**דוח מעבדה עקרונות תכנות מאובטח**

ניקוד: 5 נק' לרוב השאלות, 10 נק' לשאלות שמסומנות ב-(\*)

**חלק 1: REGEX**

קיראו על REGEX ב-Java: https://www.w3schools.com/java/java\_regex.asp וענו על השאלות הבאות:

1. איזה regex יבדוק אם טקסט מכיל אך ורק מילים מופרדות ברווח אחד או יותר?  
   רמז: זהו ה-regex לבדיקה האם טקסט מכיל אך ורק מילה אחת: ^[a-zA-Z]+$  
   רמז נוסף: ניתן לבדוק כאן: https://regex101.com

(תשובה כאן)

1. איזה regex מאפשר טקסט ללא טאגים של HTML למעט <a> עם אטריביוט href? (כלומר- ה-tag היחידי שמותר הוא <a> וה-attribute היחידי שמותר בו הוא href)

(תשובה כאן)

1. (\*) איזה עקרונות של PREVENT רלוונטיים בעת השימוש ב-regex? הסבירו בקצרה לגבי כל עקרון רלוונטי (שימו לב שעקרון יכול להיות רלוונטי מכיוון שהוא מתקיים, מכיוון שהוא לא מתקיים ומכיוון שהוא מכוון אותנו לפעולות שיש לבצע)

(תשובה כאן)

1. אילו פעולות (ספציפיות עבור regex) יש לבצע בעקבות עקרונות PREVENT שציינתם בתשובה הקודמת?

(תשובה כאן)

1. בצעו את הפעולות שציינתם בסעיף הקודם. ציינו כאן את תוצאות ביצוע פעולות אלו.

(תשובה כאן)

1. האם regex הוא דוגמא לעירבוב של קוד עם data?

(תשובה כאן)

1. (\*) נתון מימוש חלקי של ה-route:   
   http://localhost:8080/checkText?text=<some\_text>&tofind=<text>  
   למשל: http://localhost:8080/checkText?text=abc22def&tofind=bc

(הניחו שה-PARSING בוצע במקום אחר וערכי הפרמטרים כבר הוצאו מה-GET)

public boolean checkText(String text, String toFind) {

String regex = "\\d+";

boolean contained = Pattern.compile(regex).matcher(text).find();

return contained;

}

public handle(String request) {

// parse request

String text = …;

String toFind = ….

boolean response = checkText(text, toFind);

// send response using Http.sendResponse(…. response ….);

}

המימוש למעלה בודק האם text מכיל ספרות, למשל abc242jhids.

שנו את הגדרת ה-regex כך שיבדוק האם בטקסט מופיע התוכן של toFind שבצמוד אליו (לפניו או אחריו) יש ספרות. [שימו לב: יש דוגמאות בסוף ה-lab report]. רישמו כאן את ה-regex החדש

(תשובה כאן)

**חלק 2: UNICODE**

1. קראו את המאמר על חולשה ב-SPOTIFY:

https://engineering.atspotify.com/2013/06/creative-usernames/

מה היה מקור החולשה?

(תשובה כאן)

1. בהמשך לשאלה הקודמת, מהי קנוניזציה/תקנון? Canonicalization

(תשובה כאן)

1. על פי המאמר, קנוניזציה Canonicalization של מה ב-UNICODE ניסו לבצע ב-sportify?

(תשובה כאן)

1. בהמשך לשאלה הקודמת, הדגימו קנוניזציה Canonicalization של טקסט בתחום אחר (שאיננו UNICODE)

(תשובה כאן)

**חלק 3: LOGGING**

1. ע"פ OWASP CHEAT SHEET ל-LOGGING, מהם העקרונות שיש ללכת לפיהם בכתיבה ללוג?  
   (כיתבו בקצרה! כל עיקרון בלא יותר משורה)

(תשובה כאן)

**חלק 4: ENCODING**

1. בתיקייה http נתון קובץ a.txt עם התוכן הבא: Hello\tWorld\nHappy to be here!  
   נתונה התוכנית הבאה:

import java.nio.file.Files;

import java.nio.file.Paths;

public class EncodingExample {

public static void main(String[] args) throws Exception {

System.out.println("Hello\tWorld\nHappy to be here!"); // (1)

System.out.println(Files.readString(Paths.get("a.txt"))); // (2)

}

}

האם ההדפסה ב-(1) תהיה זהה או שונה מההדפסה ב-(2)? נמקו! בהסבר התמקדו במי אחראי לטפל בקידוד/encoding.

(תשובה כאן)

1. בהמשך לשאלה הקודמת:   
   אם עניתם שההדפסה **כן** תהיה זהה, הסבירו איזה שינוי יש לבצע **בתוכנית** כך שעל תוכן הקובץ לא יבוצע פיענוח של ה-encoding אלא הוא יודפס בדיוק כמו שרואים אותו בעורך טקסט.  
   אם עניתם שההדפסה **לא** תהיה זהה, הסבירו איזה שינויים יש לבצע בתוכנית כך שעל תוכן הקובץ כן יבוצע פענוח encoding וההדפסה של תוכן הקובץ תהיה זהה לזו של ה-string.

(תשובה כאן)

1. מדוע כל כך קל לשכוח לבצע קידוד/encoding לפלט?

(תשובה כאן)

1. בהמשך לשאלה הקודמת:   
   חפשו באינטרנט פתרון (יש יותר מאחד) המבטיח שלא נשכח לבצע קידוד עבור html. פרטו מהו הפתרון והסבירו כיצד הוא מבטיח ביצוע encoding

(תשובה כאן)

1. בהמשך לפתרון שמצאתם המבטיח שלא נשכח לבצע encoding עבור html: תנו דוגמת קוד המדגימה קידוד אוטומטי של ערך של משתנה.

(תשובה כאן)

1. בהמשך לפתרון שמצאתם המבטיח שלא נשכח לבצע קידוד עבור html – באיזה עקרון של PREVENT הוא עומד?

(תשובה כאן)

**דוגמאות קלט/פלט ל-checkText**

[**http://localhost:8080/checkText?text=<text**](http://localhost:8080/checkText?text=%3ctext) **to search>&toFind=<text to find>**

**Examples that Return *true***

1. **Digits before toFind**:
   * **text**: "abc 123find xyz"
   * **toFind**: "find"
   * **Explanation**: find has digits (123) immediately before it.
2. **Digits after toFind**:
   * **text**: "start find456 end"
   * **toFind**: "find"
   * **Explanation**: find has digits (456) immediately after it.
3. **Digits on both sides of toFind**:
   * **text**: "abc 123find789 xyz"
   * **toFind**: "find"
   * **Explanation**: find has digits (123 before and 789 after) on both sides.
4. **Multiple occurrences, one valid**:
   * **text**: "no digits here find, but 42find works"
   * **toFind**: "find"
   * **Explanation**: Although the first find has no digits around it, the second occurrence (42find) has digits before it, making the result true.
5. **Non-alphanumeric characters with valid digit adjacency**:
   * **text**: "some text find789! more text"
   * **toFind**: "find"
   * **Explanation**: find has digits (789) immediately after it, despite the exclamation mark following.

**Examples that Return *false***

1. **No digits around toFind**:
   * **text**: "hello find world"
   * **toFind**: "find"
   * **Explanation**: find appears without any digits before or after it.
2. **Letters instead of digits around toFind**:
   * **text**: "abc findxyz"
   * **toFind**: "find"
   * **Explanation**: find has letters (xyz) immediately after it, not digits.
3. **Non-alphanumeric characters without digits around toFind**:
   * **text**: "start find! end"
   * **toFind**: "find"
   * **Explanation**: find is followed by !, not a digit, so the result is false.
4. **Only non-digit whitespace around toFind**:
   * **text**: "some text before find some after"
   * **toFind**: "find"
   * **Explanation**: find has only whitespace before and after it.
5. **Digits near toFind but separated by spaces**:
   * **text**: "text 123 find 456 more text"
   * **toFind**: "find"
   * **Explanation**: The digits (123 and 456) are not directly adjacent to find.