GSS-Übungsblatt 3 zum 14.05.2014

A. Struck, S. Haase, E. Böhmecke

14. Mai 2014

1 Rechnersicherheit

1.1 Zugangs- und Zugriffskontrolle

a)

Zugangskontrolle:
Beschränkung (durch spezielle Eigenschaften, Biometrie, Wissen, Besitz) wer Zugang zu einem System bekommt.
Zugriffskontrolle:
Beschränkung auf welchen Inhalt/Funktionalitäten im System ein User Zugriff hat.

b)

Dies kommt auf das System an. Keine Zugriffskontrolle impliziert dass jeder Nutzer auf der selben Vertrauensebene operiert, dies kann bei Systemen mit einer größeren Zahl von Anwendern zu großen Vertrauensproblemen führen. Allerdings ist nicht auszuschließen, dass Systeme gewünscht werden in denen jeder Nutzer die selben Rechte besitzt.

c)

Die Rechte eines unidentifizierten Nutzers (Zugangskontrolle) können nicht durchgesetzt werden, in so fern ist eine Zugangskontrolle vor Zugriffskontrolle von Nöten.

d)

Die Zugangskontrolle besteht hier durch die Kenntnis des Links, dies ist zwar schwächer, als eine Kombination aus Nutzerkonto und Passwort, erfüllt aber den selben Zweck.

2 Timing-Attack

1.

```
public void isTimingAttackPossible(){
  char[] password1 = "123456789".toCharArray;
  char[] password2 = "qwert".toCharArray;

  long pwTimeTemp = System.nanoTime();
  passwordCompare(password1, password1);
```

```
  long result = System.nanoTime() - pwTimeTemp;
  System.out.println("Gleiche Passwörter in ns: " + result);

  pwTimeTemp = System.nanoTime();
  passwordCompare(password1, password2);

  result = System.nanoTime() - pwTimeTemp
  System.out.println("Unterschiedliche Passwörter in ns: " + result);
}

boolean passwordCompare(char[] a, char[] b){
  int i;
  if(a.length != b.length) return false;
  for(i=0; i<a.length && a[i]==b[i]; i++);
  return i == a.length;
}
```

3.

Zuerst muss der Angreifer die richtige Länge des Passworts herausfinden. Hierzu schreibt er sich folgendes Programm,führt es mehrere Male hintereinander aus und überprüft welche der Char-Arrays am längsten bei der Ausführung der „passwordCompare“ Methode im Durchschnitt braucht.

```
char[] b = "abcdef".toCharArray();

for(int i = 0; i < 10; i++) {
timeElapsed = getTimeElapsed(new char[i], b);

System.out.println("attack " + i + ": " + timeElapsed + "ns.");
}

static long getTimeElapsed(char[] a, char[] b) {
long timeStart = System.nanoTime();
t.passwordCompare(a, b);
return System.nanoTime() - timeStart;
}

attack 1: 330ns.
attack 2: 331ns.
attack 3: 330ns.
attack 4: 330ns.
attack 5: 330ns.
attack 6: 661ns.
Attack 7: 330ns.
attack 8: 330ns.
attack 9: 331ns.
attack 10: 330ns.
```

In diesem Fall sieht man, dass die Berechnung mit einem Char-Array der Länge 6 am längsten gebraucht hat und diese somit unsere Passwortlänge verrät. In unserem Fall haben wir bis zur Passwortlänge 10 überprüft, man kann allerdings auch nach längeren Passwörtern suchen. Als nächstes erstellt der Angreifer ein Char-Array mit der herausgefundenen Passwortlänge (in unserem Fall 6) und ruft wieder die „getTimeElapsed“ Methode auf aber diesmal mit dem Char-Array der so groß ist wie die Passwortlänge und als ersten Char probieren wir alle Zeichen, die im Passwort enthalten sein könnten. Wir haben hier die ersten 10 Buchstaben im Alphabet durchprobiert und im Durchschnitt braucht die Berechnung mit einem „a“ als ersten Char am längsten.

```
attack a: 661ns.
attack b: 331ns.
attack c: 330ns.
attack d: 331ns.
attack e: 330ns.
attack f: 330ns.
attack g: 331ns.
attack h: 330ns.
attack i: 331ns.
```

Nun weiß der Angreifer mit welchem Zeichen das Passwort anfängt und kann im nächsten Schritt den selben Vorgang anwenden, allerdings diesmal mit dem ermittelten ersten Zeichen im Char-Array und durchprobieren welches das zweite Zeichen im Passwort ist. Dies wiederholt er so lange, bis er keine Zeichen mehr durchprobieren muss und die „passwordCompare“ Methode „true“ zurückliefert.