Research for development

Scraping the internet for entry level cyber security jobs

Khaj Thompson

Elaine John

Resources to start with

Google gemini was chosen to use for prompting

Gemini was asked “ list a few developer web sites and blogs that would provide use web scraping tools”

* Reddit,
* Stack overflow,
* github,
* jav.net

Gemini was prompted “make a list of Job recruitment web sites”

From the list the following were chosen

Ziprecruiter

cwjobs.co.uk

indeed.com

jsearch

The final selection made was: Ziprecruiter

Deliverables

Key Words

15 keywords related to entry level cyber security jobs

* Cybersecurity Analyst
* Security Operations Center (SOC) Analyst
* Network Security Administrator
* Information Security Analyst
* Penetration Tester
* Vulnerability Assessor
* Incident Responder
* Digital Forensics Analyst
* Security Engineer
* Compliance Analyst
* Cyber Threat Intelligence Analyst
* Security Architect
* Ethical Hacker
* Risk Analyst
* Security Awareness Trainer

Gemini Ai Prompt “What is a Web scraper”

**A web scraper** is a software application that automatically extracts data from websites.

It works by sending requests to web pages, analyzing the HTML code, and extracting specific information.It is a software application that automatically extracts data

We chose github as the developer page to search for an autoscraper

<https://github.com/alirezamika/autoscraper>

We searched for the .py file setup.py

Checked the readme file to learn about the selected Scraper

We made sure there is an accredited body associated with the scraper, code attached and python

We prompted gemini “ how to use auto scraper responsibly and ethically”

## **Ethical Considerations in Web Scraping**

Web scraping, while a powerful tool, can raise ethical concerns if not used responsibly. Here are some guidelines to consider:

### **Respect Website Terms of Service**

* **Read the fine print:** Most websites have terms of service that outline acceptable use. Ensure your scraping activities comply with these terms.
* **Avoid overloading:** Excessive scraping can strain a website's resources. Respect the website's bandwidth and server capacity.

### **Avoid Mimicking Human Behavior**

* **Use appropriate headers:** Set headers that indicate you're a bot, not a human user.
* **Avoid excessive requests:** Limit the frequency of your requests to avoid being flagged as a malicious bot.
* **Respect CAPTCHAs:** If a website uses CAPTCHAs, don't attempt to bypass them.

### **Protect User Privacy**

* **Handle personal data responsibly:** If you scrape personal data, handle it securely and in compliance with data privacy laws (e.g., GDPR, CCPA).
* **Obtain consent:** If necessary, obtain explicit consent from users before collecting their data.

### **Consider Website Owners' Interests**

* **Communicate:** If your scraping might significantly impact a website's operations, consider reaching out to the website owners to discuss your intentions.
* **Respect intellectual property:** Avoid scraping copyrighted content without proper authorization.

### **Technical Best Practices**

* **Use a reliable scraping tool:** Choose a tool that respects website terms of service and ethical guidelines.
* **Implement rate limiting:** Limit the number of requests per unit of time to avoid overloading websites.
* **Handle errors gracefully:** Be prepared for potential errors and implement mechanisms to handle them gracefully.

### **Additional Considerations**

* **Purpose of scraping:** Ensure your scraping activities serve a legitimate purpose and contribute to a positive outcome.
* **Impact on competition:** Avoid using scraping to gain an unfair competitive advantage.
* **Legal implications:** Be aware of local and international laws related to data scraping and intellectual property.

By following these guidelines, you can use web scraping ethically and responsibly, ensuring a positive impact on the web community.

“Regenerate as a list” prompt was added

We tested our first instance of autoscraper on yahoo

We entered the auto scraper’s basic extraction code

from autoscraper import AutoScraper

import pandas as pd

import openpyxl

import json

UrlToScrape = 'https://finance.yahoo.com/markets/'

WantedList = ["https://finance.yahoo.com/markets/stocks/most-active/"]

Scraper = AutoScraper()

data = Scraper.build(UrlToScrape, wanted\_list=WantedList)

print(data)

data\_to\_extract = {

"product\_name": ".product-name",

"price": ".product-price"

}

df = pd.DataFrame(data)

df.to\_excel("stock\_data.xlsx", index=False) # Adjust filename as needed

print("Data extracted and saved to 'stock\_data.xlsx'.")

We were able to get stock information on yahoo

We prompted Gemini to regenerate the scraper to do jobs searches

We manually entered a Job search website “Ziprecruiter”

No results were returned

We received error 403

We prompted Gemini to “explain why our scraper received error 403”

**output:**

**Error 403: Forbidden** typically indicates that the server you're trying to access has denied your request. This can happen due to several reasons:

1. **Incorrect Authentication:**
   * Ensure you're using the correct API key or credentials if required.
   * Verify that your authentication headers are set up correctly.
   * Check if the API key has expired or has usage limits.
2. **Rate Limiting:**
   * Many APIs have rate limits to prevent abuse.
   * Ensure you're not exceeding the allowed number of requests per unit of time.
   * Implement a delay or backoff mechanism to respect rate limits.
3. **IP Address Ban:**
   * If you're making too many requests or violating terms of service, your IP address might be temporarily or permanently banned.
   * Try using a different IP address or wait for the ban to be lifted.
4. **Incorrect Headers or Parameters:**
   * Check if you're sending the necessary headers or parameters in your requests.
   * Refer to the API documentation to ensure you're providing the correct values.
5. **Temporary Server Issues:**
   * The server you're trying to access might be experiencing temporary downtime or maintenance.
   * Wait a while and try again later.

At this point we learned Apis were needed for to access https

We did a prompt to search for Apis and found rapid api

On rapid api we found Api links

<https://rapidapi.com/steven-ThAtabKLVl-/api/job-titles>

<https://rapidapi.com/letscrape-6bRBa3QguO5/api/jsearch>

We chose jsearch

We use f12 to gain necessary header information on ziprecruiter

And prompted gemini to incorporate the api into our code

We spent some time running the code outputted and making adjustments

All work at this point was done using command line

Jsearch

Code

import http.client

conn = http.client.HTTPSConnection("jsearch.p.rapidapi.com")

headers = {

'x-rapidapi-key': "f016c95904mshcb5f883e445ac1ep1b864ejsn4a7ffae93426",

'x-rapidapi-host': "jsearch.p.rapidapi.com"

}

conn.request("GET", "/estimated-salary?job\_title=NodeJS%20Developer&location=New-York%2C%20NY%2C%20USA&radius=100", headers=headers)

res = conn.getresponse()

data = res.read()

print(data.decode("utf-8"))

Response

* status:"OK"
* request\_id:"3c3ca9cf-3754-435e-8b44-8dccc1a336ec"
* ▶parameters:{} 3 keys
  + job\_title:"nodejs developer"
  + location:"new-york, ny, usa"
  + radius:100
* ▶data:[] 3 items
  + ▶0:{} 9 keys
    - location:"New York, NY"
    - job\_title:"Node Js Developer"
    - publisher\_name:"Ziprecruiter"
    - publisher\_link:"https://www.ziprecruiter.com/Salaries/Node-Js-Developer-Salary--in-New-York"
    - min\_salary:36.818268
    - max\_salary:82.05241
    - median\_salary:61.613396
    - salary\_period:"HOUR"
    - salary\_currency:"USD"
  + ▶1:{} 9 keys
    - location:"New York, NY"
    - job\_title:"node js developer"
    - publisher\_name:"Talent"
    - publisher\_link:"https://www.talent.com/salary?job=node+js+developer&location=new+york"
    - min\_salary:80000
    - max\_salary:173350
    - median\_salary:130000
    - salary\_period:"YEAR"
    - salary\_currency:"USD"
  + ▶2:{} 9 keys
    - location:"New York, NY"
    - job\_title:"NodeJS Developer"
    - publisher\_name:"Salary"
    - publisher\_link:"https://www.salary.com/research/company/smartyads-inc/nodejs-developer-salary?cjid=15612621"
    - min\_salary:90377.0085995
    - max\_salary:109253.00456448
    - median\_salary:99729.63121986
    - salary\_period:"YEAR"
    - salary\_currency:"USD"

After having no luck we prompted gemini to incorporate indeed into the code replace the url we previously used

The code we ran did not work

We then learnt of using google collabs and entered out code into the virtual environment

Google collab highlighted errors and suggested fixes

We entered the fixed code into gemini and asked gemini toregenerate the code with a loop and rate limiter

Our final code was outputted with a few manual tweaks

Resulting:

import autoscraper

from autoscraper import AutoScraper

import pandas

import pandas as pd

import openpyxl

import json

import csv

import time

import signal

import traceback

import random

UrlToScrape = 'https://www.indeed.com/jobs?q=cyber+security+entry+level&l=Houston%2C+TX&from=searchOnHP&vjk=b0060ebddfc95197'

WantedList = wanted\_list = [

"Cybersecurity Analyst",

"Security Operations Center (SOC) Analyst",

"Network Security Administrator",

"Information Security Analyst",

"Penetration Tester",

"Vulnerability Assessor",

"Incident Responder",

"Digital Forensics Analyst",

"Security Engineer",

"Compliance Analyst",

"Cyber Threat Intelligence Analyst",

"Security Architect",

"Ethical Hacker",

"Risk Analyst",

"Security Awareness Trainer"

]

Scraper = AutoScraper()

data = Scraper.build(UrlToScrape, wanted\_list=WantedList)

print(data)

data\_to\_extract = {

"Job\_title": ".job-title"

}

df = pd.DataFrame(data)

import http.client

conn = http.client.HTTPSConnection("indeed-jobs-api.p.rapidapi.com")

headers = {

'x-rapidapi-key': "f016c95904mshcb5f883e445ac1ep1b864ejsn4a7ffae93426",

'x-rapidapi-host': "indeed-jobs-api.p.rapidapi.com"

}

conn.request("GET", "/indeed-us/?offset=0&keyword=cybersecurity&specialist=california", headers=headers)

res = conn.getresponse()

data = res.read()

print(data.decode("utf-8"))

df.to\_excel("job\_data.xlsx", index=False)

df.to\_csv("job\_data.csv", index=False)

# Introduce random delay between scrapes

min\_delay = 2 # Minimum delay in seconds

max\_delay = 5 # Maximum delay in seconds

delay = random.uniform(min\_delay, max\_delay)

time.sleep(delay)

print("Data extracted and saved to 'job\_data.xlsx' and 'job\_data.csv'.")

The code ran successfully however the api we were using had reached its limit

Our initial aim was to use auto scraper + other python, api and extra research to build a power scraper

Now that we had a successful run we were more aware of what to do to get a successful job scraper

We now had to build a new scraper

We chose indeed12 scraper from rapid api

https://rapidapi.com/mantiks-mantiks-default/api/indeed12

Subscription cost

\*\*Pricing Plans\*\* \* \*\*Free:\*\* $0.00/month, 25 requests/month, hard limit, 1000 requests/hour \* \*\*Pro:\*\* $6.00/month, 1,000 requests/month, hard limit, 50 requests/hour \* \*\*Ultra:\*\* $30.00/month, 10,000 requests/month, hard limit, 500 requests/hour \* \*\*Mega:\*\* $90.00/month, 50,000 requests/month, no hard limit, 1000 requests/hour + $0.0018/request

We now have additional resources google collabs, and we decided to use gemini in conjunction with chat gpt

We tested our new apis response and got code 200

Which signifies that the code works

We entered the api code snippet into chat gpt and asked it to regenerate the code using the libraries request,json and csv to extract job data

We ran the code and got 0 data

Collabs suggested taking a look at our parameters

We manually replaced the job parameter to entry level cyber security and location to new york

Our final code out that worked :

import requests

import json

import csv

# API URL and parameters

url = "https://indeed12.p.rapidapi.com/jobs/search"

querystring = {"query": "entrylevelcybersecurity", "location": "newyork", "page\_id": "1"}

headers = {

"x-rapidapi-key": "f016c95904mshcb5f883e445ac1ep1b864ejsn4a7ffae93426",

"x-rapidapi-host": "indeed12.p.rapidapi.com"

}

# API request

try:

response = requests.get(url, headers=headers, params=querystring)

response.raise\_for\_status() # Raises an HTTPError for bad responses

# Check if the response contains data

if response.status\_code == 200:

data = response.json()

# Debug: Print the raw data response

print("Raw API response:", json.dumps(data, indent=4))

# Check if 'hits' key exists and has data

if 'hits' in data and len(data['hits']) > 0:

# Specify the CSV file name

csv\_file = 'cyberjobs\_data.csv'

# Open CSV file for writing

with open(csv\_file, mode='w', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

# Write the header row

writer.writerow(["company\_name", "formatted\_relative\_time", "id", "link", "locality", "location", "pub\_date\_ts\_milli", "salary"])

# Write data rows

for job in data.get('hits', []): # Use .get() to avoid KeyError

salary\_info = job.get('salary', {})

writer.writerow([

job.get('company\_name', ''),

job.get('formatted\_relative\_time', ''),

job.get('id', ''),

job.get('link', ''),

job.get('locality', ''),

job.get('location', ''),

job.get('pub\_date\_ts\_milli', ''),

f"{salary\_info.get('min', '')}-{salary\_info.get('max', '')} {salary\_info.get('type', '')}"

])

print(f"Data successfully written to {csv\_file}")

else:

print("Error: No job listings found for the given criteria.")

else:

print(f"Error: API request failed with status code {response.status\_code}")

except requests.RequestException as e:

print(f"Error: Unable to fetch data ({e})")

We prompted gemini to create information for a readme file using this code

We copied the output to chat gpt and prompted chat gpt to improve the read me and export it as a readme file

We then entered out code into chat gpt and prompted it to create a powerpoint presentation on our job scraper code