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