GSS Blatt 5

Riemenschneider, Hildebrandt, SoSe 16

Aufgabe 1.1

a) Zugangskontrolle: Beschränkung vom Zugang zu einem System z.B. durch Biometrie, Wissen, Besitz
   Zugriffskontrolle: Beschränkung auf welchen Inhalt/Funktionalitäten im System ein User Zugriff hat
b) In einem Startup mit wenigen Mitarbeitern wäre es durchaus sinnvoll, da man es sich gut vorstellen kann, dass jeder Mitarbeiter ohne Probleme auf alles zugreifen darf. Dort reicht nur die Zugangskontrolle, um Unbefugte den grundsätzlichen Zugang zum System zu verwehren.
c) Durch eine Zugriffskontrolle legen wir bereits fest, dass nicht jeder User auf alles zugreifen kann, d.h. dass ein nicht-identifizierter Nutzer auch auf nichts Zugriff bekommen wird und dadurch automatisch eine Zugangskontrolle in Kraft tritt.

Aufgabe 1.4

a) Rollen: Kunde/Außenstehender
   Verbreitung: Erhält Informationen durch den Strichcode, muss die Struktur erkennen, um Strichcodes zu fälschen
   Verhalten: aktiv, verändernd
   Rechenkapazität: unbeschränkt
b) 1. Angreifer schickt Bestellung an das System
   2. System erstellt einen Strichcode basierend auf den Informationen der Bestellung
   3. Angreifer erhält den Strichcode und System speichert Bestellungsdetails
   4. Angreifer erkennt das Verfahren, wie der Strichcode aus seinen Informationen erstellt wird
   5. Angreifer erstellt eigene Strichcodes
   6. Angreifer benutzt seine Strichcodes
   7. System zieht Bestellungsdetails aus den Strichcodes
   8. Falls es eine legitime Bestellung mit den gleichen Details gibt, wird dem Angreifer Einlass gewährt und das System setzt anschließend die Strichcodes auf “benutzt”

Aufgabe 2

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 das folgende Programm, führt es mehrere Male hintereinander aus und überprüft welche der Char-Arrays am längsten bei der Ausführung der „passwordCompare“ Methode im Durchschnitt brauchen:

```
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();

        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: 330ns.
Attack e: 330ns.
Attack f: 331ns.
```

```
Attack g: 330ns.
Attack h: 330ns.
Attack i: 330ns.
```

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.

Aufgabe 3

1)

Zeichenvorrat: 36

100 Studenten haben bereits eine Lösung hochgeladen, d.h. es wurden 100 Sicherheitscodes generiert und anschließend herausgegeben.

26 Zeichenlänge

36^{26} Möglichkeiten einen Sicherheitscode zu generieren

100 / 36^{26} = **$3.4366474 * 10^{-39}$%** Wahrscheinlichkeit einen von 100 generierten Sicherheitscodes zu erraten.