Hitelesítés és engedélyezés

Webprogramozás – 12. előadás

Sulyok Csaba
csaba.sulyok@gmail.com

A hitelesítés problémája

- A HTTP egy *állapot nélküli* protokoll, **de**:
  - szeretnénk kapcsolatot teremteni ugyanazon felhasználótól érkező kérések között
  - szeretnénk korlátozni az erőforrásokhoz való hozzáférést a kliens identitása alapján
- Nem eléggé robusztus használni a kliens IP címét, MAC-jét vagy custom query paramétereket
- Lehetőségek:
  - Bejelentkezés a HTTP beépített hitelesítéssel (`Authorization` header)
    - `basic` felhasználónév-jelszó (nem biztonságos, csak lokális hálózatban)
    - **token** használatával (amit vagy a helyi, vagy egy külső szerver állít elő)
  - Sütik (cookie) segítségével
    - a süti tartalmazhat session ID-t
    - a süti tartalmazhat egy tokent, mint fennebb
  - Ezeknek kombinációja

Biztonsági felvetések

- *authentication* (hitelesítés)
  - ki a bejelentkezett felhasználó
  - mivel a HTTP állapot nélküli protokoll, össze kell kössük a hívásainkat `Cookie` vagy `Authorization` fejlécek segítségével
  - HTTP *401 Unauthorized* = a felhasználó nincs *hitelesítve*
  - egy hitelesítési szerver előállít egy darab információt (session ID, access token), amellyel későbbi hívások engedélyeztetve lehetnek
  - használhatnak pl. felhasználónév/jelszó párost, külső szolgáltatást, stb.
- *authorization* (engedélyezés)
  - mit szabad a bejelentkezett felhasználónak csinálni (abban a kontextusban, hogy ismerjük a felhasználót)?
  - HTTP *403 Forbidden* = a felhasználó nincs *engedélyezve* (tudjuk hogy ki ő, de nem szabad a megadott erőforrást elérnie)
  - használja a korábban beállított session ID-t vagy tokent, hogy ellenőrízze a felhasználó engedélyeit

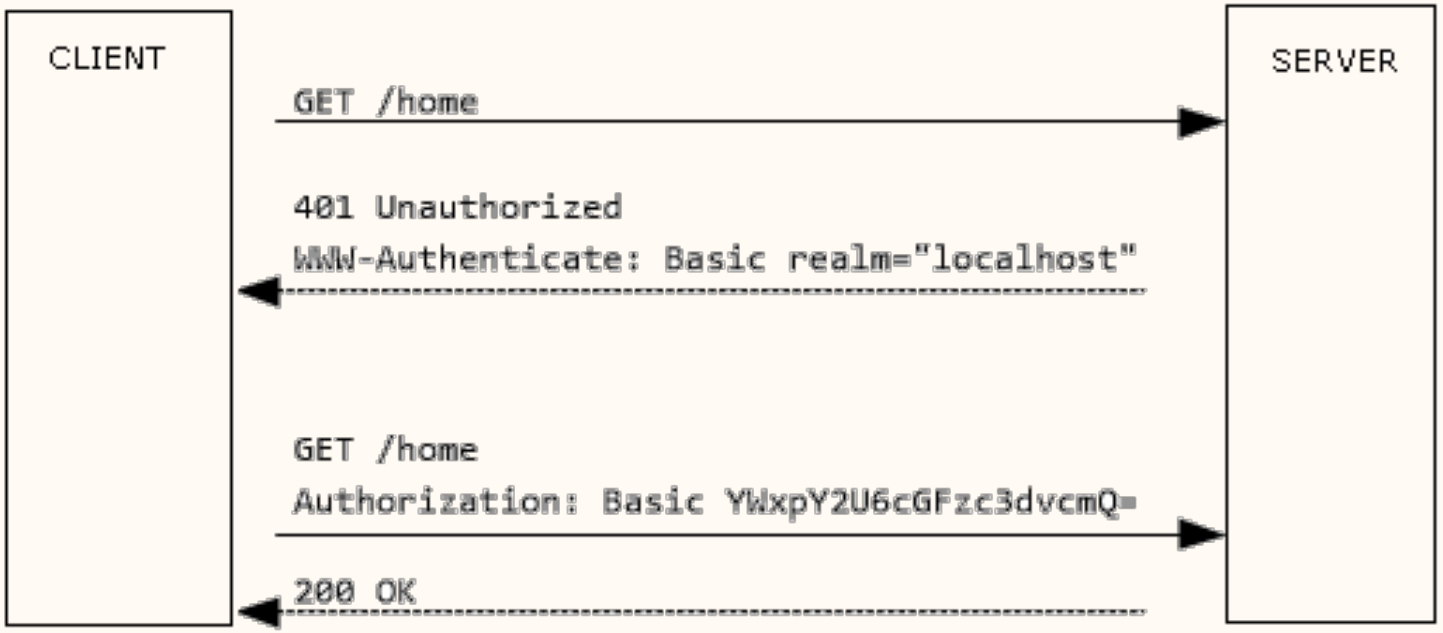
1. rész

HTTP ‘basic‘ hitelesítés

HTTP hitelesítés

- A HTTP protokoll specifikál hitelesítésre használatos HTTP fejléceket (header):
  - `WWW-Authenticate` - egy szerver ezáltal küldi el a kliensnek a módszert, amellyel be kell jelentkeznie a rendszerbe. Formája `<type> realm=<realm>`, ahol a `type` egy elfogadott bejelentkezési módszer, pl `basic`, `bearer`, `oauth`, stb. (teljes lista)
  - `Authorization` - egy kliens ezáltal küldi a szervernek a bejelentkezési információját. Formája `<type> <credentials>`, ahol a `type` egyenértékű a fentivel, míg a `credentials` formáját a bejelentkezési mechanizmus diktálja.
- A HTTP `basic` módszer működése:
  1. A szerver ellenőrzi, hogy a HTTP kérés tartalmaz-e egy `Authorization` fejlécet.
  2. Ha nem, visszaküldi a 401-es státuszkódot, valamint a válasz fejlécében a `WWW-Authenticate` bejegyzést (`basic` típussal).
  3. A böngésző ennek hatására megjeleníti a bejelentkező ablakot.
  4. A beírt adatokat átalakítja és visszaküldi a szervernek. A `credentials` formája: *base 64 encoding* `:` *elválasztókarakterrel*
  5. Minden további kérés során a kérés fejlécében újra elküldi a login adatokat.

Hitelesítés: HTTP basic



forrás: dotnetthoughts.net

Hitelesítés: HTTP basic

Példa: `6-auth/basic_authentication`

```
// HTTP basic auth middleware
app.use((req, res, next) => {
  const auth = req.headers.authorization;
  if (auth) {
    // kibányásszuk a bejelentkezési infót a headerekből
    const b64auth = auth.split(' ')[1];
    const [username, password] = Buffer.from(b64auth, 'base64').toString().split(':');

    // megnézzük a helyességüket
    if (users[username] === password) {
      // Access granted...
      next();
      return;
    }
  }

  // Access denied...
  res.set('WWW-Authenticate', 'Basic realm="WebProg példa"');
  res.status(401).send('Authentication required');
});
```

Hitelesítés: HTTP basic

- **Előnyök**:
  - Egyszerűség – a HTTP protokoll része
  - Minden böngésző támogatja
- **Hátrányai**:
  - **Nem biztonságos:** információk nincsenek kódolva. Ezért csak HTTPS-sel együtt érdemes használni
  - Nem véd a hamis szerverekről érkező kérések ellen (*Man in the Middle*)
  - Nem testreszabható – a böngésző dönti el a bejelentkezési ablak kinézetét
  - Nincs egyszerű kilépési lehetőség, a böngésző cache-ét kell törölnünk (ezt megtehetjük böngészőben futó JavaScript kóddal)
- **Alternatívák**:
  - Sütiben (cookie) tárolt session ID
  - Web-token alapú hitelesítés (RFC 7519)

2. rész

Session (munkamenet)-alapú hitelesítés

Hitelesítés express-session-nel

Példa: 6-auth/session_authentication

```
import express from 'express';
import session from 'express-session';

// ezek az értékek adatbázisból jöjjenek
const users = { egyuser: 'egyjelszo', masikuser: 'masikjelszo' };

const app = express();
app.use(session({ secret: '142e6ecf42884f03', resave: false, saveUninitialized: true }));
app.use(express.urlencoded({ extended: true }));

// belépés
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (username && password && users[username] === password) {
    req.session.username = username;
    res.send('Login successful');
  } else {
    res.status(401).send('Wrong credentials');
  }
});
```

Hitelesítés express-session-nel

```
// kilépés
app.post('/logout', (req, res) => {
  req.session.destroy((err) => {
    if (err) {
      res.status(500).send(`Session reset error: ${err.message}`);
    } else {
      res.send('Logout successful');
    }
  });
});

// session auth middleware
app.use((req, res, next) => {
  if (req.session.username) {
    next();
  } else {
    res.status(401).send('Not logged in');
  }
});

// Minden hívás bejelentkezést kényszerít
app.get('/*', (req, res) => {
  res.send('Welcome, authenticated user!');
});
```

3. rész

JWT-alapú hitelesítés

JSON Web Token (JWT)

- RFC 7519-ben definiált standard
- két fél között kompakt, önleíró, biztonságos információcserét biztosít JSON objektumok formájában
- aláírást tartalmaz - annak szavatolása, hogy a küldött adatok megbízható forrásból származnak, és menet közben nem voltak módosítva
- megengedi az **állapot nélküli hitelesítés**t
- megengedi **külső hitelesítési szolgáltatók** használatát (pl. Facebook, Google, stb.)
- alapból **nem titkosítja** az adatokat—inkább arra való, hogy biztosítsa, hogy az adatok egy helyes hitelesítési szervertől származnak
- generálása kipróbálható a `https://jwt.io` segítségével
- node.js-ben használható a `jsonwebtoken` modul segítségével

JSON Web Token (JWT)

- szerkezete: `fejléc.payload.aláírás`
  - a **fejléc** tartalmazza az hash-elő algoritmus és a token típusát
  - a **payload** tartalmazza a hasznos információt (felhasználónév, lejárati idő, stb.)
  - az **aláírás** generálásához a fejlécben megadott algoritmust alkalmazzuk a `fejléc.payload` stringre egy **titkos kulcs** (secret) segítségével
- a fejléc és payload JSON-ban van tárolva, majd mindhárom elem *base64* URL-biztos formában van kódolva, így URL-ben is biztonságosan küldhető, s az elválasztó `.` karakter sem okozhat gondot
- a szerver ismeri a titkos kulcsot, így mindig biztos lehet benne, hogy a token valid-e (megfelel-e a megadott aláírás az elvárt aláírásnak).
- a hash-elő algoritmus lehet:
  - *szimmetrikus* - ugyanaz a secret szükséges az aláíráshoz és ellenőrzéshez, pl. `HS`
  - *aszimmetrikus* - kulcspárost használunk, ahol a *privát* kulccsal generálunk aláírást a tokenhez, s az ebből derivált *publikus* kulccsal ellenőrízhetjük ennek helyességét, pl. `RS`

JWT payload tartalom

- A payload kulcsai a **claim**ek
- Van pár beépített (mind opcionális) standard claim, de kiterjeszthető tetszőlegesen sajátosakkal:
  - `iss` (issuer): ki állította elő a tokent (pl. auth szerver hostja)
  - `sub` (subject): "felhasználó", akinek a token szól
  - `aud` (audience): kinek szól a token
  - `iat` (issued at time), `exp` (expiration time), `nbf` (not before time): kezdeti-lejárati idők (UNIX epoch szám mind)
  - `jti` (JWT ID): egyedi azonosító, hogy lehessen egyenként visszavonni JWT-ket

JSON Web Token (JWT)

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVC
J9.eyJzdWIiOiJjc2FiYS5zdWx5b2tAdWJ
iY2x1ai5ybyIsIm5hbWUiOiJDc2FiYSBTd
Wx5b2siLCJpYXQiOjE1MTYyMzkwMjIsImV
4cCI6MTUxNjI0MjYyMiwiaXNzIjoiaHR0c
HM6Ly9teS1hdXRoLXNlcnZlci5jb20vIiw
iYXVkIjoiaHR0cHM6Ly9teS1vdGhlci1hd
WRpZW5jZS5zZXJ2ZXIuY29tLyJ9.U4kwRQ
T8vWb2SaLGdY2D5Uu9Pzsy4pTWrEryRpdJ
lvKZgeP24w2hKvFBQiExVu-
E6eHKFrM3mCzZV2mt9kJ1M3cTn-
uppER4qs65bzsKJjvMcD0A_zfPmxpg0ADp
sX7z8uILvYM0R6Tl-pRJY5N1kA0Ex-
SProMwu2l9HLipYd9qKpOedoaNi2MdJj1a
dbYdm3Q5N-
ZF1jXxaVx7LIMxygNVNm7kR_FEUxaMuWEa
Gr-
zGiCm68RIvaq7KUhG3_MvuRbXHlZszOskf
jU3Oxki2zt_PjUmShn2reFfMkLcwW3IYgx
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
  "name": "Csaba Sulyok",
  "iat": 1516239022,
  "exp": 1516242622,
  "iss": "https://my-auth-server.com/",
  "aud": "https://my-other-audience.server.com/"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCA
```

JWT példa

Példa: `6-auth/jwt_authentication`

```
import express from 'express';
import jwt from 'jsonwebtoken';

const users = {
  egyuser: 'egyjelszo',
  masikuser: 'masikjelszo',
};
const secret = '1c28d07215544bd1b24faccad6c14a04';
// ...
// belépés
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  if (users[username] === password) {
    // sikeres belépés, JWT generálása
    const token = jwt.sign({ username }, secret);
    res.send(`Log in successful.
      Your JWT token is: ${token}`);
  } else {
    res.status(401).send('Wrong credentials');
  }
});
```

```
// erőforrás, melyet csak valid JWT tokennel érünk el
app.get('/restricted', (req, res) => {
  const auth = req.headers.authorization;
  if (auth) {
    // kibányásszuk a JWT tokent a headerből
    const token = auth.split(' ')[1];
    // verifikáljuk az aláírást
    jwt.verify(token, secret, (err, payload) => {
      if (payload) {
        console.log(`JWT successfully verified
          for ${payload.username}`);
        res.send(`Congratulations, your JWT
          token is valid`);
      } else {
        res.status(401).send('You are not logged in');
      }
    });
  } else {
    res.status(401).send('You are not logged in');
  }
});
```

4. rész

Engedélyezés (Authorization)

Engedélyezés

- A hitelesítés (**authentication**) folyamata azonosítja a felhasználót, s megengedi hogy biztosítsuk a kilétet.
- Az engedélyezés (**authorization**) egy már hitelesített felhasználó felhatalmozottságát határozza meg—mely endpointokat érheti el.
- Az engedélyezés általában **szerepkörökön** alapszik (*role-based authorization*).
  - Egy felhasználó beletartozik egy vagy több előre meghatározott szerepkörbe (standard felhasználó, karbantartó, adminisztrátor, stb.).
  - A szerver endpointokat védelmezzük azáltal, hogy csak bizonyos szerepkörű felhasználók érhetik el.
- Noha a node.js-express kombináció nem nyújt beépített implementációt szerepkör-alapú engedélyezésre, middleware-ek és routing segítségével könnyen megvalósítható.

Engedélyezés példa sessionek segítségével

Példa: `6-auth/authorization`

```
import express from 'express';
import session from 'express-session';

// ezek az értékek adatbázisból jöjjenek
const users = {
  egyuser: { password: 'egyjelszo', role: 'user' },
  masikuser: { password: 'masikjelszo', role: 'admin' },
};

// app felépítése

// belépés
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (username && password && users[username] && users[username].password === password) {
    req.session.username = username;
    req.session.role = users[username].role;
    res.send('Login successful');
  } else {
    res.status(401).send('Wrong credentials');
  }
});
```

Engedélyezés példa sessionek segítségével

```
// middleware az engedélyezéshez
function authorize(roles = ['user', 'admin']) {
  return (req, res, next) => {
    if (!req.session.role) {
      // a felhasználó nincs bejelentkezve
      res.status(401).send('You are not logged in');
    } else if (!roles.includes(req.session.role)) {
      // a felhasználó be van jelentkezve de nincs joga ehhez az operációhoz
      res.status(403).send('You do not have permission to access this endpoint');
    } else {
      // minden rendben
      next();
    }
  };
}

app.get('/restricted', authorize(), (req, res) => {
  res.send('Congrats, you are in an area only for users');
});

app.get('/restricted_admins', authorize(['admin']), (req, res) => {
  res.send('Congrats, you are in an area only for admins');
});
```