How to Avoid Bot Detection With Selenium

Yuvraj Chandra
Updated: September 10, 2024 · 9 min read

Table of contents

How Do Anti-Bots Work?
How Do Websites Detect Selenium?
Top Methods to Avoid Bot Detection With Selenium
1. IP Rotation / Proxy
2. Disabling the Automation Indicator WebDriver Flags
3. Rotating HTTP Header Information and User Agent
4. Avoid Patterns With a Selenium Bot
5. Remove JavaScript Signature
6. Using Cookies

Is it still possible to avoid bot detection with Selenium? The short answer is yes, but...

It requires some clever techniques and an understanding of how anti-bot systems actually work.

As websites improve their bot detection methods, avoiding blocked requests and CAPTCHAs has become increasingly challenging. Here're the nine most important methods to bypass Selenium detection in this guide:

- Rotate IPs / Use proxies.
- Disable the automation indicator WebDriver Flags.
- Rotate HTTP Header information and User Agent.
- Avoid patterns with a Selenium bot.
- Remove JavaScript signature.
- Use cookies.
- Follow the page flow.
- Use a browser extension.
- Use Selenium Stealth Plugin.

How Do Anti-Bots Work?

Bot detection is the process of scanning and filtering the network traffic to detect and block malicious bots. Web Application Firewalls (WAFs) and anti-bot providers like Cloudflare, PerimeterX, and Akamai work tirelessly to find ways to detect bots using a headless browser, header data, and different behavioral patterns.

When a client requests a web page, information about the nature of the request and the requester is sent to the server to be processed. Active and passive detection are the two main methods an anti-bot uses to detect bot activities. Check out our article on bot detection to learn more.

How Do Websites Detect Selenium?

Selenium is among the popular tools in the field of web scraping. As a result, websites with strict anti-bot policies try to identify its unique attributes before blocking access to their resources.

So, how is Selenium detected? Selenium bot detection mainly works by **testing for specified JavaScript variables that emerge while executing Selenium**. Bot detectors often check for the words "Selenium" or "WebDriver" in any of the variables (on the window object), as well as document variables named `$cdc_` and `$wdc_`.

They also check for the values of automation indicator flags in the WebDriver, like `useAutomationExtension` and `navigator.webdriver`. These attributes are enabled by default to allow a better testing experience and as a security feature.

Additionally, advanced Selenium detection methods may employ browser fingerprinting to identify characteristics unique to automated browsers. Selenium can also get blocked websites when they analyze user behavior patterns to spot inhuman speed or consistency in interactions, which can indicate bot activity.

Top Methods to Avoid Bot Detection With Selenium

Bot detection nowadays has become a headache for web scrapers, but there are ways to get around it. Here are some of the techniques you can use to avoid bot detection in Selenium using Python:

1. IP Rotation / Proxy

One of the major ways most bot detectors work is by inspecting IP behaviors. Web servers can draw a pattern from an IP address by maintaining a log for every request.

They use Web Application Firewalls (WAFs) to track and block IP address activities and blacklist suspicious IPs. The repetitive and programmatic request to the server might hurt the IP reputation and result in getting blocked permanently.

To avoid bot detection, you can use IP rotation or proxies with Selenium. This could be considered as one of the easiest Selenium antidetect approaches where proxies act as an intermediary between the requester and the server.

The responding server interprets the request as coming from the proxy server, not the client's computer. As a result, it won't be able to draw a pattern for behavioral analysis.

scraper.py

```python
# pip3 install selenium
from selenium import webdriver

# define the proxy server
PROXY = "<PROXY_IP_ADDRESS>:<PROXY_PORT>"

# set ChromeOptions()
options = webdriver.ChromeOptions()

# add the proxy as argument
options.add_argument(f"--proxy-server={PROXY}")
driver = webdriver.Chrome(options=options)

# send the request
driver.get("https://httpbin.io/ip")

# close the driver
driver.close()
```

For best results, it's highly recommended to use premium proxies. Premium proxies offer more stable connections and higher uptime, provide faster speeds, and are less likely to be flagged or blacklisted by websites.

ZenRows is an excellent example of a premium proxy service. It offers residential proxies, which appear as real user devices to websites, making them harder to detect. It also provides an auto-rotator feature that automatically switches between different IP addresses. This constant rotation significantly reduces the likelihood of pattern detection and IP bans.

You can learn more about using proxies with Selenium in our guide.

2. Disabling the Automation Indicator WebDriver Flags

While web scraping with Selenium, the WebDriver sends information to the server to indicate the request is automated.

The WebDriver is expected to have properties like `window.navigator.webdriver`, mandated by the W3C WebDriver Specification to allow better testability and as a security feature. This results in getting detected by the web servers, which leads to being flagged or denied access.

With the availability of `execute_cdp_cmd(cmd, cmd_args)` commands, **you can now easily execute Google-Chrome-DevTools commands using Selenium**. That makes it possible to change the default flagships.

scraper.py

```python
from selenium import webdriver

# create Chromeoptions instance
options = webdriver.ChromeOptions()

# adding argument to disable the AutomationControlled flag
options.add_argument("--disable-blink-features=AutomationControlled")

# exclude the collection of enable-automation switches
options.add_experimental_option("excludeSwitches", ["enable-automation"])

# turn-off userAutomationExtension
options.add_experimental_option("useAutomationExtension", False)

# setting the driver path and requesting a page
driver = webdriver.Chrome(options=options)

# changing the property of the navigator value for webdriver to undefined
driver.execute_script("Object.defineProperty(navigator, 'webdriver', {get: () => undefined})")

driver.get("https://www.google.com")

# close the driver
driver.close()
```

3. Rotating HTTP Header Information and User Agent

The HTTP header contains information about the browser, the operating system, the request type, the user language, the referrer, the device type, and so on.

Example

```
{
    "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
exchange;v=b3;q=0.9",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "en-US,en;q=0.9",
    "Host": "httpbin.org",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/108.0.0.0 Safari/537.36",
}
```

The values of some of these attributes are different for headless browsers by default, and **anti-bots identify these discrepancies to distinguish between legitimate visitors and bots**. To mitigate detection, rotating user agents in Selenium can be helpful.

Here's how we used rotating HTTP header information to avoid bot detection with Selenium:

scraper.py

```python
from selenium import webdriver

driver = webdriver.Chrome()

# initializing a list with two User Agents
useragentarray = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36",
]

for i in range(len(useragentarray)):
    # setting User Agent iteratively as Chrome 108 and 107
    driver.execute_cdp_cmd(
        "Network.setUserAgentOverride", {"userAgent": useragentarray[i]}
    )
    print(driver.execute_script("return navigator.userAgent;"))
    driver.get("https://httpbin.io/headers")

driver.close()
```

Manually maintaining and updating a list of User Agents can be tiresome and costly. Even with significant effort, a manually compiled list may still not be diverse or up-to-date enough to avoid detection.

Web scraping APIs, such as ZenRows, offer auto-rotation of User Agents, which can be a valuable feature for your scraping projects. It continuously updates its User Agent lists, ensuring that your requests always use current and diverse User Agents. This approach significantly reduces the chances of detection and saves you time and resources in the long run.

To learn more about rotating User Agents in Selenium, check out our detailed guide on the topic.

4. Avoid Patterns With a Selenium Bot

One of the major mistakes that automation testers make is to create a bot with a defined time frame. Humans don't have a solid consistency like a bot, so **it becomes fairly easy for the anti-bots to identify the consistent patterns of the bots**.

It's also a common mistake to rapidly navigate from page to page, which humans don't do.

Randomizing time frames, using waits, scrolling slower, and generally trying to mimic human behavior surely elevate the chance of avoiding bot detectors. Here's how to bypass Selenium detection by avoiding patterns:

scraper.py

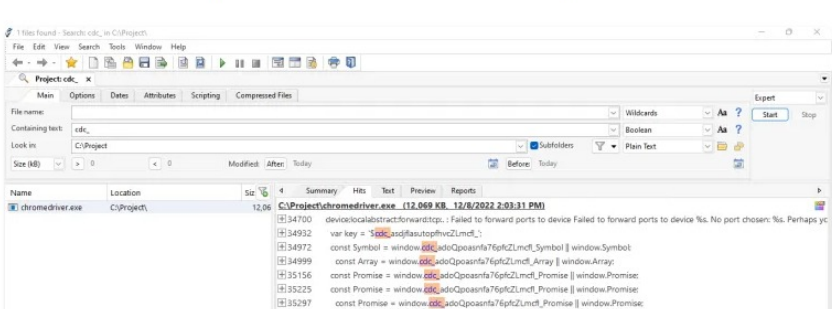
```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

driver = webdriver.Chrome()
driver.get("https://scrapingcourse.com/ecommerce/")

# wait 3.5 seconds on the web page before trying anything
time.sleep(3.5)

# Wait for 3 seconds until finding the element
wait = WebDriverWait(driver, 3)
element = wait.until(
    EC.presence_of_element_located(
        (By.CSS_SELECTOR, ".woocommerce-loop-product__title")
    )
)
print("Product name: " + element.text)

# wait for 4.5 seconds before scrolling down 700px
time.sleep(4.5)
driver.execute_script("window.scrollTo(0, 700)")

# wait for 2 seconds before clicking a link
time.sleep(2)
wait = WebDriverWait(driver, 3)
element = wait.until(
    EC.presence_of_element_located(
        (By.CSS_SELECTOR, "a.woocommerce-loop-product__link")
    )
).click()

# wait for 5 seconds until finding the element
wait = WebDriverWait(driver, 5)
element = wait.until(
    EC.presence_of_element_located(
        (By.CLASS_NAME, "woocommerce-product-details__short-description")
    )
)
print("Description: " + element.text)

# close the driver after 3 seconds
time.sleep(3)
driver.close()
```

5. Remove JavaScript Signature

One of the ways bot detectors like FingerprintJS and Imperva work is by inspecting the JavaScript signature inside WebDrivers, like ChromeDriver and GeckoDriver.

This signature is stored in the `cdc_` variable. Websites look for the `cdc_` variable in the document before denying access.

We'll use a tool called Agent Ransack to search for this signature in the chromedriver.exe binary file. It works the same way for WebDrivers like GeckoDriver and EdgeDriver.



Click to open the image in full screen

As you can see, the signature is `$cdc_asdjflasutopfhvcZLmcfl_`. In order to evade detection, we can change "cdc" to a string of the same length as "abc". First, we need to open the binary of ChromeDriver. We'll use the Vim editor to open and edit the binary file.

You can download Vim here. Click on "standard self-installing executable" for Windows. The program comes pre-installed by default for Mac and Linux.

MS-Windows: Recent and signed MS-Windows files are available on the vim-win32-installer site
The current stable version is gvim_9.0.0000.exe
An alternative is the standard self-installing executable, currently also version 9.0.
Unix: See the GitHub page, or Mercurial, if you prefer that. There is also an Appimage which is build daily and runs on many Linux systems.
Mac: See the MacVim project for a GUI version and Homebrew for a terminal version

Click to open the image in full screen

After installation, open CMD and type this and navigate to the folder. Then, run the following command:

Terminal

```
vim.exe <pathTo>\chromedriver.exe
```

Then type `:%s/cdc_/abc_/g` to search and replace `cdc_` with `abc_`.

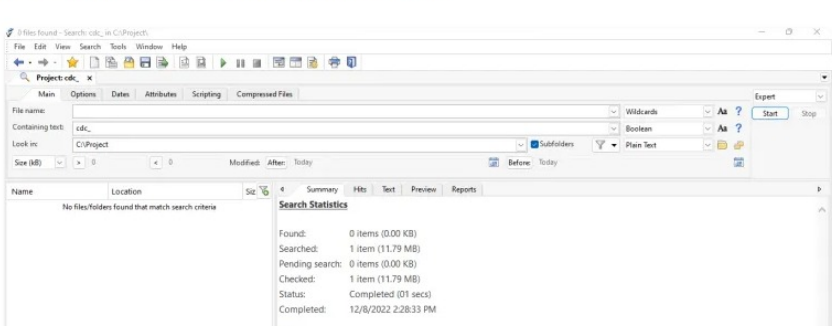
Click to open the image in full screen

Next, to exit Vim, type `:wq` and hit Enter to save the changes.

Some files might be generated with `~` at the end of the file names. Delete these files.

Click to open the image in full screen

Now, let's try the same search using Agent Ransack.



Click to open the image in full screen

As you can see now, the `cdc_` signature variable isn't found in the file.

6. Using Cookies

When trying to scrape data from social media platforms or other sites that require some form of authentication, it's very common to log in repeatedly.

This iterative authentication request raises the alarm, and the account might be blocked or face a CAPTCHA or JavaScript challenge for verification.

In order to avoid this, we can use cookies. After logging in once, we can collect the login session cookies to reuse them in the future.

7. Follow the Page Flow

When interacting with a website, it's important to follow the same flow that a human user would.

That means clicking on links, filling out forms, and navigating the website naturally.

Following the page flow can make it less obvious that you're performing automation.

8. Using a Browser Extension

Another way to bypass Selenium detection is by using a browser extension, like uBlock Origin, to block JavaScript challenges and CAPTCHAs from being loaded on the page. That can help reduce the chances of your bot being detected by these challenges.

uBlock Origin is a free, open-source browser extension designed to block unwanted content (such as ads, tracking scripts and malware) from being loaded on web pages.

It can also be configured to block JavaScript challenges and CAPTCHAs, which can help reduce the chances of your bot being detected by these challenges.

To use uBlock Origin to avoid Selenium bot detection, you'll need to install the extension in your browser and configure it to block JavaScript challenges and CAPTCHAs.

You can then use Selenium to interact with the browser as you normally would, and uBlock Origin will automatically block any unwanted content from being loaded on the page.

It's important to note that uBlock Origin may not work with all websites, and it may not be able to block all types of JavaScript challenges and CAPTCHAs.

However, it can be a useful tool for reducing the chances of your bot being detected by these challenges, especially when combined with other methods.

9. Use Selenium Stealth Plugin

The Selenium Stealth plugin is a powerful tool designed to help your Selenium-based scrapers avoid detection. It modifies Selenium's default configuration to mimic real browser fingerprints, making it harder for websites to identify your scraper as a bot.

Selenium Stealth works by implementing several key changes:

- It sets the WebDriver navigator property to false, hiding one of the most common indicators of automated browsing.
- In headless mode, it replaces the `HeadlessChrome` User Agent with an actual Chrome User Agent, making your requests appear more like those from a real browser.
- It adjusts various browser properties and behaviors to more closely resemble those of a typical user's browser.

These modifications can be particularly helpful when dealing with websites that employ basic to moderate anti-bot measures. By masking the telltale signs of automation, Selenium Stealth allows your scraper to fly under the radar in many scenarios where base Selenium would be detected and blocked.

However, it's important to note that Selenium Stealth isn't a perfect solution and has some limitations, including:

- It only partially patches Selenium, meaning some bot-like attributes would still be detectable by more sophisticated anti-bot systems.
- The plugin would struggle against advanced detection techniques used by complex security measures, such as those employed by Cloudflare.
- As anti-bot technologies evolve, the effectiveness of Selenium Stealth will decrease over time if it's not regularly updated.

Despite these limitations, Selenium Stealth remains a valuable tool in a web scraper's arsenal, particularly for sites with less stringent anti-bot measures. When combined with other techniques like proxy rotation and mimicking human behavior, it can significantly improve your scraper's ability to avoid detection.

To learn more about implementing Selenium Stealth and maximizing its effectiveness, check out our detailed tutorial on using Selenium Stealth in Python.

Alternatives to Base Selenium

While Selenium is a powerful tool for web scraping, it has its limitations. Selenium's detectable automation properties often lead to blocking by sophisticated anti-bot systems. Also, setting up and maintaining Selenium scripts can be time-consuming, especially for complex websites or large-scale projects.

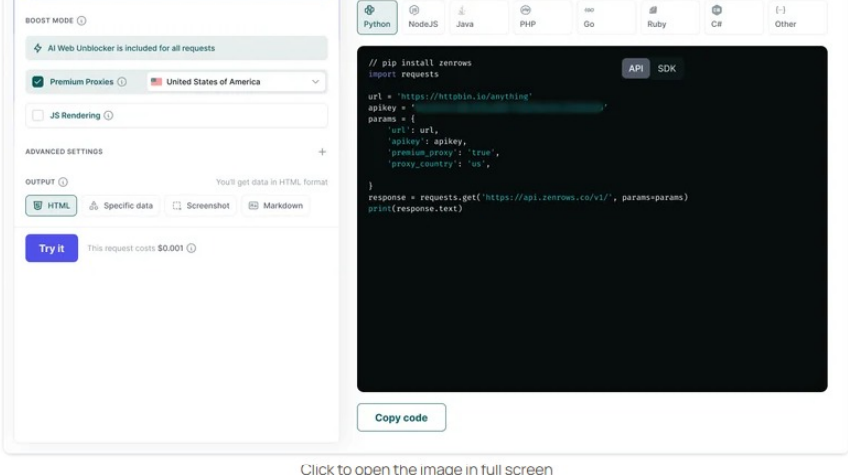
In this section, we'll explore some of the best options available, each offering unique features and advantages for different web scraping scenarios.

ZenRows Web Scraping API

ZenRows web scraping API is designed to overcome the challenges of modern web scraping. It offers a comprehensive solution that can effectively replace Selenium in many scenarios. It excels at bypassing CAPTCHAs and anti-bot measures, boasting a near 98.7% success rate in accessing protected web pages.

Unlike Selenium, ZenRows eliminates the need for complex configurations. To demonstrate its capabilities, let's use it to scrape a heavily protected site: G2 Reviews.

To get started, sign up for ZenRows. After logging in, you'll be redirected to the Request Builder page. Enter the target URL in the link box, activate Premium Proxies, and enable JS Rendering boost mode. Select your programming language (e.g., Python) and choose the API connection mode.



Click to open the image in full screen

The Request Builder will generate Python code similar to this:

scraper.py

```python
# pip install requests
import requests

url = "https://www.g2.com/products/asana/reviews"
apikey = "<YOUR_ZENROWS_API_KEY>"
params = {
    "url": url,
    "apikey": apikey,
    "js_render": "true",
    "premium_proxy": "true",
}
response = requests.get("https://api.zenrows.com/v1/", params=params)
print(response.text)
```

After running this code, you'll receive the full HTML content of the target page:

Output

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="https://www.g2.com/images/favicon.ico" rel="shortcut icon" type="image/x-icon" />
    <title>Asana Reviews, Pros + Cons, and Top Rated Features</title>
</head>
<body>
    <!-- other content omitted for brevity -->
</body>
</html>
```

The ZenRows web scraping API's ability to handle JavaScript rendering, bypass CAPTCHAs and anti-bot measures, automate proxy management, and more makes it a powerful alternative for your web scraping needs.

Undetected ChromeDriver

Undetected ChromeDriver is an open-source project that makes Selenium ChromeDriver "look" human. It's an optimized Selenium ChromeDriver patch that lets you bypass anti-bot services like DataDome, Distill Network, Imperva, and Botprotect.io.

One of its key strengths is handling JavaScript challenges, which are commonly used to mitigate cyberattacks like DDOS and prevent scraping. These challenges involve sending JavaScript code to the browser, which legitimate browsers can understand and pass due to their JavaScript stack. It's compatible with Brave browser and other Chromium-based browsers, expanding its versatility.

Undetected ChromeDriver is more capable than base Selenium in bypassing some bot detection measures. But, it can still be detected and blocked by sophisticated anti-bot systems, especially those using advanced browser fingerprinting or machine learning techniques.

SeleniumBase

SeleniumBase, particularly its Undetected ChromeDriver (UC) Mode, is an advanced web automation framework that builds upon Selenium to provide enhanced capabilities for bypassing anti-bot detection systems. UC Mode allows bots to appear more human-like. This enables them to evade detection from anti-bot services that might otherwise block them or trigger CAPTCHAs.

SeleniumBase UC Mode achieves this through several mechanisms, including automatically changing User Agents, setting various Chromium arguments as needed, and providing special UC-specific methods for interacting with web elements.

SeleniumBase offers several advantages over standard Selenium. It provides a higher success rate in bypassing anti-bot measures. The framework includes built-in methods for handling common scenarios, such as reconnecting to avoid detection and interacting with CAPTCHAs when necessary. It also simplifies the process of setting up and managing the Undetected ChromeDriver. Additionally, it offers improved performance and stability compared to basic Selenium implementations.

However, SeleniumBase is not without limitations. While it's effective against many anti-bot systems, it still struggles with highly sophisticated detection mechanisms, particularly those employing advanced AI techniques. SeleniumBase doesn't work well at scale, making it less suitable for large-scale web scraping or automation projects. This limitation can be a major concern for users needing to perform high-volume, concurrent operations.

Conclusion

In this article, we've discussed why websites use anti-bots, how they work and the best ways to avoid bot detection with Selenium.

We also have seen alternative tools, like Undetected ChromeDriver, SeleniumBase, and ZenRows' web scraping API, to avoid detection while web scraping with Python.

ZenRows' web scraping API is the best option to avoid bot detection, and you can get your API key for free.

Frequent Questions

1. How Can I Improve Selenium's Success Rate Against Bot Detection?
2. Does Selenium Bypass Bot Detection?
3. How to Disable Bot Detection?
4. How to Avoid CAPTCHA in Selenium Java?
5. How to Disable the Automation Indicator WebDriver Flag?