Practica 3: CTF

Metodologías de desarrollo seguro - Ingeniería de la Ciberseguridad - 2023

Carlos Barahona Pastor y Ángel del Castillo González

Índice

Whack-a-mole

Descripción

Detección

Explicación

Flag

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.

Calculadora

Descripción

Detección

Explicación

Flag

Arreglos y mejoras

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/il")).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/il")).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 coincida.*

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/il")).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 `Javacuya` generación garantiza que los números sean criptográficamente seguros.