Aufgabe 1: Rechnersicherheit

Aufgabe 1.1: Zugangs- und Zugriffskontrolle

- a)

  Zungangskontrolle: Beschränkt, wer überhaupt Zugang zu dem System hat. Z.B.: Username/Passwort-Abfrage auf bei der Anmeldung auf einem Server.

  Zugriffskontrolle: Beschränkt, auf was genau im System der User Zugriff hat, nachdem ihm Zugang gewährt wurde.

- b) Grundsätzlich ist diese Möglichkeit nicht auszuschließen. Beispielsweise in Systemen mit wenigen oder einzig und allein vertrauenswürdigen Nutzern, oder in Systemen, in denen Benutzer einfach grundsätzlich nichts schlimmes anstellen können.
- c) Man kann die Zugriffsrechte eines Users nicht überprüfen/durchsetzen, wenn man ihn nicht vorher identifiziert hat. Um einen User zu identifizieren, benötigt man mindestens Zugangskontrolle.
- d) Es wird hier von einer Zugriffskontrolle auf *Ordner-Ebene* geredet. Das bedeutet, dass für bestimmte Ordner die Zugriffskontrolle *aufgehoben* wird. Diese Ordner sind dann öffentlich zugänglich. Ordner, die nicht freigegeben sind, sind weiterhin nur für eingeloggte Benutzer sichtbar, die Zugangskontrolle ist also weiterhin aktiv und es gilt das Statement aus aufgabe c).

Aufgabe 1.2: Biometrische Techniken: EasyPASS

- a)
- b)

  Webcam: kann manipuliert werden und fremdes Bild übermitteln.

Aufgabe 1.3: Tippverhalten

- a) Der Angreifer kann den Benutzer dazu bringen den Authentifizierungs-Satz auf einer eigenen Website einzugeben, um so das Tippverhalten mitzuschneiden. Dies kann er dann benutzen um sich zu authentifizieren.
- b) Eine Gegenmaßnahme könnte sein für jeden Benutzer einen zufälligen Authentifizierungs-Satz zu wählen, der dem Angreifer nicht bekannt ist.

Aufgabe 1.4: Realisierung eines Online Tickets

-

Aufgabe 2: Timing-Attack

- 1.

```
boolean isTimingAttackPossible()
{
    char[] pass1 = ['h','a','s','e'];
    char[] pass2 = ['t','i','g','e','r','e','n','t','e'];

    int t1 = System.nanoTime();
    passwordCompare(pass1, pass1);
    int t2 = System.nanoTime();

    int result1 = math.abs(t1 - t2);

    t1 = System.nanoTime();
    passwordCompare(pass1, pass2);
    t2 = System.nanoTime();

    int result2 = math.abs(t1 - t2);

    return result1 == result2;
}
```

  Man könnte beide Varianten von passwordCompare zwischen den Zeitmessungen mehrfach ausführen, um die Zeitunterschiede deutlicher zu machen.

- 2. Eine Timing-Attack ist hier möglich, da in passwordCompare entweder sofort abgebrochen, oder eine naive for-Schleife ausgeführt wird, welche stoppt sobald ein Zeichen nicht übereinstimmt. Dies kann zu sehr unterschiedlichen Ausführungszeiten führen.

- 3. Zunächst versucht der Angreifer, die Länge des Passworts herauszufinden. Hierfür probiert er der Reihe nach Passwörter verschiedener Länge (und mit beliebigem Inhalt) aus: `a`, `aa`, `aaa`, `aaaa` etc. Dabei sollten alle Passwörter bis auf eines gleich lang brauchen. Das eine Passwort, das länger braucht (da das Programm nicht sofort abbricht, sondern beginnt, die Passwörter Zeichen für Zeichen zu vergleichen), hat die richtige Länge.

  Nun beginnt der Angreifer, das erste Zeichen des Passwortes zu knacken. Hierfür probiert er Passwörter mit der richtigen Länge aus und ändert dabei immer das erste Zeichen, z.B. (bei einer Länge von 4) `aaaa`, `baaa`, `caaa`, `daaa` etc. Auch hier wird wieder ein Passwort länger brauchen als die anderen, da das Programm das erste Zeichen dieses Passwortes als richtig ansieht und noch (mindestens) das zweite Zeichen vergleicht. So kann der Angreifer sicher sein, dass das erste Zeichen dieses Passwortes das richtige ist und wendet nun die gleiche Methode auf das zweite Zeichen an. Nachdem er jedes Zeichen geknackt hat, besitzt der Angreifer das korrekte Passwort.

- 4.

```
// b ist die Benutzereingabe!
boolean passwordCompare(char[] a, char[] b)
{
    int e;
    int timefake;
    boolean correctLength = if(a.length == b.length);
    for (i=0; i < a.length; i++)
    {
        if (correctLength && a[i] == b[i])
        {
            e++;
        } elseif (a[i] == a[i])
        {
            timefake++;
        }
    }
    return  i == a.length ;
```

Aufgabe 3: Real-World-Brute-Force Angriff

Bei Testversuchen kam in dem Zeichenvorrat kein B, I, O und U vor. Dies kann aber auch an der geringen anzahl an Testversuchen (20) liegen. Bei 26 Buchstaben und 10 Ziffern wären unser Zwichenvorrat 36 Zeichen groß. mit 36 Zeichen der Länge 20 gäbe es 1336749453884373406783884597657 6 verschiedene Kombinationen. Bei 1000 Versuchen pro Sekunden würde es 42388047000000000000 Jahre dauern um alle Kombinationen durchzuprobieren. Selbst mit geringerem Zeichenvorrat von nur 10 Zeichen wären es noch 3170979200 Jahre.