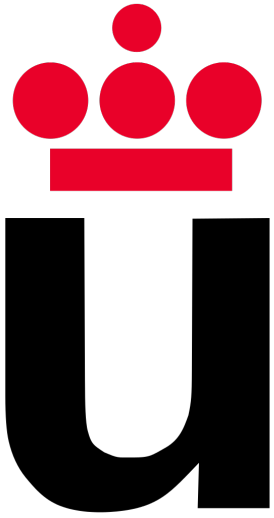
UNIVERSIDAD REY JUAN CARLOS



PRÁCTICA 2
ANÁLISIS ESTÁTICO

MÉTODOS DE DESARROLLO SEGURO

Por:
Javier Rojas, Gabriel Medrano

Contents

1 Expresiones regulares

En este apartado crearemos expresiones regulares para buscar, validar y dar respuesta a los siguientes ejercicios propuestos.

1.1 Ejercicio 1

En este ejercicio, creamos una expresión regular para validar todos los años correctos que aparecen en una línea de texto:

```
import re
text = input()
pattern = "\\b[0-9]{4}\\b|^\\b[0-9]{4}\\b"
results = re.findall(pattern, text)

for match in results:
    print(match)
```

1.2 Ejercicio 2

En este ejercicio, crearemos una expresión regular para validar todas las matrículas de una línea. Por ejemplo; E1337ZZZ, E-0000 PCB...

```
import re
text = input()
pattern = "(\\b(E|E-|E |)[0-9]{4}(-| |)[A-Z]{3}\\b)"
results = re.findall(pattern, text)

for match in results:
    print(match[0])
```

1.3 Ejercicio 3

En esta expresión regular, validaremos todas las fechas correctas que aparecen en una línea de texto, y cambiaremos su formato de 2023-04-16 a 16.04.2023 al imprimirlas:

```
import re
text = input()
pattern = "\\b(\\d{4}-\\d{2}-\\d{2})\\b"
results = re.findall(pattern, text)

for match in results:
    div = match.split('-')
    new = div[2] + "." + div[1] + "." + div[0]
    text = text.replace(match,new)

print(text)
```

1.4 Ejercicio 4

Con este ejercicio, determinamos cuándo se ha encontrado un email de alumno de nuestra universidad "@alumnos.urjc.es" o profesor "@urjc.es". La expresión regular es la siguiente:

```
import re
text = input()
pattern =   "\\b[a-z]\.[a-z]{2,}\.[0-9]{4}@alumnos\.urjc\.es\\b|" +
            "\\b[a-z]+\.[a-z]+@urjc\.es\\b"
results = re.findall(pattern, text)
alumno = "@alumnos"
profesor = "@urjc"
for match in results:
    div = match.split('.')
    if alumno in match:
        print("alumno " + div[1] + " matriculado en " + div[2][:4])
    else:
        div2 = div[1].split('@')
        print("profesor " + div[0] + " apellido " + div2[0])
```

1.5 Ejercicio 5

En esta expresión, validamos las calles válidas con diferentes formatos como "Calle Dulcinea N10, 28091", y las imprimimos "28926-Dulcinea-10":

```
import re
text = input()
pattern =   "\\b((Calle|C\\/)\\s([A-Z|\\u00C1-\\u00DA][a-z|\\u00E1-\\u00FA]+)" +
            "\\s*(,|)\\s+(Nº|nº|N |n |Nº |nº |n|N|)([0-9]+)\\s*(,|)\\s+([0-9]" +
            "{5}))\\b"
results = re.findall(pattern, text)
for match in results:
    print(match[7] + "-" + match[2] + "-" + match[5])
```

1.6 Ejercicio 6

En esta expresión regular, transformaremos cada línea de un .log a CSV, mediante:

```
import re
text = input()
pattern =   "\\b([0-9]{4}-[0-9]{2}-[0-9]{2}\\s[0-9]{2}:[0-9]{2}:[0-9]{2}." +
            "[0-9]{3}\\s+([A-Z]+)\\s[0-9]{7}\\s---\\s\\[([a-zA-Z0-9]+)\\]" +
            "\\s([a-z|\\.]+([A-Za-z]+)|[A-Za-z]+)\\s+:\\s([^\r\n]+))"
results = re.findall(pattern, text)
for match in results:
    if match[4] == '':
        print("\"" + match[1] + "\",\"" + match[2] + "\",\"" + match[3] +
        "\",\"" + match[5] + "\"")
    else:
        print("\"" + match[1] + "\",\"" + match[2] + "\",\"" + match[4] +
        "\",\"" + match[5] + "\"")
```

2 Integración Continua

2.1 Parte 1: Puesta en marcha del proceso de análisis automático

Una vez realizado todo el proceso de integración del SonarCloud con el repositorio WebGoat, debemos contestar a las siguientes preguntas.

2.2 Preguntas

• *1. ¿Cuantas vulnerabilidades, bugs y code smells ha detectado SonarCloud en el proyecto entero?*

Como podemos observar en la imagen, en nuestro proyecto SonarCloud ha encontrado un total de **30** vulnerabilidades, **32** bugs y **565** codesmells.

• *2. Haz click sobre el proyecto para abrir la vista de detalle. Revisa las vulnerabilidades detectadas y la lista de Security Hotspots. ¿Qué diferencia crees que existe entre las vulnerabilidades y los Security Hotspots?*

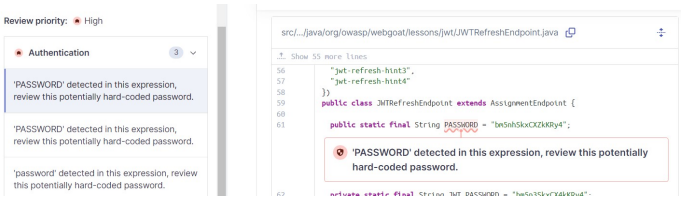
- **Vulnerabilidades**

Podemos encontrar vulnerabilidades como las siguientes:

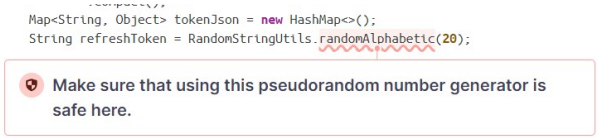
Nos indica que debemos **cambiar esas partes** del código.
Como podemos comprobar en los ejemplos, las vulnerabilidades son **debilidades** que pueden ser **explotadas** por **atacantes** para llevar a cabo sus ataques. Pueden ser errores en la programación, por no actualizar el código, etc.
Hay que **corregirlas lo antes posible**.

- **Security Hotspots**

Algunos ejemplos de Security Hotspots en el código son los siguientes:



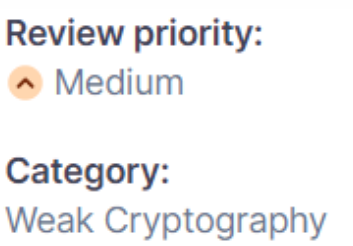
En esta imagen podemos ver como nos sugiere **revisar** esta parte de código ya que la variable **PASSWORD** está **hardcodeada**.



Aquí nos dice que revisemos que esa **función criptográfica** sea **segura**.

SonarCloud lleva la cuenta de los **Security Hotspots revisados** (no corregidos) y nos indica como corregir los posibles fallos:

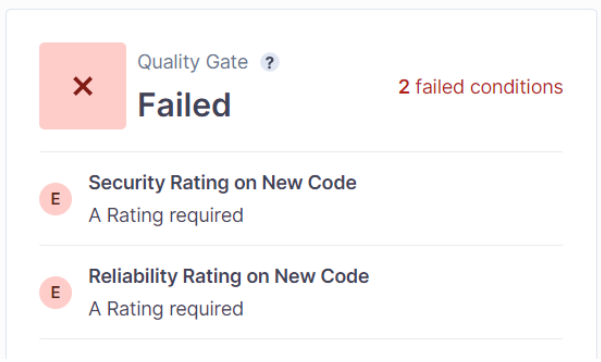
También se dividen los Security Hotspots por **prioridad** y **categoría**



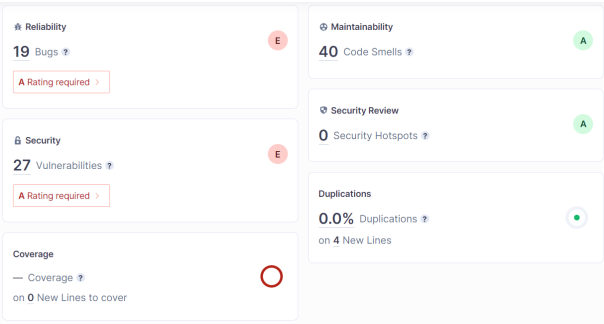
En resumen, los Security Hotspots son **zonas del código** que podrían ser **propensas** a **provocar errores de seguridad**, el programador debe de **revisarlas** y comprobar si deben ser corregidas o no. Las **vulnerabilidades** deben **corregirse** lo **antes posible** y los **Security Hotspots** deben ser **revisados** porque en un futuro pueden convertirse en vulnerabilidades.

• *3. Haz cualquier commit para forzar un reanalisis (por ejemplo editando el README.md). Espera a que finalice y vuelve a Sonar. ¿Cual es el estado del proyecto (Passed/Failed)?. ¿A qué crees que se debe? ¿Crees que el numero de vulnerabilidades afecta a dicho veredicto?*

- Editamos el fichero **README.md** añadiéndole una línea. Posteriormente creamos un **commit** "EditReadme" y hacemos **PUSH** a la rama Main. Tras actualizar el proyecto, vemos que el estado de este es **FAILED**:



- Esto se debe a que SonarCloud posee **diferentes definiciones de código** para determinar que cuenta como **código nuevo**. Seguidamente, verifica si dicho código **pasa** una serie de **condiciones** de código nuevo. Este Failed en nuestro proyecto implica que nuestros cambios en el README.md **no siguen estas condiciones** por defecto que posee Sonar en el proyecto.
Las condiciones que el código no pasa las podemos ver reflejadas en Sonar:



- El **número de vulnerabilidades NO afecta** a que nuestro proyecto sea Failed. Ya que Sonar sólo define las condiciones en el código nuevo y no tiene en cuenta el código antiguo. Las vulnerabilidades son código antiguo que no estamos añadiendo en el commit, y nos da Failed. Esto implica que las vulnerabilidades anteriores no afectan a este Failed del proyecto.

2.3 Parte 2: Mitigación

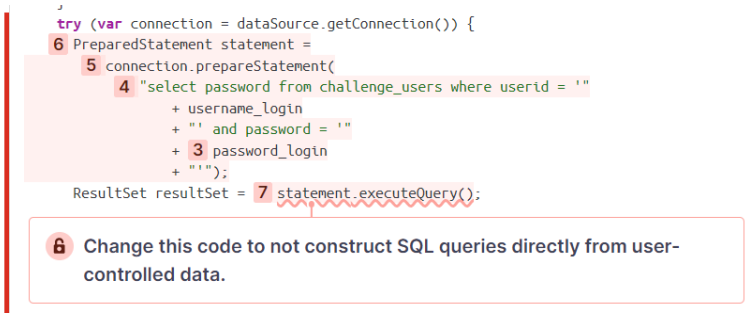
2.4 Preguntas

• *1. Elige una vulnerabilidad de tipo Blocker o Critical, explica cual es la vulnerabilidad detectada, porqué ha sido detectada (y si realmente es una vulnerabilidad y no un falso positivo).*

La primera vulnerabilidad que hemos escogido es la siguiente:

Se trata de una vulnerabilidad **Zip Slip** la cual se produce cuando se extrae un archivo comprimido que contiene **rutas maliciosas** o malintencionadas, permitiendo al atacante **sobrescribir archivos**.

Podemos ver que **no es un falso positivo** ya que la línea que nos sugiere cambiar hace lo siguiente:
Files.copy sirve para **copiar archivos**, se le pasa los parámetros ***is*** (el **archivo** que se va a copiar), ***f.toPath()*** (**ruta y nombre** del archivo de destino) y ***StandardCopyOption.REPLACE_EXISTING*** (opción que indica que **si el archivo ya existe que lo reemplace** por el que se está copiando).
Por lo tanto, si descomprimes ese zip, su **contenido** se te guardará en la **ruta indicada** y si hay un archivo con el **mismo nombre** del que contenga el zip te **lo reemplazará** por el nuevo archivo.
La segunda vulnerabilidad que hemos cogido es una **inyección SQL** de tipo Blocker:



Este tipo de vulnerabilidades permiten al atacante **inyectar código arbitrario** mediante las variables del código "username_login" y "password_login". **No es un falso positivo** ya que podemos apreciar que las variables, al sumar los strings, no siguen ningún control de seguridad y no podemos garantizar que no lleven código inyectado. Ese código se fía del usuario.
• *2. Propón una solución e implémentela en una nueva rama del repositorio. Explica los cambios que arreglan dicha vulnerabilidad.*

Para **solucionar la vulnerabilidad del Zip Slip** podemos sustituir la línea que nos indica por el siguiente código:

```
String canonicalDestinationPath = f.getCanonicalPath();
    if(canonicalDestinationPath.startsWith(tmpZipDirectory.toString())){
            Files.copy(is, f.toPath(),
            StandardCopyOption.REPLACE_EXISTING,
            LinkOption.NOFOLLOW_LINKS);
    }
```

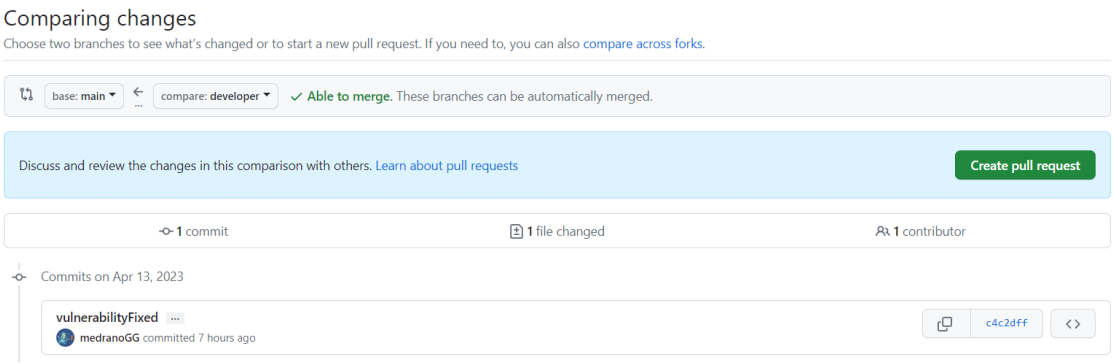
Este código lo que hace es **comprobar** que la **ruta del archivo coincide con el inicio de la ruta destino**. Si es así se copiará el contenido del archivo sustituyendo los archivos si ya existen.
La opción ***LinkOption.NOFOLLOW_LINK*** (hay que importar java.nio.file.LinkOption) indica que los **enlaces simbólicos no se seguirán** al copiar el archivo.

Para **solucionar la vulnerabilidad SQL Inyection** podemos sustituir la línea que nos indica por el siguiente código:

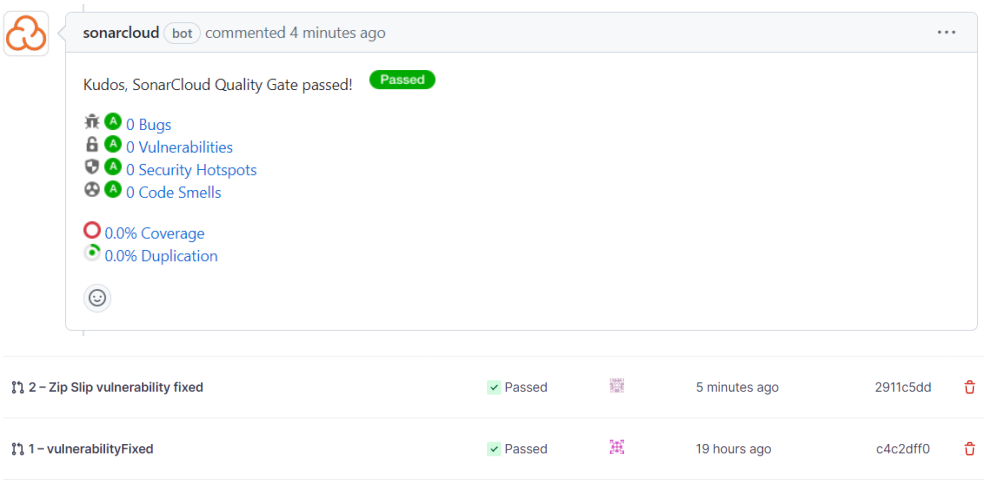
```
try (var connection = dataSource.getConnection()) {
      PreparedStatement statement =
          connection.prepareStatement(
              "select password from challenge_users where userid = ? and password = ?");
      statement.setString(1, username_login);
      statement.setString(2, password_login);
      ResultSet resultSet = statement.executeQuery();
```

Este código, en la **query**, nos permite **comprobar que el usuario ha introducido correctamente las credenciales** usuario-password, y que no ha inyectado código adicional. Las **"?"** compueban esto.

• *3. Crea una Pull Request de la nueva rama a la rama principal. ¿Qué opina Sonar de los cambios realizados?*



Al crear la Pull Request desde nuestra rama "developer" a la rama main Sonar nos indica que **ha analizado el nuevo código y que pasa el análisis:**



• *4. Junta (merge) la nueva rama con la rama principal. Una vez finalizado el análisis, ¿qué cambios se han producido en el proyecto? ¿Cuantas vulnerabilidades detecta ahora?*

Ahora podemos ver como al solucionar la vulnerabilidad, ahora **detecta 28 vulnerabilidades** (dos menos de las que había antes):

Lo correcto sería ahora ir **corrigiendo** el **resto de vulnerabilidades** igual que hemos hecho con esta y **revisar los Security Hotspots** de nuestro proyecto.

3 Bibliografía

https://github.com/jrhjavier/WebGoat
https://github.com/jrhjavier/MDS_P2
https://sonarcloud.io/project/overview?id=jrhjavier_WebGoat
https://es.overleaf.com/learn/latex/Tutorials
https://www.redeszone.net/2018/06/06/vulnerabilidad-zip-slip/