sender en forespørgsel for at få adgang til en sikker ressource.
2. Hvis klienten ikke kan godkendes, serveren omdirigere klienten tilbage til en loginside.
3. Klienten indtaster og sender sit brugernavn og adgangskode tilbage til serveren.
4. Serveren forsøger at godkende klienten.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Eksempel på FORM-based Authentication**

Nedenstående tilføjes til web.xml.

Et billede, der indeholder kniv

Automatisk genereret beskrivelse

Lav en login.html, med en form (samt en error.html):

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Tilføj en Realm med en LockOut, som er en log-ud-mekanisme. Den aktiveres når man har for mange mislykkede forsøg, og betyder at man logges ude af ressourcen i en rum tid. Dette er med til at undgå DoS angreb.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Fordele:**

1. Den er simpel og brugervenlighed (visning af loginside).
2. Man kan selv designe ens loginvindue.

**Ulemper:**

1. Ens loginoplysninger bliver ikke gemt i state, og man skal derfor logge ind for hver eneste sikre ressource man forsøger at tilgå.
2. Hver gang man sender en forespørgsel, omdirigeres klienten til en loginside og sender herfra sit brugernavn og adgangskode til serveren. Dette eksponerer ens brugernavn og adgangskode.
3. Man skal tilgå databasen hver gang serveren skal godkende en klients loginoplysninger. Dette er tungt for serveren og i tilfælde hvor siden er meget brugt, kan man risikere at databasen er en ”flaskehals”.

**Token Based Authentication**

Using JSON Web Token (JWT)

1. Klienten sender en POST forespørgsel til en loginside.
2. Serveren validerer oplysninger, og hvis de er korrekte, generes et JSON Web Token, som sendes retur til klienten i et response (200 OK), og gemmes lokalt, typisk i local storage, eller session storage, men cookies kan også bruges. Dette er i modsætning til den traditionelle tilgang, hvor session oprettes på serveren og en cookie returneres.
3. Et billede, der indeholder skærmbillede

   Automatisk genereret beskrivelseKlienten sender en GET-forespørgsel til den beskyttede ressource, så vil serveren lede efter et valid web-token i headeren, og hvis det er til stede, får klienten adgang til den beskyttede ressource.

**Fordele:**

1. Tokens er ”stateless”, hvilket vil sige at de selv indeholder den information der skal bruges til godkendelse. På denne måde, frigøres serveren og session state skal ikke gemmes derpå. Dette betyder også at databasen ikke skal tilgås flere gange.
2. Oplysningerne er krypteret og derfor er færre også eksponeret.
3. Payload indeholder alle nødvendige informationer om brugeren, for at undgå kald til databasen mere end højest nødvendigt.
4. Signeret – en måde hvorpå browseren kan verificere serveren og serveren kan verificere klienten.
5. Det er skalerbart, hvilket vil sige at der ikke er nogen ”flaskehals” i databasen, som vi f.eks. ville se i de ovenstående.
6. Kan deles blandt mange services.
7. Kan bruges på tværs af teknologier.

**Ulemper:**

1. Så snart et Token er givet, er det gyldigt indtil udløbsdato. Det er dog muligt at ”vække” Token igen.
2. Man kan logge ud ved at slette Token hos klient, men Token er stadig gyldig.
3. ”Secret” skal opdateres jævnligt. Hvis det signerede ”secret” på serveren bliver eksponeret, så er alle klienter der har tilgået serveren, i fare.
4. Det er nemt at dekryptere.

**Lav et Token:**

Tilføj ”claim” til Token, så det indeholder brugernavn, rolle samt en issuer.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Se data gemt i et udstedt Token:

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse