Seminar 2 Preparation

Regex

The second language concept we will look at is regular expressions (regex). We have already presented some studies on their use, and potential problems, above. The lecturecast also contains a useful link to a tutorial on creating regex. Re-read the provided links and tutorial (Jaiswal, 2020) and then attempt the problem presented below:

The UK postcode system consists of a string that contains a number of characters and numbers – a typical example is ST7 9HV (this is not valid – see below for why). The rules for the pattern are available from idealpostcodes (2020).

Create a python program that implements a regex that complies with the rules provided above – test it against the examples provided.

Examples:

- M1 1AA
- M60 1NW
- CR2 6XH
- DN55 1PT
- W1A 1HQ
- EC1A 1BB

Code

```
import re

def uk_postcodes(postcode):
    # This will check the regex pattern for UK postcodes
    fit = "^[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][A-Z]{2}$"

    # This will compile the regex pattern
    regex = re.compile(fit)

    # This will check the postcode to see if it fits the pattern
    if regex.match(postcode):
        print(f"{postcode} True")
    else:
        print(f"{postcode} False")

# This function should result in the match being found as true

print("\nRegex used: '^[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][A-Z]{2}$'")

print("\nTrue:")
uk_postcodes("M1 1AA")
uk_postcodes("M60 1NW")
uk_postcodes("CR2 6XH")
uk_postcodes("DN55 1PT")
uk_postcodes("W1A 1HQ")
uk_postcodes("EC1A 1BB")

# This function should result in the match being found as false
print("\nFalse:")
uk_postcodes("ABC 1234")
uk_postcodes("ZYXW 987")
uk_postcodes("SO16 ZFD")
uk_postcodes("8PA UR6")
uk_postcodes("S016 FD9")
uk_postcodes("16SO 9FD")
```

Output

Postcodes that are not 1 letter, 2 numbers, 1 number and 2 letters fail.
Postcodes that are not 2 letters, 2 numbers, 1 number and 2 letters fail.
Postcodes that are not 2 letters, 1 number, 1 letter, 1 number and 2 letters fail.

```
Regex used: '^[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][A-Z]{2}$'

True:
M1 1AA True
M60 1NW True
CR2 6XH True
DN55 1PT True
W1A 1HQ True
EC1A 1BB True

False:
ABC 1234 False
ZYXW 987 False
SO16 ZFD False
8PA UR6 False
S016 FD9 False
16SO 9FD False
```

How do you ensure your solution is not subject to an evil regex attack?

The OWASP® Foundation defines an evil regex attack as a pattern that has repetition in the grouping and inside the repeated group. Some examples given are:

- (a+)+
- ([a-zA-Z]+)*
- (a|aa)+
- (a|a?)+
- (.*a){x} for x \> 10

These can be vulnerable if the input is aaaaaaaaaaaaaaaaaaaaaaaa!.

To avoid an Evil Regex attack you can use multithreading and stop the attacker from killing anticipated processes. If a regex evaluation is taking a long time it should be stopped. You should also refrain from sharing regex patterns online and do not write code that fits the patterns for an Evil Regex attack.

You can share your responses with tutor for formative feedback or discuss it in this week's seminar. There will also be an opportunity to review your team's progress during the seminar.

https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS#

https://sec.okta.com/articles/2020/04/attacking-evil-regex-understanding-regular-expression-denial-service

Remember to record your results, ideas and team discussions in your e-portfolio.
Learning Outcomes

- Identify and manage security risks as part of a software development project.
- Critically analyse development problems and determine appropriate methodologies, tools and techniques (including program design and development) to solve them.
- Design and develop/adapt computer programs and to produce a solution that meets the design brief and critically evaluate solutions that are produced.