Undetectable_web_scraper documentation

Table des matières

I Short Description

The following project is a scraping framework designed to extract data from any website requiring user profiles while evading detection by anti-bot systems. This is achieved by mimicking human behavior and utilizing the undetected_chromedriver

library. However, it has not been proven to bypass Cloudflare's highest security measures and has not been tested since September 2023. Additionally, it requires optimization and further development to be suitable for industrial-scale use.

undetected_chromedriver

CronJobs

II Installation

The main library used for driver's stillness; undected-chromedriver is built on selenium.

Its author indicates that driver's window shouldn't be interacted with manually otherwise it may turn driver's stillness ineffective.

The program structure was initially designed to run autonomously on a server.

The thought strategy was to use to program run of the main script.

It has finally only been implemented locally because of time shortage.

Indeed undected-chromedriver uses as default fingerprint of the host machine making the use of a server difficult to mask. Also, the library is declared as unstable in headless meaning that a gpu emulator or other similar solution might be requested if the server doesn’t have one.

Due to driver’s configuration modifications on newest versions related to chrome profiles path selection (check out driv3r.py, line 27); selenium 4.13.0 is the version employed to run undetectedchromedriver without complication. This point is one of those to review to improve program stillness.

Notice that undetected-chromedriver installation automatically installs selenium based on the provided python version.

Furthermore, in intent to run the program on a Debian 12 “bookworm” server, the used python version to build this project is python 3.9.2.

To begin working with this program, creation of a virtual env with help of the requirement file is therefore recommended.

III General description of the structure and main functions

To read this chapter may I suggest you to split your screen in two parts: one with the program file system in your favourite code editor or GitHub and the other with this file.

A Modules synthesis

main1.py
Call other modules

driv3r.py
Functions related to driver.

m0ng0.py
Functions related to MongoDB Atlas the chosen cloud solution.

planning.py
Functions related to program’s activity scheduling.

quickstart.py

Functions related to gmail api.

The API is called in this program for 3 use cases:

. as a remote system to stop/run the program

.to send the scraped data as a file to referred mail addresses.

.to report bugs (this last point still waits to be edited, to make the program efficient in the long run)

This module is contained in gmai1 file along with log_files and __init__.py. log_files

is meant to contain json files allowing to interact with the API.

To implement it please follow the instructions on https://thepythoncode.com/article/use-gmail-apiin-python.

Now that we ended our module's introduction let's dive in the script calling them all to fully grasp their interactions:

B Backbone

To ease your reading; described blocks of code are referenced by the number of the first line of code in the corresponding module. The code is available in the Scrap file of the repository https://github.com/t0r3l/undetectable_web_scraper/blob/main/Scrap/main1.py.

main1.py, Line 1

First program action is to select the right file directory to allow importation of local modules, then to import those between others.

Line 15

First path to credentials must be set in.

Then scripts connects our application to gmail API thanks to quickstart.py.

Query will allow refer mail addresses to interact with the one of the program.

Destination is the destinated mail to receive status switches of the program.

Subject is the subject of send mails to destination mail.

Line 23

Next the script will call robotUnion() a planning.py locale module function which aims to schedule bot’s activity on random hours from 7:30 AM to 21:59 PM with a break between 12:30 and 13:59 on the HHMMSS format.

The purpose of this function being to fool a potential analysis on our activity.

Line 28

The next action is to create a session thanks to Driver() function from driv3r.py locale module. Driver()

configs the session calling a chrome profile.

Once the path and profile name specified, the function creates or not a corresponding profile depending on its existence.

When used for the first time; it is necessary to log in (manually with a regular chrome session) to the wanted website to scrap and to fill “remember me” option if exists.

Once done cookies will be allocated to the profile and requesting the page to scrap will no longer require passing by a log page if corresponding profile’s path is passed to Driver()

Otherwise, repetition of this action might look suspicious in addition of being time costing in the long run.

Line 51

Then once columns of the data set has been declared; the program calls connect2collection() a m0ng0.py function. This function allows connection with MongoDB Atlas after providing a connection token, along with DB and collection name.

Line 53

Depending on what has been previously scraped and exported to the defined collection, the program follows a decision tree to know if it must either resume or begin the data retrieval.

A Collection is empty

If collection is empty indexes of page and vignettes to scrap will consequently be 1.

Driver.get(‘url to scrap’) allows as in selenium to reach wanted web page sleep(np.random.uniform(10,20)) page to fully charge and to have a random time of reaction to

behave the most human as possible.

Get_window_and_vignettes () (see driver.py line 291) is a driver.py function returning

Window object to scroll in and vignettes in this order in a list object.

The strategy being to count number of length vignettes for each page to scroll the wright way while scraping information.

Current scroll poll position is determined thanks Javascript and previously determined window object.

Collection empty means that we are dealing with first session of our campaign, last_capchat_occurence_since_beginning is therefore set to 0 by default.

This variable has for purpose to determine stillness of our application and to compare price if capchat solving by a tier service is applied (see driver.py capchat_recording() Line 176 and scrap() Line 195).

Line 66

B Collection is not empty.

If collection is not empty last iterators values, last_capchat_occurrence_since_begening and tot_vignettes_in_page (total number of vignettes in current page), are retrieved from collection thanks to last_value() function (from m0ng0.py Line 35).

Line 73

B1 Then if last_i (last vignette index) equals to tot_vignettes in page, collection is not empty and last retrieved vignette is the last one of a page => program must go to the next page.

To do so driver will request the next seen page, then scroll down to the bottom of it to target the “next button” of page thanks to go2nextpage function (from driver.py Line 39).

This approach allows to avoid direct connection with an unseen web page which may look suspicious.

Line 90

B2 Collection is not empty and last retrieved vignette is not the las one of a page => program goes back where he left its scrapping thanks to scroll() function (from driver.py Line 113).

Line 105

L is the list of our database variables. It will be used later in IV Scraping a page.

Line 109

Then it enters an infinite loop implemented to run until defined last hour will occur.

It runs a for loop which will iterate while vignettes have not all been treated.

Line 112

Then we enter a decision tree. First it will check if the current time is or isn't included in its schedule of activity previously defined by robotUnion().

Line 114

If current time fits in, therefore the program searches for unread messages from designated mail addresses in Gmail's API connected to the bot accompt.

If the last unread message contains the pattern "sleep" (case insensitive), the program will mark the message as read, send "Asleep" to the designated mail address then sleep until receiving the pattern "run" and treat it in an analogical way however allowing the program to execute its next instructions.

Therefore, the program can start scraping untreated vignettes of a page. This part will be explained in the following section "Scraping a page".

Line 200

If the latest time of the schedule is not exceeded while the current time is out of the schedule it will sleep until fitting in.

Line 202

Else if current time exceeds schedule's last hour; the infinite loop will break once the driver quitted and scraped data exported to MongoDB Atlas.

The code in comment allows to send a mail containing 500 extracted vignettes as an excel file to selected mails each time that variable sent equals to False is superior or equal to 500.

It Then updates this variable to True in the DB.

IV Scraping a page

A initiating in main1.py

Line 123

Declaring variables to scrap through their XPATH values indexed by an iterator.

Line 127

Height of the current vignette to treat is assigned to VignetteSize variable.

This height corresponds to height in pixels of a single vignette.

Line 128

A If a previous session did not finish to scrap the entire page, lastVignetteHeight will differ from the initialized value 0 (see B case Line 66).

Consequently, sum of lastVignetteHeight and VignetteSize will be append to VignetteHeight.

VignetteHeight being the list keeping score of total height of a vignette relatively to start of vignettes

Then lastVignetteHeight will then be set to 0 as scrolling has been reset.

Line 132

B Else if VignetteHeight is not an empty list and if iterator differs from 1 meaning that we are not dealing with the first retrieval of a session and page;

Sum of previous VignetteHeight and current VignetteSize is append to current VignetteHeight.

Line 135

C Otherwise, it is the first vignette of a session therefore only VignetteSize is append to VignetteHeight.

Line 138

Once the variable set, X is declared.

It is a list containing variables to scrap XPATH in the same order that the one of the lists containing their corresponding scraped values between others in "L" (main1.py, line 105).

Line 142

Then a for loop begins, it iterates through L for n values to scrap and X.

B Scrap()

First action will be to call scrap(); let's dive in this locale function.

Parameters

driv3r.py, line 195, scrap()

This function from drv3r.py locale module takes in argument.

1 driver

Driver session in our case Driver() .

2 elementList

List in which to append scraped data of a variable.

3 l_control

Loop iterator aiming to mark a variable to treat them depending on their range.

4 scrap_var_i

List containing variables scraped by the program at i iteration. Its purpose is to know the pattern to strip from a following scraped value(i plus position) marked by l_control if this value contains scraped_var_i last content and the wanted content to retrieve.

Eg:

scrap_var_i = “Alan Turing”

scrap_var_i_and_i_plus_HTML = “AllanTuring   1912-1954”

scrap_var_iplus = scrap_var_i_and_i_plus_HTML.text.lstrip(scrap_var_i)

⇨ Scrap_var_iplus value is “ 1912-1954”

5 elementXPATH

Is the XPATH of a searched value.

6 page

Index of current scraped page to check out if current url corresponds to what we are reaching for in case of capchat occurrence which might be detected through an url change.

7 capchat_witness

List containing number of capchats which occurred since beginning of a scraping campaign.

Useful data to keep track of its pricing in case of using a capchat solving tier priced to unit of solving or merely to improve the program.

8 vignettes_window

Is the current sub window containing vignettes to scrap within a page.

9 current_scroll_position

Is current position of the cursor which might slightly differ from vignetteHeight due to scroll() function which will soon be explained.

10 last_capchat_occurence_since_begening

Variable containing last capchat occurrence since beginning of campaign.

11 vignettes_window_XPATH

Is the XPATH of vignettes_window

12 vignettes_selector

Is the HTML code containing all vignettes containers.

13 VignetteHeight

List of each relative vignette height value in pixels from top of vignettes_window.

14 VignetteSize

Absolute height of current vignette to scrap in pixels.

15 refreshing

It is the counter of necessary refreshes while scraping a same variable.

It is an optional parameter equals to 0 by default.

Now that our parameters have been introduced, let’s dive in the function.

Body

driv3r.py, line 195

Main structure of scrap() is a try decision tree based on selenium.common.exceptions.

Line 197

A Has 600 seconds to detect Xpath if fails exception occurs otherwise it vignette's variables can begin to be retrieved.

Line 204

A1 Normal retrieving:

Line 206

A2 retrieving iplus variable:

If scraping of a variable encounters no exception; it will check l_control in case of iterator being the one set to be truncated (see scra_var_i variable description) above, if not it scraps current variable as usual and increment l_control by 1.

Line 214

B If an exception occurs 3 scenarios have been written:

Line 215

B1 Capchats are detected based on a different url retrieved than the one expected.

If this happens last_capchat_occurence will be incremented by 1.

Then capchat _recording() will be called.

This function declared in the same module (driv3r.py) records url, sreenshot, and source code of a capchat in a file system of capchat_control file. Those records have for purpose to adapt the program to any capchat without having to monitor it in case that the capchat is lost if not solved in time.

Once done a generic structure to handle capchats has been written in comments.

Line 229

B2 Specific values may miss while others never depending on the scraped website.

To handle it those variables are marked with l_control and assigned to 'Na' value if not found at time.

Line 233

B3 If neither a capchat or a casual variable has been detected, page will be refreshed, vignette_window and vignette selector will be reset then scrap will be retrieved, scrolling to the current vignette will be done once more then scrap() will be called by recursively this time with the refreshing parameter set to 1 in for limiting refreshing solution to be over used.

Line 254
B4 Otherwise, the program has failed; it must be debugged.

Line 261
scrap() returns l_control, last_capchat_witness and reset_v2(containing vignette_window and vignette_selector) in a list.

main1.py, line 143 scrap() output will then be assigned to Scrap variable in main script.

All technical variables will then be appended to their corresponding list

Line 165
Then as long as the for loop didn't scrape all vignettes of a page; program will enter a try decision tree.

If no exception occurs scroll() locale function will be called, if StaleElementException occures scroll() will be called again because this might be only due to a time shortage for the request to work properly.

C Scroll():

Let's dive in scroll() function.

Parameters

driv3r.py, line 113

1 driver
driver session in our case Driver()

2 vignettes_window
window to scroll to reach next vignette to scrap

3 current_scroll_position
Position in pixels of the scroll

3 VignetteHeight
List of each relative vignette height value in pixels from top of vignettes_window.

4 VignetteSize
Absolute height of current vignette to scrap in pixels.

5 last_scroll_position
An optional argument which is the scroll position of a previous session if exists.

Body
Line 116
up_or_down:

This variable is meant to define deviation of scrolling from totalVignetteHeight to make scrolling appear more human. Chosen interval is [0,1] pixels of deviation however it might be changed as needed.

Scroll_position_to_compare and deviation:

They both have for purpose to compute deviation after a previous scroll to prevent outranging from a selected interval.

Line 121
The function will then enter a first decision tree to decide of the value which will be incremented to current_scroll_position to reach next vignette to scrap.

Decision will be taken depending on last_scroll_position's value : If

0, it will be a casual scrolling from a vignette to the following one

See main1.py at line 168 or 171.

Otherwise scrolling will have to be performed from position 0 of a page to a vignette anywhere within; see main1.py line 104 or driv3r.py line 249 in refreshing part of scrap() function.

Line 136

Once the scroll_distance is selected, the number_of_scroll is randomly assigned a value of either 1 or 2 to determine whether scrolling will be split with short breaks for simulating human behavior.

Line 137

Case number one:

number_of_scroll equals to 1; consequently, driver will scroll from current distance to scroll_distance

by incrementing it's position by 1 pixel with random pauses between each incrementation.

Finally it will scroll to match float divergence.

Line 147 Case number 2:

number_of_scrolls equals to 2; implying that scrolling must be acted in two parts.

To perform this task, the program will in a first hand randomly chose a percentage that will allow to split scroll_distance in two parts.

It will then perform same actions than in first case for each split parts of distance to scroll but with a longer break between both for a more human appearance.

Line 176

Finally scroll() returns current_scroll_position.

D End of the cycle

main1.py, line 168

Once current_scroll_position is assigned, and iterator incremented this last one

is checked line 175 as mentioned; if it is strictly inferior to totality of vignettes to scrap then the loop continues.

Otherwise data will be exported to MongoDB Atlas thanks to export2atlas() local function from m0g0.py.

Lists will then be reset, page will be turned thanks to go2nextpage() from driv3r.py, iterator and page will be actualised as well as vignettes window and vignettes thanks to get_window_and_vignettes() from driver and finally current_scroll_position.