Práctica Análisis Estático

Metodologías de Desarrollo Seguro, Ingeniería de la Ciberseguridad

Carlos Barahona Pastor y Ángel del Castillo González, Grupo Q

Índice

4. Junta (merge) la nueva rama con la rama principal. Una vez finalizado el analisis, ¿qué cambios se han producido en el proyecto? ¿Cuantas vulnerabilidades detecta ahora? . 10

Enlaces Repositorios

https://github.com/c4rl0s01/PracticasMDS

https://github.com/c4rl0s01/WebGoat

Expresiones regulares

Ejercicio 1.

Enunciado

Dado un texto de una sola línea, determinar todos los años (números formados estrictamente por 4 dígitos) que aparecen. Un año es una cadena numérica de longitud 4, con cualquier valor en el rango [0000, 9999]. Se tendrán que imprimir por pantalla en el lenguaje deseado usando una expresión regular todos los años que vienen en el texto en orden de aparición, en una línea cada uno.

Resolución

El código correspondiente a este ejercicio es el siguiente:

```
import re

new = input()

pattern = r"\b\d{4}\b"

results = re.findall(pattern, new)
for match in results:
    print(match)
```

En el código anterior, la expresión regular comienza y termina con `\b` (para indicar que debe haber un separador en esos 2 puntos) y en el medio `\d{4}` que hace referencia a que tiene que haber 4 dígitos (cada uno será cualquier número del 0 al 9) para que se cumpla el rango del enunciado `[0000, 9999]`. Esta expresión regular se usa para buscar todas las coincidencias mediante la función `re.findall()` y después mediante un `for` se imprimen todas las coincidencias.

Ejercicio 2.

Enunciado

Dado un texto de una línea, determinar todas las matrículas que aparecen. Una matrícula es una cadena que tiene las siguientes características:

- 4 dígitos seguidos de un separador (guion, espacio o nada) y 3 letras en mayúsculas al final tal que [0000 - AAA, . . . , 9999 - ZZZ].
- Las matrículas pueden llevar una E mayúscula delante para indicar que se trata de un vehículo especial. Ejemplos: E1337ZZZ, E-0000 PCB.
- Los dígitos pueden estar separadas de las letras utilizando un guion, un espacio, o ningún separador. Se tendrán que imprimir por pantalla usando el lenguaje elegido usando una expresión regular todas las matrículas que vienen en el texto en orden de aparición

Resolución

Para la resolución del ejercicio, hemos escrito el siguiente código:

```
import re
```

```
new = input()
pattern = r'(\bE[-|\s|\S]?[0-9]{4}[|\s|\-]?[A-Z]{3})|([0-9]{4}[-|\s|\S]?[A-Z]{3}\b)'

x = re.findall(pattern, new)


for match in x:
    if match[1] != '':
        print(match[1])
    elif match[0] != '':
        print(match[0])
```

En la anterior expresión regular se diferencian dos grupos, en el primero tendremos que buscar matrículas que comiencen con E y estén seguidas de un guión, espacio o ningún espacio, "[-|\s|\S]", a continuación tendrá 4 números "[0-9]{4}" seguidos de un espacio, guión o ningún espacio, después de esto deberá tener tres letras mayúsculas "[A-Z]{3}". En el último grupo tendremos solamente 4 números "[0-9]{4}" seguidos de espacio, guión o ningún espacio, finalizando con tres letras mayúsculas "[A-Z]{4}"

Ejercicio 3.

Enunciado

Dado un formato de fechas yyyy-mm-dd, se pide convertir a dd.mm.yyyy. Para cada match encontrado en los documentos propuestos se tendrá que imprimir en el siguiente formato (los rangos de fecha puede ser erróneos, pueden existir un mes 20).

Resolución

El código para realizar este ejercicio es el siguiente:

```
import re

new = input()

pattern = r"\b(\d{4})-(\d{2})-(\d{2})\b"
sub_pattern = r'\3.\2.\1'
results_replaced = re.sub(pattern, sub_pattern, new)
print(results_replaced)
```

En este caso se tienen 2 expresiones regulares:

- `pattern`: en esta expresión lo que se hace es buscar fechas en el formato solicitado (yyyy-mm-dd) usando para ello grupos que contienen `\d` para indicar que tiene que haber un dígito, y a continuación de cada `\d` poniendo `{numero}` siendo número la cantidad de dígitos dependiendo de si es el correspondiente al año, mes o día. Además entre los grupos hay guiones (`-`) que es el formato solicitado y `\b` al principio y al final de la expresión indicando que tiene que haber un separador en cada punto.
- `sub_pattern`: esta expresión se usará para reemplazar todas las fechas con el nuevo formato solicitado (dd.mm.yyyy). Para ello, se usa `\grupo` siendo grupo el número correspondiente a cada grupo para cambiar su orden y poniendo un `.` entre medias de cada grupo ya que es el nuevo carácter de separación.

Una vez vistas las 2 expresiones, se usa la función `re.sub` para reemplazar las fechas que encuentre la primera (`pattern`) con la segunda (`sub_pattern`). Tras eso se imprime el resultado obtenido tras la sustitución.

Ejercicio 4.

Enunciado

Dado un texto, determinar cuando se ha encontrado un email de alumno de nuestra universidad "@alumnos.urjc.es" o profesor "@urjc.es". Los emails de alumnos están formados del siguiente patrón (puedes asumir que el input siempre estará en minúscula):

- Inicial del usuario, seguido de punto.
- Apellido del usuario, siempre mayor o igual a 2 caracteres.
- Seguidos de un punto y la fecha de matriculación. ´
- todos finalizan con "@alumnos.urjc.es".

Los correos de los profesores constan de:

- Nombre del profesor seguido de un punto.
- Apellido del profesor.
- Finalizando con "@urjc.es".

Para cada match encontrado se tendrá que imprimir en el siguiente formato:

- Para el caso de prueba i.lozanoo.2015@alumnos.urjc.es reportaremos "alumno lozanoo matriculado en 2015"
- Para un profesor reportaremos para el ejemplo isaac.lozano@urjc.es "profesor isaac apellido lozano"

Resolución

Para la resolución del ejercicio, hemos escrito el siguiente código:

```
import re

new = input()

pattern = r"\b(([a-z]{1,})\.([a-z]{2,})@urjc\.es)|\b(([a-z]{1}) \
            \.([a-z]{2,})\.([0-9]{4})@alumnos\.urjc\.es)"

results = re.findall(pattern, new)


for match in results:
    if match[0] != "":
        print("profesor "+match[1]+" apellido "+match[2])
    else:
        print("alumno "+match[5]+" matriculado en "+match[6])
```

En este ejercicio tenemos dos grupos principales, el correo de profesor y el correo de alumno. En el grupo del profesor, creamos dos grupos internos que son el nombre el cual está compuesto por letras minúsculas "[a-z]{1,}" seguido de un punto, y el siguiente grupo que es el apellido compuesto por letras minúsculas "[a-z]{1,}" finalizando con @urjc.es. El segundo grupo principal es el correo de alumnos que también está formado por otros grupos internos, en este caso tres, y son: la letra del nombre "[a-z]{1}", seguido de un punto, el apellido "[a-z]{1,}" seguido de un punto, y el año de matriculación,"[0-9]{4}", finalizando con @alumnos.urjc.es. Para mostrar la salida según el formato pedido, accedemos al grupo con el que hacemos el match.

Ejercicio 5.

Enunciado

Dado un texto devolver las direcciones postales. Una dirección estará compuesta de una calle representada por "C/" o "Calle" seguido de un espacio con el nombre de la calle (una sola palabra) donde la primera letra debe estar en mayúscula, opcionalmente una coma, un numero arbitrario de espacios, el número en cualquiera de los siguientes formatos (Nº7, nº7, N 7, n 7, 7, Nº 7, nº 7, n7, N7). No es valido Nº7, n º7, ni º7, la N podría estar en mayúsculas o minúsculas. Seguido, una coma, un número arbitrario de espacios y un numero de 5 dígitos correspondiente a un código postal. Los nombres de las calles deben poder validar calles de Madrid formadas por una sola palabra, no es necesario que reconozca "Calle Almendro Azul", porque el nombre de la calle tiene dos palabras en vez de una. Ejemplos de casos:

- "C/ Dulcinea Nº 10, 28936"
- "Calle Dulcinea 10, 28106"
- "Calle Dulcinea N10, 28091"

Para cada calle encontrada se reportara: "CP-Calle-Numero", por ejemplo: "28926-Dulcinea-10".

Resolución

El código usado para la resolución es el siguiente:

```
import re

new = input()

pattern = r"\b(C\/|Calle)\s([A-ZÁÉÍÓÚÑ][a-záéíóúñ]+),?\s+" \
          r"(Nº|nº|N\s|n\s|Nº\s|nº\s|n|N)?(\d+),\s+([0-9]{5})\b"
results = re.findall(pattern, new)
for match in results:
    print(match[4], match[1], match[3], sep='-')
```

En este caso la expresión regular utilizada esta compuesta por varias partes que se explican a continuación:

- `\b` al principio y al final para indicar que debe haber un separador en los extremos del contenido que encuentre la expresión.
- `(c\/|Calle)\s` hace referencia a que se debe comenzar o bien con `C/` o con `Calle` y que a continuación debe haber un espacio `\s`
- `([A-ZÁÉÍÓÚÑ][a-záéíóúñ]+)` es el grupo con el nombre de la calle. La calle debe empezar con una letra mayúscula pudiendo incluir vocales con tilde o la Ñ (lo cual se representa con `[A-ZÁÉÍÓÚÑ]`) seguida de una o más (`+`) letras minúsculas (`[a-záéíóúñ]`).
- `,?\s+` representa que después del nombre de la calle puede ir opcionalmente una coma (`,?`) y uno o más espacios (`\s+`).
- `(Nº|nº|N\s|n\s|Nº\s|nº\s|n|N)?` este grupo hace referencia a los formatos que pueden ir delante de los dígitos que representan el número de una calle, y pueden ser opcionales ya que como indica el enunciado el número puede ir solo.
- `(\d+),\s+` es el grupo que se refiere al número de la calle, es decir uno o más (`+`) dígitos (`\d`) seguidos de una coma (`,`) y un número arbitrario de espacios (`\s+`).
- `([0-9]{5})` es el último grupo y se refiere a un número de 5 dígitos (`{5}`) del 0 al 9 (`[0-9]`) que indican el código postal.

Como se puede observar en el código, tras la expresión regular, se usa `re.findall()` para encontrar las coincidencias que detecte la expresión. Por cada coincidencia se imprime el código postal (`match[4]`), el nombre de la calle (`match[1]`) y el número de la calle (`match[3]`) separados por un guión.

Ejercicio 6.

Enunciado

Dado un fichero de logs, transformar cada línea a CSV extrayendo la siguiente información:

```
- Nivel de log
- Hilo donde se ha producido el log (este hilo se corresponde
  con letras en mayúscula, minúscula y de números)
- Clase responsable de emitir el log
- Mensaje de log
```

Resolución

Para la resolución del ejercicio, hemos escrito el siguiente código:

```
import re

new = input()

pattern = r"([A-Za-z]{1,}).{1,}\[([a-zA-Z0-9]{1,})\]\s+(.{1,}\S)\s+\:\s+(.{1,})"

results = re.search(pattern, new)

if results:
    level_log = results.group(1)
    thread_log = results.group(2)
    class_log = results.group(3)
    message_log = results.group(4)

    if class_log.__contains__("."):
        class_split = class_log.split(".")
        l = len(class_split) - 1
        class_log = class_split[l]

    print(f'"{level_log}","{thread_log}","{class_log}","{message_log}"')
```

En este ejercicio diferenciamos cuatro grupos: nivel, hilo, clase, mensaje. El nivel lo encontramos con las primeras letras que haya en el log y puede contener mayúsculas o minúsculas las veces que sea "[A-Za-z]{1,}", con ".{1,}" seleccionaremos todo lo contenido en el log hasta encontrar el segundo grupo. El hilo se encuentra entre corchetes y puede contener mayúsculas, minúsculas o números, "[a-zA-Z0-9]{1,}", seguido de un número de espacios encontraremos el tercer grupo. La clase cogemos todos los caracteres que haya hasta el siguiente carácter que no sea un espacio,"(.{1,}\S)", seguido de un número de espacios hasta ":" y unos espacios después encontramos el último grupo. Para el mensaje que se encuentra después de los ":" y espacios, el mensaje será todos los caracteres siguientes hasta terminar el log ".{1,}". Para tener el output pedido hemos hecho un tratamiento de la clase, ya que puede contener caracteres y puntos, y solo nos interesa los últimos caracteres, por lo tanto, si contiene puntos, dividimos cada el resultado por cada punto y nos quedamos con el último miembro del array obtenido.

Integración Continua

Preguntas Puesta en marcha del proceso análisis automático

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

Estos son los datos que ha arrojado SonarCloud:

- Vulnerabilidades: 30
- Bugs: 32
- Code smells: 565

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?

La diferencia es que las **vulnerabilidades** hacen referencia a código que se ha detectado que puede ser explotado, mientras que los **Security Hotspots** son fragmentos de código que deben ser revisados, ya que aunque no se haya podido determinar a priori si son una vulnerabilidad o no en el escaneo, podrían serlo por lo que deben analizarse de forma manual para determinarlo.

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?

Tras editar el archivo README.md, el estado del proyecto es `Failed`. Esto se debe a que para que el estado de un proyecto sea `Passed`, se tienen que cumplir una serie de condiciones de la `Quality Gate`. En este caso específico, las condiciones que no se cumplen son:

- `Reliability Rating`: este apartado hace referencia al número de bugs (errores de código que hacen que no funciones) que se han detectado en el código analizado. En el caso de esta métrica se requiere que el valor no sea menor que `A` y el valor que ha arrojado el análisis del código es de `E` (19 bugs)
- `Security Rating`: este apartado hace referencia al número de vulnerabilidades (código que puede ser explotado por adversarios) que se han detectado. Esta métrica requiere un valor que no sea menor que `A`, y en este caso el valor obtenido es `E` (27 vulnerabilidades).

Efectivamente el número de vulnerabilidades afecta a dicho veredicto, ya que el hecho de que existan vulnerabilidades hacen que no se pueda obtener una puntuación de `A` (la cual se corresponde con 0 vulnerabilidades). Con los bugs pasa exactamente lo mismo, la puntuación de `A` se obtiene con 0 bugs.

Preguntas Mitigación

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

Para este ejercicio una de las vulnerabilidades que hemos escogido ha sido la de "Default PasswordEncoder relying on plaintext". La vulnerabilidad ha sido encontrada en el siguiente fragmento de código:

```
@Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService);
  }
```

Es vulnerable puesto que si nos roban la base de datos de nuestros usuarios y sus contraseñas, las verán en texto plano, y podrán suplantar la identidad de cualquier usuario. Para solucionarla deberíamos almacenarlas hasheadas mediante algoritmos seguros como bcrypt, PBKDF2 o argon que son más modernos a diferencia de MD5 o algún SHA (familia) que son bastante vulnerables a ataques de fuerza bruta.

Otra de las que se han escogido ha sido una de "not construct SQL queries directly from user-controlled data". Dicha vulnerabilidad ha sido encontrada en el siguiente fragmento:

```
try (var connection = dataSource.getConnection()) {
      PreparedStatement statement =
          connection.prepareStatement(
```

```
            "select password from challenge_users where userid = '"
                + username_login
                + "' and password = '"
                + password_login
                + "'");
      ResultSet resultSet = statement.executeQuery();
```

Es vulnerable ya que permite que el usuario pueda construir consultas cuyo efecto no es el deseado. Podría ser capaz de acceder a recursos de la base de datos para los que no tenga autorización o ejecutar comandos para cambiar el comportamiento de la base de datos.

2. Propon una solucion e impleméntela en una nueva rama del repositorio. Explica los cambios que arreglan dicha vulnerabilidad.

Como hemos dicho en el algoritmo anterior, tendremos que usar algún algoritmo de hashing como bcrypt para almacenar el hash de la contraseña. Para ello es adecuado importar las librerías necesarias para que funcione nuestro código.

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder


@Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(null).passwordEncoder(new BCryptPasswordEncoder());
}
```

Al código anterior le hemos añadido un password encoder que emplea una función con el algoritmo bcrypt para hashear la contraseña.

Para solucionar la vulnerabilidad de inyección sql, se van a usar sentencias preparadas. A diferencia de la forma que se estaba ejecutando en el código, en la que la consulta se crea con el input del usuario y después se manda para que la ejecute la base de datos, con las sentencias preparadas lo que se hace es mandar una especie de molde a la base de datos para que valide la consulta que se va a realizar pero sin los parámetros que se van a recibir del usuario. Después, una vez se tiene el input del usuario, se manda a la base de datos para agregarlos a la consulta ya validada. Esto puede prevenir la inyección ya que el input del usuario no hace variar la construcción de la consulta. Esta es la solución que se ha implementado:

```
String query = "SELECT password FROM challenge_users WHERE userid = ? AND password = ?";
    try (var connection = dataSource.getConnection()) {
      PreparedStatement statement = connection.prepareStatement(query);
      statement.setString(1, username_login);
      statement.setString(2, password_login);
      ResultSet resultSet = statement.executeQuery(query);
```

Como se puede observar ahora se prepara la consulta, se determinan los parámetros, y tras eso se ejecuta, pero no se construye directamente con los parámetros del usuario si no que primero se valida y luego se le añaden.

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

Tras crear la pull request de la nueva rama “developer”, Sonar ha realizado el análisis correctamente y no detecta vulnerabilidades en el nuevo código

A continuación, seguimos en la “rama developer” para parchear la segunda vulnerabilidad, se hace la nueva pull request, y sonar no detecta vulnerabilidades en el nuevo código.

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

Realizando los merge de cada pull request vemos como cada análisis que realiza sonar le rama main va reduciendo el número de vulnerabilidades, primeramente 29, y después 28.