Experiment 06

AIM:
To utilize website crawling OSINT tools to gather a comprehensive list of URLs, internal links, and the structure of a given website.

Theory

Website crawling is the process of systematically navigating through websites and downloading their content for purposes such as search engine indexing, data mining, or monitoring. The automated programs that perform this task are called **crawlers** or **web spiders**. They follow hyperlinks, extract content, and build a map of a site's structure.

One of the most popular frameworks for this purpose is **Scrapy**:

- **Scrapy** is an open-source Python-based framework for web crawling and scraping.
- It supports large-scale data extraction, automated crawling, and flexible spider definitions.
- It uses **spiders** (custom Python classes) to define starting URLs, crawling logic, and parsing rules.

Key concepts:

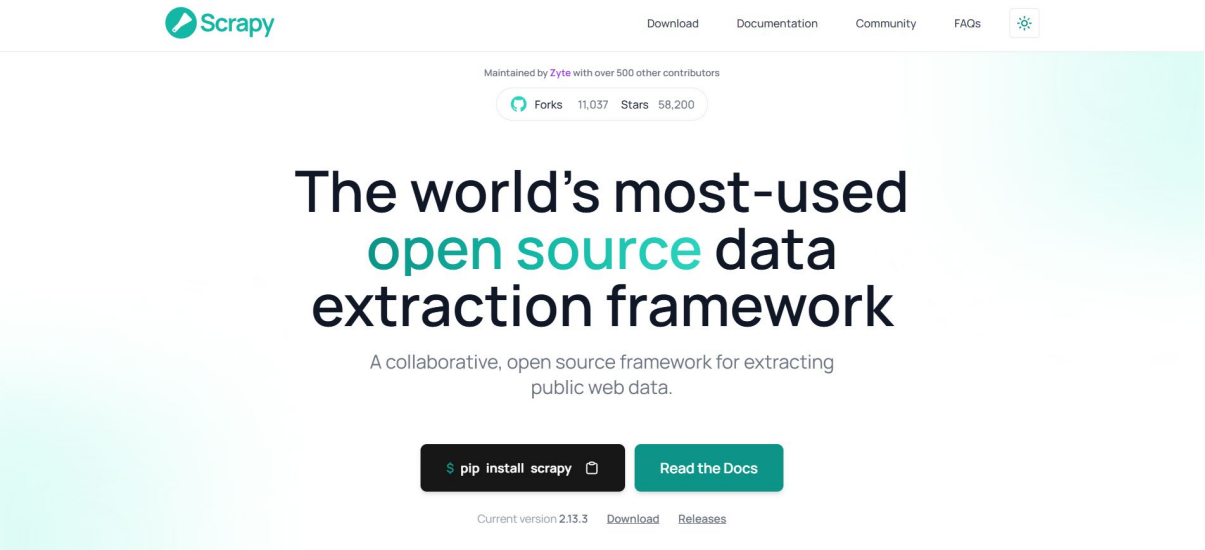
- **Spiders** → Classes that define crawling logic.
- **Start Requests** → Initial URLs from where crawling begins.
- **Selectors (XPath / CSS)** → Used to extract relevant information from HTML.
- **Responses** → Processed content received from HTTP requests.

Requirements / Tools Used

1. Python 3.x installed on the system.
2. Scrapy framework (pip install scrapy).
3. Basic understanding of HTML and CSS selectors.
4. A target website for crawling (demo/test site recommended for ethical use).

Procedure:

1. Install Scrapy



>> pip install scrapy

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> pip install scrapy
Collecting scrapy
  Using cached scrapy-2.13.3-py3-none-any.whl.metadata (4.4 kB)
Collecting cryptography>=37.0.0 (from scrapy)
  Using cached cryptography-45.0.7-cp311-abi3-win_amd64.whl.metadata (5.7 kB)
Collecting cssselect>=0.9.1 (from scrapy)
  Using cached cssselect-1.3.0-py3-none-any.whl.metadata (2.6 kB)
Collecting defusedxml>=0.7.1 (from scrapy)
  Using cached defusedxml-0.7.1-py2.py3-none-any.whl.metadata (32 kB)
Collecting itemadapter>=0.1.0 (from scrapy)
  Using cached itemadapter-0.12.2-py3-none-any.whl.metadata (22 kB)
Collecting itemloaders>=1.0.1 (from scrapy)
  Using cached itemloaders-1.3.2-py3-none-any.whl.metadata (3.9 kB)
Collecting lxml>=4.6.0 (from scrapy)
  Using cached lxml-6.0.1-cp313-cp313-win_amd64.whl.metadata (3.9 kB)
Collecting packaging (from scrapy)
  Using cached packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Collecting parsel>=1.5.0 (from scrapy)
  Using cached parsel-1.10.0-py2.py3-none-any.whl.metadata (11 kB)
Collecting protego>=0.1.15 (from scrapy)
  Using cached protego-0.5.0-py3-none-any.whl.metadata (6.4 kB)
Collecting pydispatcher>=2.0.5 (from scrapy)
  Using cached PyDispatcher-2.0.7-py3-none-any.whl.metadata (2.4 kB)
Collecting pyopenssl>=22.0.0 (from scrapy)
  Using cached pyopenssl-25.1.0-py3-none-any.whl.metadata (17 kB)
```

2. Create a Scrapy Project

```
Administrator: Windows PowerShell
PS D:\Sem_VII_Word\OSINT\Experiment06> scrapy startproject crawl_project
New Scrapy project 'crawl_project', using template directory 'C:\Users\humay\AppData\Local\Programs\Python\Python313\Lib
\site-packages\scrapy\templates\project', created in:
    D:\Sem_VII_Word\OSINT\Experiment06\crawl_project

You can start your first spider with:
    cd crawl_project
    scrapy genspider example example.com
PS D:\Sem_VII_Word\OSINT\Experiment06> cd crawl_project
PS D:\Sem_VII_Word\OSINT\Experiment06\crawl_project>
```

>> scrapy startproject crawl_project && cd crawl_project

3. **Generate a Spider**

   >> scrapy genspider sitecrawler google.com

   - Here, sitecrawler is the spider name, and example.com is the target domain.

4. **Define Spider Logic**

   In spiders/sitecrawler.py:

   - Set **start_urls** (initial URLs).
   - Override parse() method to extract links and content.
   - Use **XPath/CSS selectors** to identify elements.
   - Example snippet:

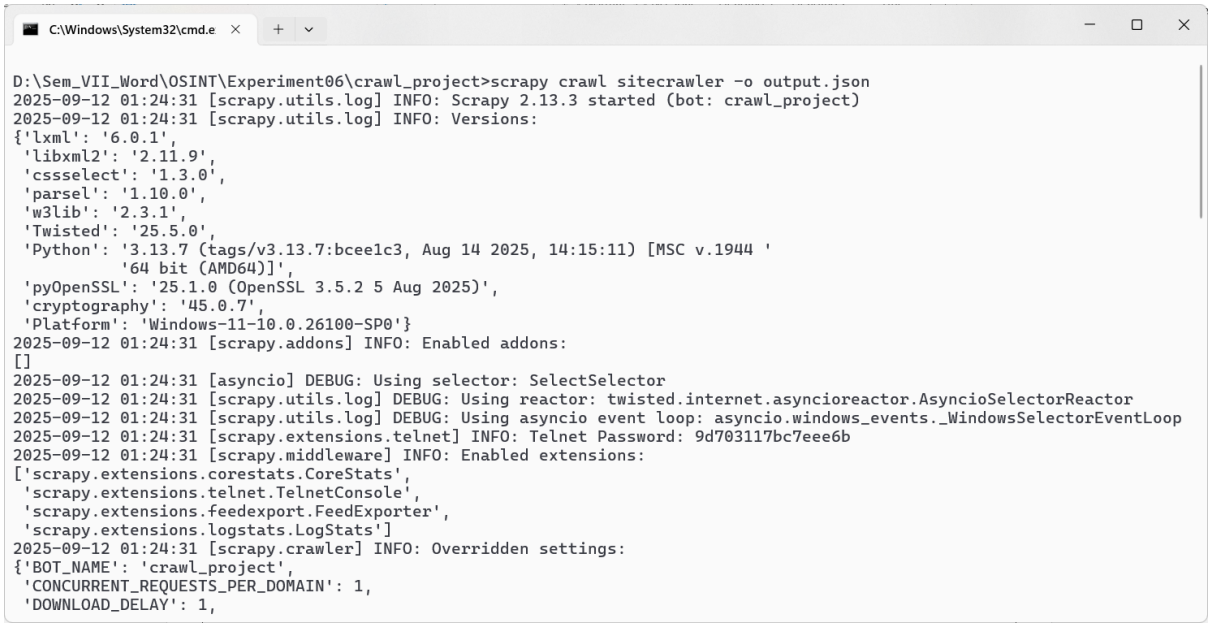
```
import scrapy
class SitecrawlerSpider(scrapy.Spider):
    name = "sitecrawler"
    start_urls = ["http://google.com"]

    def parse(self, response):
        self.log(f"Visited: {response.url}")
        # Extract all links
        links = response.css('a::attr(href)').getall()
        for link in links:
            yield {"link": link}
            yield response.follow(link, callback=self.parse)
```

5. **Run the Spider**

   >> scrapy crawl sitecrawler -o output.json

   - This saves results in **output.json** containing URLs and internal links.



6. Analyze Results

- The JSON/CSV file contains a structured list of crawled URLs.
- It can be further analyzed to identify:
  - Internal structure of the website.
  - Navigational hierarchy.
  - Suspicious or hidden links.

Observations

- Scrapy successfully crawls starting from the root page and follows internal links.
- Data can be exported in formats like JSON, CSV, or XML.
- Website structure and hidden links are clearly mapped.

Result / Conclusion

We have successfully used utilize website crawling OSINT tools to gather a comprehensive list of URLs, internal links, and structure of the website.