IT-Security Lab 1

Abgabegruppe 22 (Kai Werk, Jan Hinrichs)

Aufgabe 1.1

Der Angriffsvektor ergibt sich aus der Kombination von der punishment Variable und dem sleep Befehl.
Der seed für rand() lässt sich nachstellen und so die gleiche “random” Zahl erzeugen. Zusätzlich lässt sich der Punishment-Faktor berechnen. Dadurch lässt sich durch Ausprobieren der einzelnen Stellen im Passwort herausfinden, welche Zeichen richtig sind, da die waittime Auskunft darüber gibt, ob ein Zeichen richtig geraten wurde oder nicht.

Vorgehen:

I. Manuelles Herausfinden der Passwortlänge anhand Returncode 2

1) Nehme ein Passwort, das garantiert an jeder Stelle falsch ist. Hier: “.....”
2) Lasse für jeden potentiellen Buchstaben an index k einen Subprozess für jedes mögliche Zeichen laufen, der das Zeichen einsetzt und gegen crackme laufen lässt
3) Berechne in jedem Subprozess anhand des Buchstabens und der PID die Zufallszahl und damit die worst-case waittime für jeden Subprozess
4) Ist der Prozess schneller fertig als worst Case, dann stimmte der Buchstabe den der Prozess an index k ausprobiert hat
5) Schreibe das passwort mit dem korrekten Zeichen in eine ausgabedatei und gib Erfolg zurück
6) warte auf alle prozesse, lese das Passwort aus der Datei und probiere das nächste Zeichen an index k+1 analog ab Schritt 2)
→ dadurch verringert sich auf die waittime, weil wir ja mit jedem richtigen buchstaben weniger punishment bekommen
7) Wenn alle Zeichen bearbeitet wurden, kennen wir das Passwort

Ein Brute-Force Angriff müsste, selbst wenn er die Länge des Kennworts kennen würde, alle möglichen Kombinationen für die 8 Stellen ausprobieren. Das ergibt bei Möglichkeiten a-z, A-Z und 0-9 dann 62 Möglichkeiten auf 8 Stellen, also 62^8 bzw O(62^n).

Im Fall des Seitenkanalangriffs gibt es 62 Möglichkeiten pro Zeichen, die aber nicht mit den anderen kombiniert werden müssen. Also 8*62 Versuche bzw O(n*62).

Aufgabe 1.2

Die crackme muss im selben Ordner wie die crack.sh liegen. Die derandomizer.c entweder selbst compilen oder das binary aus der .tar nutzen. Gegebenenfalls Berechtigungen zum Ausführen setzen. Das Ergebnis (Passwort) muss manuell an der crackme überprüft werden.

```
hinricja@gruenau5:~/itsec git:(main) (55.972s)
time crack.sh
current pw: ........ - attempting position 0 now
current pw: q....... - attempting position 1 now
current pw: q2...... - attempting position 2 now
current pw: q2a..... - attempting position 3 now
current pw: q2ao.... - attempting position 4 now
current pw: q2aoF... - attempting position 5 now
current pw: q2aoFd.. - attempting position 6 now
current pw: q2aoFdZ. - attempting position 7 now
password is q2aoFdZ8

real    0m55,903s
user    0m6,976s
sys     0m4,342s

hinricja@gruenau5:~/itsec git:(main) (1.666s)
./crackme q2aoFdZ8
Correct
```

Aufgabe 1.3

- Returncode für falsche Passwortlänge angepasst
- Punishment und reproduzierbaren random seed entfernt, der Angreifern basierend auf der sleep-Dauer falsche und richtige Zeichen im Passwort verrät

Code in der “fixed_source.c” zu finden.

```c
// fixed return code which told attackers password length
// removed punishment and reproduceable random seed _
//  which told attacker amount of wrong chars and therefore correct chars based on sleeptime
int compare_key() {
    unsigned int len = strlen(key_input);
    unsigned int correct_len = strlen(correct_key);
    int i, false_key = 0;

    if (len != correct_len) {
        false_key = 1;
    }

    for (i = correct_len - 1; i >= 0; i--) {
        if (key_input[i] != correct_key[i]) {
            false_key = 1;
        }
    }

    if (false_key == 1) {
        usleep(800000 * 15); // no "random", no info
    }

    return false_key;
}

```