Login strategies

Webtjenester kunne let fås adgang til af enhver bruger, hvis de kendte URL og input-parameter for at få et output. For at forhindre, at uautoriserede personer får adgang til API'erne, er vi nødt til at autentificere brugerne ved hjælp af et kontrolpunkt.

Nogle af metoderne er:

- [Java's declarative authentication and authorization features] (././JavasDeclarativeAuthenticationAndAuthorization features.md)
- Basic HTTP-authentication
- Form-based authentication
- Token Based Authentication

Basic HTTP-authentication

Grundlæggende godkendelse er standard, når du ikke specificerer en godkendelsesmekanisme.

Basic authentication ligner standardmetoden med brugernavn / adgangskode, hvor vi bruger et brugernavn og en adgangskode til at godkende API'et. Her en Base64-kodet streng, der indeholder brugernavnet og adgangskoden sendt til klienten. Base-64-kodning er ikke en krypteringsmetode, bare en måde at tage binære data på og omdanne dem til tekst, så de let kan overføres.

Base64UrlEncode: Dette er en kodning. Det er konvertering fra en datatype til en anden. Afsendelse af ASCII-byte kan være et problem på mange stier. Så kodér dataene i base64. Der er 64 tegn. URL-kodning betyder, at det er sikkert at sende som url. F.eks .: - bruges i stedet for + og _ bruges i stedet for /

Specificering af basic HTTP-authentication kræver, at serveren anmoder om et brugernavn og adgangskode fra webklienten og kontrollerer, at brugernavnet og adgangskoden er gyldig ved at sammenligne dem med en database med autoriserede brugere i den specificerede eller standardreal.

Usecase

HTTPS Basic passer godt til et relativt uvigtig internt API, som kræver blot en simpel autentificering, men manglen på krypteringsdel gør det til et dårligt valg for API'er med følsomme data, da der er stor chance for eksponering i tilfælde af MITM (Man in Mellemangrebet).

Fordele

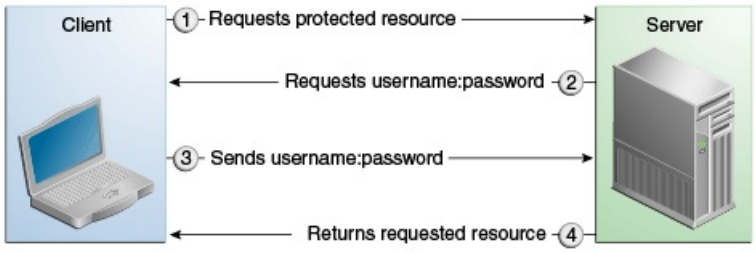
- Implementering er temmelig enkel, da der ikke er nogen kryptering involveret
- Brug relativt mindre tid på at svare, da det kun har et enkelt opkald
- Manglen på token-oprettelse og krypteringsmetode giver klienten en fordel ved at bruge mindre kode til at kalde API
- Oplysningerne hentes fra serveren med kun et opkald, hvilket gør dem hurtigere end andre komplekse godkendelser

Ulemper

- SSL tager tid at køre grundlæggende HTTP, så dette vil gøre svartiden betydeligt langsom
- Manglen på kryptering gør sikkerhedsrisikoen temmelig høj. Sikkerhedsproblemer skyldes overvejende brugen af Basic-autorisation uden SSL, i hvilket tilfælde brugernavnet og adgangskoden udsættes for en MITM.
- I en browser er der også problemer med udløb af legitimationsoplysninger, men dette er ikke så meget af et problem for REST-tjenester.

Når der bruges grundlæggende godkendelse, foregår følgende handlinger.

- En klient anmoder om adgang til en beskyttet ressource.
- Webserveren returnerer en dialogboks, der anmoder om brugernavn og adgangskode.
- Klienten sender brugernavnet og adgangskoden til serveren.
- Serveren autentificerer brugeren i den specificerede verden og returnerer den ønskede ressource, hvis den er vellykket.



Form-based authentication

Hoved funktionen ved FBA er dens evne til at autentificere sig imod forskellige brugere og roller.

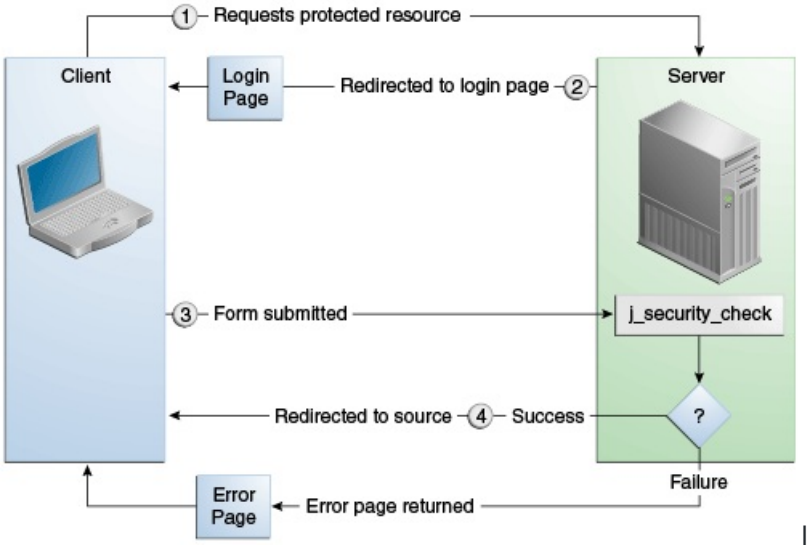
Tillader udvikleren at kontrollere udseendet og følelsen af login-godkendelsesskærmbillederne ved at tilpasse login-skærmen og fejlsiderne, som en HTTP-browser præsenterer for slutbrugeren.

Når form-baseret godkendelse er erklæret, forekommer følgende handlinger.

Usecase

Fordele

Ulemper



Java's declarative authentication and authorization features

Container-managed security (J2EE declarative security), og Java Authentication and Authorization Services (JAAS) er sikkerhedsteknologierne til godkendelse og godkendelse i Java 2 Enterprise Edition (J2EE) version 1.4 og nyere.

Som standard tillader Oracle JAAS-sikkerhedsudbyderen, at brugerkontooplysninger og sikkerhedsroller kan gemmes på en af to placeringer: et filbaseret XML-format eller i et LDAP-bibliotek, der udnytter Oracle Internet Directory (OID).

Man definere roller og brugere igennem web.xml filen F.eks. giver man brugere der er medlemmer af USERS rollen afgang på følgende måde:

```
<security-constraint>
```

```
<web-resource-collection>
    <web-resource-name>AllPublic</web-resource-name>
    <url-pattern>/</url-pattern>
</web-resource-collection>
<auth-constraint>
    <role-name>USERS</role-name>
</auth-constraint>
```

Dette beskytter en webapplikation, der starter fra dens J2EE-rodkatalog som angivet gennem "/" mønsteret. For kun at anvende beskyttelse på undermapper, skulle mønsteret ændres til "/ undermappe-navn". Rollenavnet USERS er et logisk navn, der skal kortlægges til en eksisterende rolle.

Usecase

- Når man alligevel bruger java

Fordele

- Du kan nemt ændre den måde, brugeren autentificerer udelukkende ved at konfigurere din server - uden at skulle ændre noget i din kode.
- Fordelen ved J2EE-erklæringssikkerheden er, at den kræver ringe eller ingen programmering af applikationsudvikleren, fordi de fleste af sikkerhedsbeslutningerne træffes under installationen. Denne model gør det også muligt at anvende ændringer til sikkerhedsdefinitionen uden at kræve genkodning i applikationen.
- Sikkerhed er unikt tilpasset applikationens behov.
- Sikkerhed er finkornet med applikationsspecifikke indstillinger.

Ulemper

- Kan være kompliceret at sætte sig ind i
- Applikationen er afhængig af sikkerhedsattributter, der ikke kan overføres mellem applikationstyper.
- Support til flere protokoller gør denne type sikkerhed sårbar.
- Data er tæt på eller indeholdt i sårbarhedspunktet.

Token-based Authentication

Når vi taler om token-baseret autentificering, henviser vi ofte til JWT (JSON Web Token), fordi det er blevet brugt meget i alle brancher og er blevet en de-facto standard til godkendelse.
JWT er en åben standard, der definerer en kompakt, sikker og selvstændig måde at transmittere data mellem parter i JSON.
JWT er en statsløs type godkendelse. Det betyder, at serveren ikke gemmer nogen sessioninformation i databasen. Det behøver ikke at føre en fortegnelse over, hvilken bruger der er logget ind, eller hvilken token der er udstedt for hvilken bruger. I stedet sender klienten efterfølgende anmodninger til serveren med en header i form af bærer- {JWT-token}, eller oftere sender klienten den i kroppen af en POST-anmodning eller som en URL-parameter.

JWT'er består af tre dele adskilt med prikker (.), Som er:

```
**Header** - Headeren består typisk af to dele: typen af token, som er JWT, og den signeringsalgoritme, d
For eksempel:
```

> {
> "alg": "HS256",
> "typ": "JWT"
> }

Derefter er denne JSON Base64Url kodet til at danne den første del af JWT. Har forklaret Base64Url-kodning nedenfor.

```
**Payload** - Dette indeholder kravene. De data, vi ønsker. Et eksempel på payload kan være:
```

> {
> “sub”: “1234567890”,
> “name”: “John Doe”,
> “admin”: sandt
> }

Nyttelasten kodes derefter Base64Url for at danne den anden del af JSON Web-token. Dette kan læses af alle som det lige er kodet. Det er ikke krypteret. Kodning er bare konvertering fra den ene form til den anden. Så undgå at bruge følsomme data som Aadhaar-nummer her.

```
**Signatur** - For at oprette signaturdelen skal du tage den kodede header, den kodede nyttelast, en hemm

Hvis du f.eks. Vil bruge HMAC SHA256-algoritmen, oprettes signaturen på følgende måde:
```

> HMACSHA256 (
> base64UrlEncode (header) + “.” +
> base64UrlEncode (nyttelast),
> hemmelighed)

Signaturen bruges til at bekræfte, at meddelelsen ikke blev ændret undervejs, og i tilfælde af tegn, der er underskrevet med en privat nøgle, kan den også bekræfte, at afsenderen af JWT er den, den siger, den er.

Når vi modtager tokenet, er det første, vi gør, at verificere signaturen. Vi afkoder overskriften og nyttelasten og foretager derefter denne signatur med vores hemmelighed som med ovenstående algoritme. Hvis de matcher, er det en underskrevet JWT.

Derfor ser en JWT typisk ud som følgende. xxxxx.yyyyy.zzzzz

Fordele

- De er state-less. Man behøver ikke at gemme i en session ID.
- JSON Web Tokens er letvægt og kan let bruges på tværs af platforme og sprog. De er en smart måde at autentificere og autorisere uden sessioner.
- Skalerer let (f.eks. i forhold til cookie baseret auth)

Ulemper

- JWT’er kan være store (payload) i forhold til en sessions ID, Men de involverer ikke nogen handling med lagring af værdier på serveren.
- Når en token er udleveret, hvordan trækker man dem tilbage. Token levetid. Hvis der ikke er levetid på token, så vil den virke altid. Så skal man holde øje med hvilke tokens der er givet ud og så er det ikke stateless mere
- Hvis serverens “secret” signing string bliver kompromiteret, så er alle tokens kompromiseret.
- Kan være sårbar for XSS og CSRF angreb
  XSS - Cross-site scripting gør det muligt for en angriber at udføre vilkårlig JavaScript i et offerbrugers browser.
  CSRF Cross-site request forgery giver en angriber mulighed for at tilskynde en offerbruger til at udføre handlinger, som de ikke har til hensigt.

Token-based Authentication