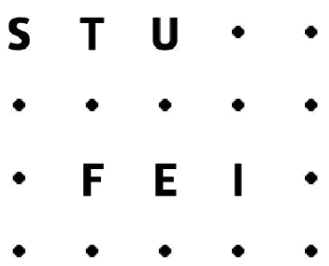


Zadanie č.3

Úvod do počítačovej bezpečnosti

Implementácia správy používateľských hesiel

2016/2017

Autor: Michal Drahovský, Ján Dzvoník

Obsah

1 Postup riešenia

1.1 Funkcie na generovanie a aplikovanie saltu

Salt je náhodne vygenerovaný reťazec ktorý sa prikladá k nezahashovanému heslu aby sa skomplikovali útoky zamerané na zlomenie hesla. Salt by mal byť pre každé nové heslo náhodný a unikátny. Vygenerovaný salt je následné potrebné si zapamätať aby sa s ním ďalej dalo overiť heslo.

Funkcie na generovanie saltu sme vytvorili v Classe Security. Vygenerovaný salt sa následne aplikuje pri hashovaní hesla a zapíše sa do súboru hesla.txt do stĺpca za zahashované heslo. Pre každé heslo sa generuje unikátny a náhodný salt.

Na encodovanie saltu sme využili knižnicu `com.sun.org.apache.xerces.internal.impl.dv.util.Base64;`
Na generovanie saltu sme využili knižnicu `java.security.SecureRandom, java.util.Random.`

Ukážka zdrojového kódu funkcie, ktorá generuje salt.

```
protected static String getNextSalt() {
  Random RANDOM = new SecureRandom();
  byte[] salt = new byte[16];
  RANDOM.nextBytes(salt);
  return Base64.encode(salt);
```

Ukážka zdrojového kódu, ktorý aplikuje salt a zavolá funkciu na zahashovanie hesla.

```
String salt = Security.getNextSalt();
String hashPassword=Security.getHashPassword(heslo, salt);
```

Vygenerovaný salt môže vyzerať napríklad: `uEtD3IBQ4XWSk1t1UfFOkA==`

1.2 Funkcie na zahashovanie saltového hesla

Z bezpečnostných dôvodov je potrebné jednotlivé heslá ukladať v nečitateľnej forme. Na tento problém sa používajú asymetrické hashovacie funkcie. To znamená, že sa nedajú spätne dešifrovať. V súčastnosti sa za bezpečnú hashovaciu funkciu považuje aj SHA-256. Túto funkciu sme si vybrali aj v našom zadaní.

Funkcie na zahashovanie saltového hesla sme doprogramovali v Classe Security. Táto funkcia zahashuje saltové heslo funkciou SHA256 a následne tento hash znova zahashuje ešte 2 krát taktiež funkciou SHA256.

Na hashovaciu funkciu SHA256 sme využili knižnicu `java.security.MessageDigest;`

Ukážka kódu generovania hesla

```
public static String getHashPassword(String pass, String salt){

    String hashPassword=null;
        for(int i=0;i<3;i++){
            if(i==0){
             hashPassword = sha256(pass + salt);
            }
            else{
                hashPassword = sha256(hashPassword);
            }
        }
        return hashPassword;
}
```

Ukážka kódu funkcie ktorá hashuje heslo funkciou SHA256

```
private static String sha256(String base) {
     try{
         MessageDigest digest = MessageDigest.getInstance("SHA-256");
         byte[] hash = digest.digest(base.getBytes("UTF-8"));
         StringBuffer hexString = new StringBuffer();

         for (int i = 0; i < hash.length; i++) {
             String hex = Integer.toHexString(0xff & hash[i]);
             if(hex.length() == 1) hexString.append('0');
             hexString.append(hex);
         }

         return hexString.toString();
     } catch(Exception ex){
        throw new RuntimeException(ex);
     }
  }
```

Zahashované heslo môže byť v tvare: `105544b0cfa33a1910b32c7cf910f705b5b66b3778fe4fdf8bdef75c0860fda1`

1.3 Funkcie na kontrolu zložitosti hesla

Kontrola zložitosti hesla je dôležitá kvôli ochrane užívateľského účtu. Ak by užívateľ nebol prinútený si zvoliť bezpečné heslo, niektorí lenivejší užívatelia, ktorým sa nechce pamätať si heslá, by volili veľmi slabé heslá. Takéto slabé heslá sú následne ľahko prelomiteľné pomocou brute force alebo slovníkovým útokom, čo predstavuje vysokú bezpečnostnú hrozbu. Následne by užívatelia mohli prísť o svoj účet alebo by mohli byť zneužité informácie z účtu.

Funkcie na kontrolu zložitosti hesla sme doprogramovali v Classe Security. Užívateľ je prinútený zvoliť heslo tak aby spĺňalo zadefinované pravidlá pre heslo.

Tieto pravidlá sú

- Dĺžka hesla musí byť od 8 po 16 znakov
- Heslo musí obsahovať aspoň 1 veľké písmeno
- Heslo musí obsahovať aspoň 1 malé písmeno
- Heslo musí obsahovať aspoň 1 číslicu

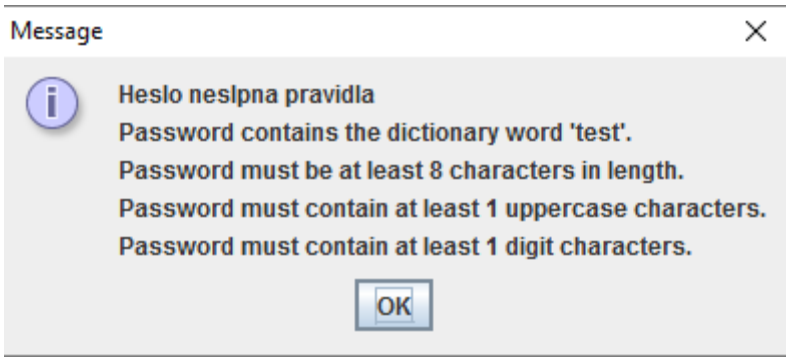
Heslo, ktoré síce spĺňa tieto požiadavky, stále môže byť slabé. Preto heslo, ktoré užívateľ zvolí, je ešte kontrolované slovníkom či neobsahuje často používané slová v heslách. Ak heslo spĺňa všetky požiadavky a neobsahuje slovo zo slovníka, užívateľ je úspešne registrovaný.

Pre kontrolu zložitosti hesla sme použili knižnicu - passay-1.2.0

Ukážka kódu kontroly zložitosti hesla

```
public static String checkPassword(String pass) {

     try {
         ArrayWordList awl;
         awl = WordLists.createFromReader(
                 new FileReader[] {new FileReader("dic.txt")},true,
                 new ArraysSort());

         WordListDictionary dict = new WordListDictionary(awl);

         DictionarySubstringRule dictRule = new DictionarySubstringRule(dict);
         dictRule.setMatchBackwards(true); // match dictionary words backwards

         List<Rule> ruleList = new ArrayList<>();
         ruleList.add(dictRule);
         ruleList.add( new LengthRule(8, 16));
         ruleList.add(new CharacterRule(EnglishCharacterData.UpperCase, 1));
         ruleList.add(new CharacterRule(EnglishCharacterData.LowerCase, 1));
         ruleList.add(new CharacterRule(EnglishCharacterData.Digit, 1));
         ruleList.add(new WhitespaceRule());

          PasswordValidator validator = new PasswordValidator(ruleList);

          PasswordData passwordData = new PasswordData(pass);
          System.out.println(pass);
          RuleResult result = validator.validate(passwordData);

          if (result.isValid()) {

              return "valid";

          } else {

              StringBuilder sb = new StringBuilder();
            for (String msg : validator.getMessages(result)) {

              sb.append(msg);
              sb.append("\n");

            }
            return "Heslo neslpna pravidla"+"\n" + sb.toString();
          }


     } catch (IOException ex) {
        return "Nepodarilo sa načítať slovník";
     }

}
```

Ak požívateľ nesplní požiadavky, je upozornený, ktoré pravidlá nespĺňa.



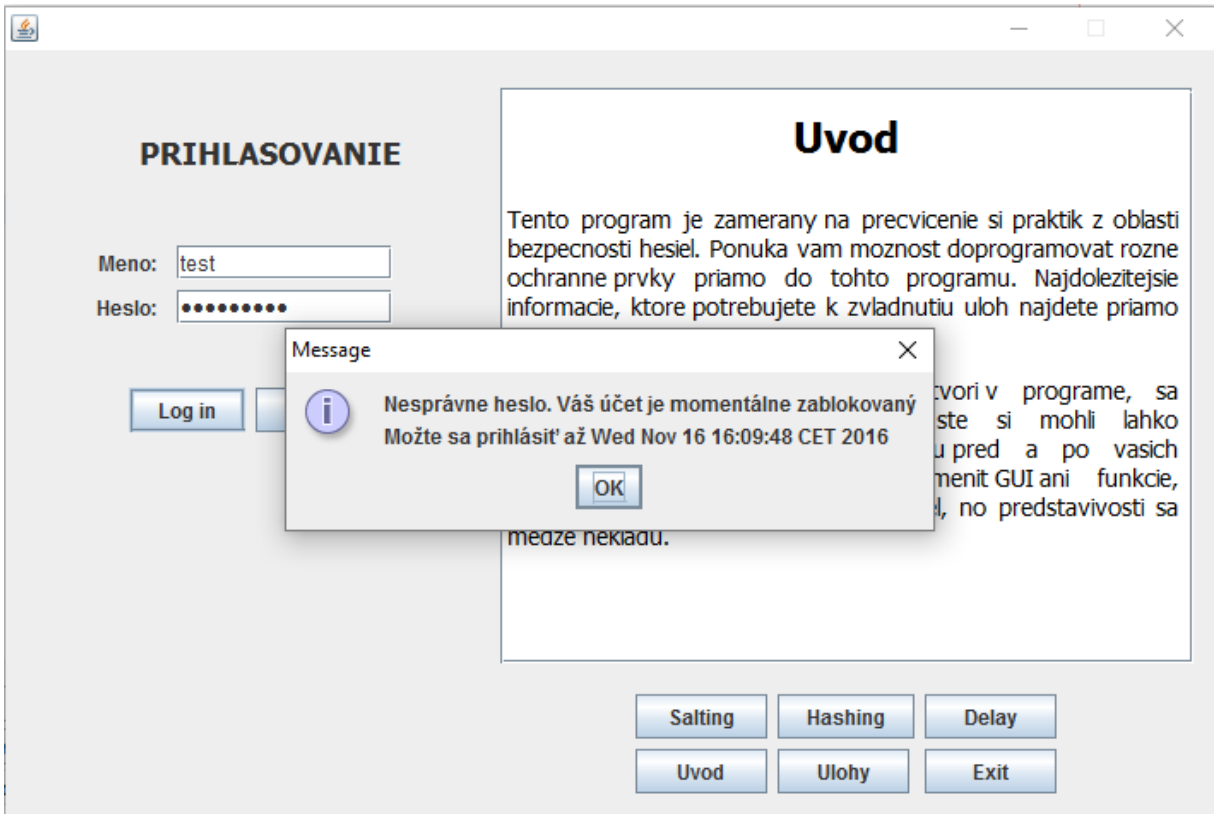
1.4 Funkcie na vytvorenie časového odstupu medzi pokusmi o prihlásenie

Vytvorenie časového odstupu medzi jednotlivými nesprávnymi pokusmi sa používa aby sa predišlo prelomeniu hesla technikou brute force. Jednotlivé pokusy sa zaznamenávajú v programe a následne sa obmedzí prístup k aplikácií pre zadaného používateľa. Ak sa používateľ prihlasuje veľa krát so zlým heslom, jeho čas, dokedy sa nemôže prihlásiť, sa zväčšuje o niekoľko sekúnd až minút. Pre užívateľa niekoľko sekúnd nepredstavuje veľké množstvo času ale pri počítačom riadenom útoku sa predĺži útok o podstatne viacej času.

Funkcie, ktoré kontrolujú správnosť údajov a vytvárajú časový odstup medzi prihláseniami, sme vytvorili v Class Login a funkcie, ktoré následne ukladajú informácie o jednotlivých uživateľoch a ich pokusoch o prihlásenie, v Class Database. Ďalej sme si vytvorili pomocnú Class WaitContants. Táto Classa iba uchováva čas vo formáte Date a počet sekúnd, o ktoré sa zvyšuje suspendovanie užívateľského konta.

Ak sa používateľ prihlási so zlým heslom, je pridaný do mapy, kde ako kľúč je login a hodnota je inštancia class WaitConstant s dátumom a časom dokedy musí čakať a počet sekúnd koľko má čakať. Ak používateľ znova zadá zlé heslo, aplikácia sa pozrie do mapy, či sa tam používateľ nachádza. Ak áno, tak skontroluje, či jeho čas, dokedy sa nemôže prihlásiť, je menši ako súčasný čas. Ak sa nenachádza v mape, tak sa používateľa pokúsi prihlásiť. Ak používateľova suspendácia vypršala, tak sa ho pokúsi prihlásiť. Ak sa znova zle prihlási, tak jeho čas sespendovania sa navýši o počet sekúnd, ktoré mu prislúchajú. Počet sekúnd koľko má čakať a čas suspendovania sa vždy zvyšuje pri každom zlom pokuse o prihlásenie alebo pri pokuse o prihlásenie pred vypršaním suspendácie. Mapa s používateľmi je vždy aktualizovaná a zapisovaná do databázy (súboru), pri každom pokuse o prihlásenie, aby sa predišlo tomu, že ak používateľ vypne aplikáciu, jeho suspendácia by sa stratila. Preto pri spustení aplikácie si načíta dáta zo súboru (attends.txt) a aktualizuje mapu.

Ak sa užívateľ prihlasuje počas suspendácie, je upozornený dokedy sa nemôže prihlásiť.



Ukážka kódu overovania prihlásenia

```
//if map contains username then check if its good login if not add wait time, else chceck if its good login
if(Database.getAtt().containsKey(meno)){
        //if user wiated setted time then ceck password if not add wait time
        MyResult usrWait= userWait(meno,currentTime,rightPassword);
        //write map to serializable file
        Database.writeMap("attends.txt");
        return usrWait;

}
else{
    if (!rightPassword)  {
        Database.getAtt().put(meno,new WaitContants(currentTime,1));
        Database.writeMap("attends.txt");
        return new MyResult(false, "Nespravne heslo.");
    }
    else{
        Database.writeMap("attends.txt");
        return new MyResult(true, "Uspesne prihlasenie.");
    }
}
```

Ukážka kódu funkcie, ktorá pridá k súčasnému dátumu počet sekúnd, ktoré ma užívateľ čakať

```
public static Date addTimeBySecondsDemo(Date date,int sec){

    Calendar calender = Calendar.getInstance();
    calender.setTimeInMillis(date.getTime());
    calender.add(Calendar.SECOND, sec);
    Date changeDate=calender.getTime();
    return changeDate;
```

Ukážka kódu funkcie, ktorá aktualizuje čas suspendácie používateľa

```
public static void addWaitTime(String meno, Date currentTime){

    // set new wait time in seconds
    Database.getAtt().get(meno).setSec(Database.getAtt().get(meno).getSec()+4);
   //get new time to wait current time + wait sec
   Date toWait=addTimeBySecondsDemo(currentTime,Database.getAtt().get(meno).getSec());
   //set new time to wait
   Database.getAtt().get(meno).setToWait(toWait);

}
```

Ukážka kódu funkcie, ktorá overí, či používateľ čakal zadaný suspendačný čas

```
public static MyResult userWait(String meno, Date currentTime,boolean rightPassword ){
   //check if  time to wait is waited then check password if its wrong add wait time, else add wait time
   if(Database.getAtt().get(meno).getToWait().before(currentTime)){
           if (!rightPassword)  {
                addWaitTime(meno,currentTime);
                return new MyResult(false, "Nesprávne heslo. Váš účet je momentálne zablokovaný"+
                        "\n"+"Možte sa prihlásiť až " + Database.getAtt().get(meno).getToWait().toString());
           }
           else{
                 Database.getAtt().remove(meno);
                 return new MyResult(true, "Uspesne prihlasenie.");
            }
      }
      else{
           addWaitTime(meno,currentTime);
           return new MyResult(false, "Nesprávne heslo. Možte sa prihlásiť až " +
                   Database.getAtt().get(meno).getToWait().toString());
      }

}
```

2 Záver

Je potrebné dodržiavať určité bezpečnostné pravidlá, ak chceme ochrániť našu aplikáciu pred únikom informácií. Preto aj pri tvorbe našej aplikácie sme sa snažili dodržiavať zadané bezpečnostné pravidlá. Funkcie, ktoré bolo potrebné doprogramovať, sme doprogramovali, aby sme našu aplikáciu ochránili pred ľahkým zlomením hesla účtov používateľov hackermi.

Prílohou zadania je priečinok, v ktorom sa nachádza NetBeans projekt so zdrojovými súbormi.