Web Scraping Prevention Approaches

Protecting user and business data

1. Introduction

Imagine you are an owner of a trendy real estate website. Each day, dozens of users are consulted by your employees and entrust you with their personal data by publishing their sale and rent offers. Everything looks nice, but someday, one of your employees virtually by chance spots the same offers at another website. The problem is that each offer contains the contacts of a different real estate agency. According to your customers, they have never discussed their properties with a side agency. But the fact is that your data has been stolen, and this situation is gaining traction: some of the customers close their offers, but you cannot realize the real reason.

What happened? Congrats: you are a victim of a web scraping attack. And you are not alone. Recently, LinkedIn and Facebook reported massive scraping attacks against them during the last three years. And with the development of virtual services, scrapers get even more records for further selling at dark markets.

Can we qualify it as a leakage? No. Most of the times the users explicitly confirm their permission to make this data open to general public. The scraped data is fully open and does not contain any sensitive pieces users did not want to publish. Legislations of most countries do not qualify the collection of intelligible data as a definite cybercrime. In the US, data scraping activities are even legal: according to the rule of the Washington court, using automated tools to access publicly available information on the open web is not a crime, even when automated access is banned in the website terms of service. That is why the activities of web scrapers are not regulated; their communities are massive and publicly available (just an example of a Google query), all new tips, tricks, and new approaches are widespread and change in a very swift manner. The interest in web scraping has constantly been growing over the past years. The latest most popular related queries demonstrate the actual technological stack of web scrapers: Puppeteer, Octoparse, and Parsehub.

Moreover, it is nearly impossible to prove a loss resulted from scraping if it is not connected with violating intellectual property rights or unauthorized access to private data.

But still, is scraping harmful? Yes. It is often dangerous because it makes users' personal information vulnerable and can compromise users' privacy. Moreover, data scraping can lead to further cyberattacks and improvement of social engineering approaches to perform more effective spear-phishing attacks using the scraped user data. That is why we can say that although the scraping itself is a fully legal practice, it often becomes a first link in the virtual criminal activity chain. Moreover, website scraping reduces owner's revenues in many ways: they will be analyzed in the next paragraph.

Many businesses widely use web scraping as a part of data mining practices. It allows obtaining information about prospective and actual customers, rivals, and their products, as well as the overall market situation in different dimensions. Data miners often use automated tools for data scraping in order to get massive arrays of structured data related to a requested aspect of company activity. This will be discussed in the third paragraph.

Can you mitigate the risks? Yes. There are many methods and approaches to withstand automated website scraping that will be described in the fourth paragraph of this paper.

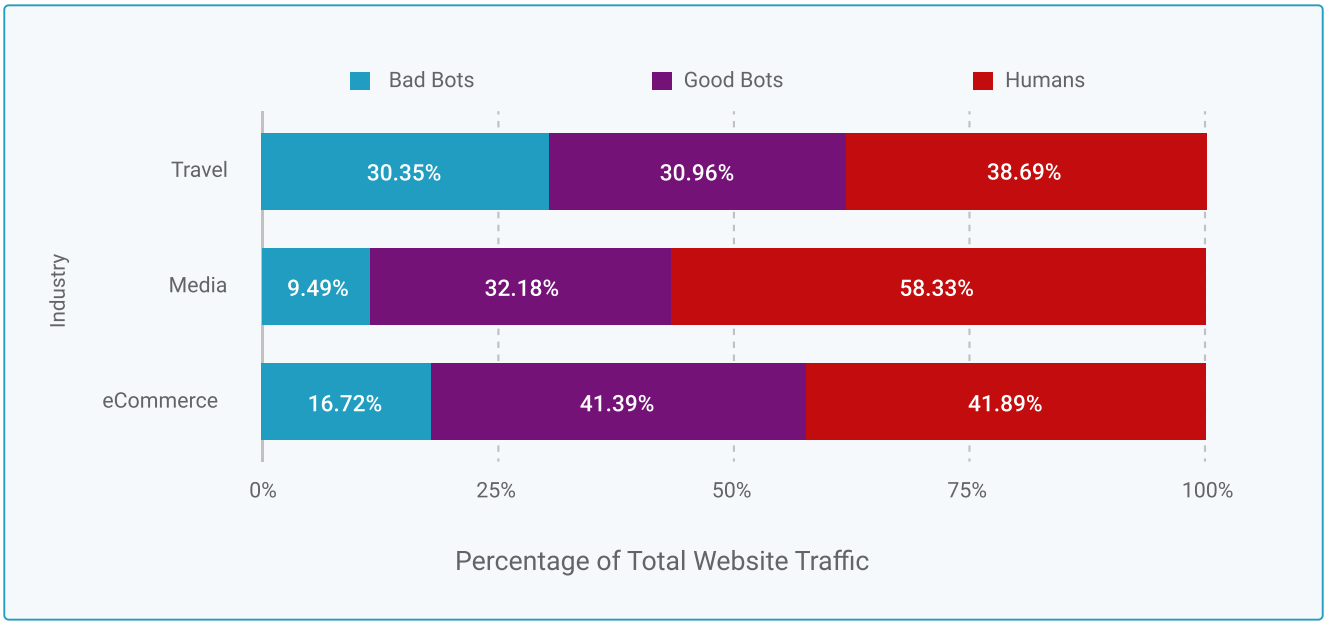
2. How web scraping drives to loss

According to The Business Impact of Website Scraping research performed by Aberdeen Strategy and Research agency, website scraping bots represent a surprisingly high (from 10% to 30% depending on a definite website topic) percentage of total website traffic, which leads to both higher costs and lower revenue due to the following facts:

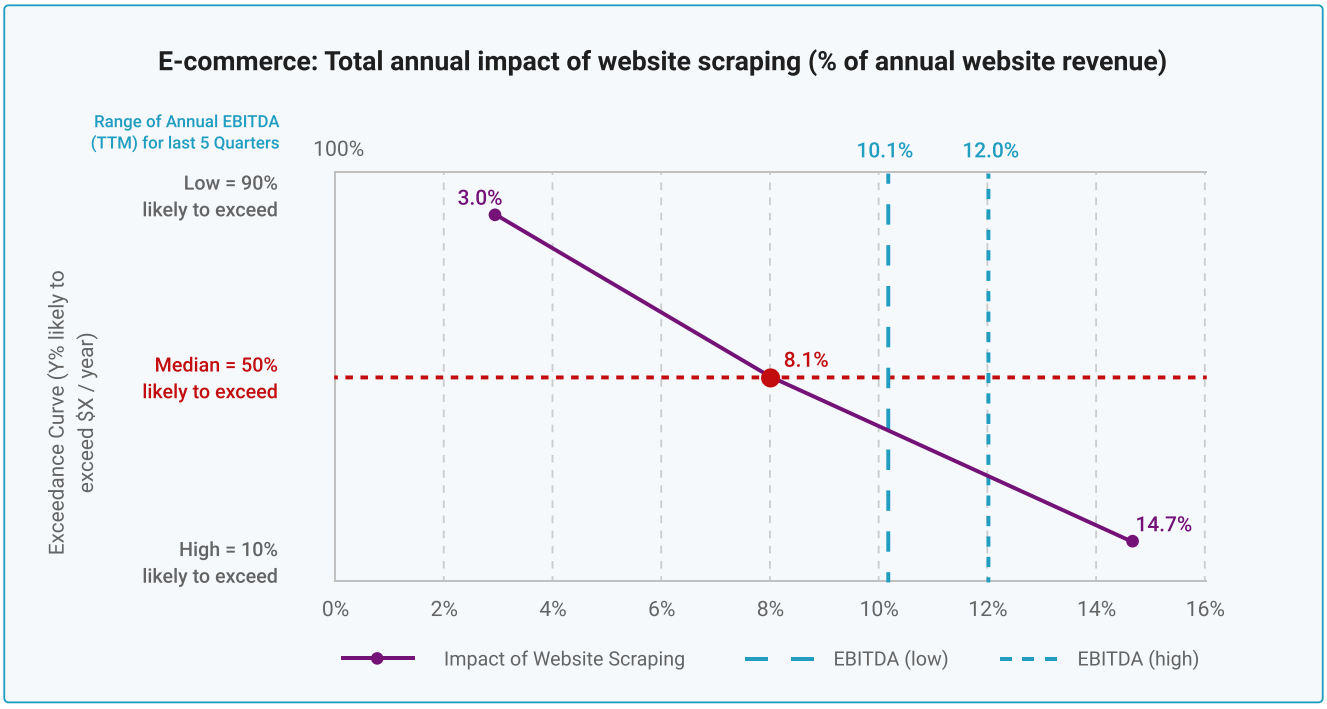
• A website owner has to spend more on website infrastructure because of higher loads and on website marketing because bots are not converted into buyers.

• A website owner makes less profit because of decreased traffic from target buyers, loss of existing or new buyers, and competitive advantages (e.g., products, content, prices).

Web scraping bots represent a significant percentage of total website traffic

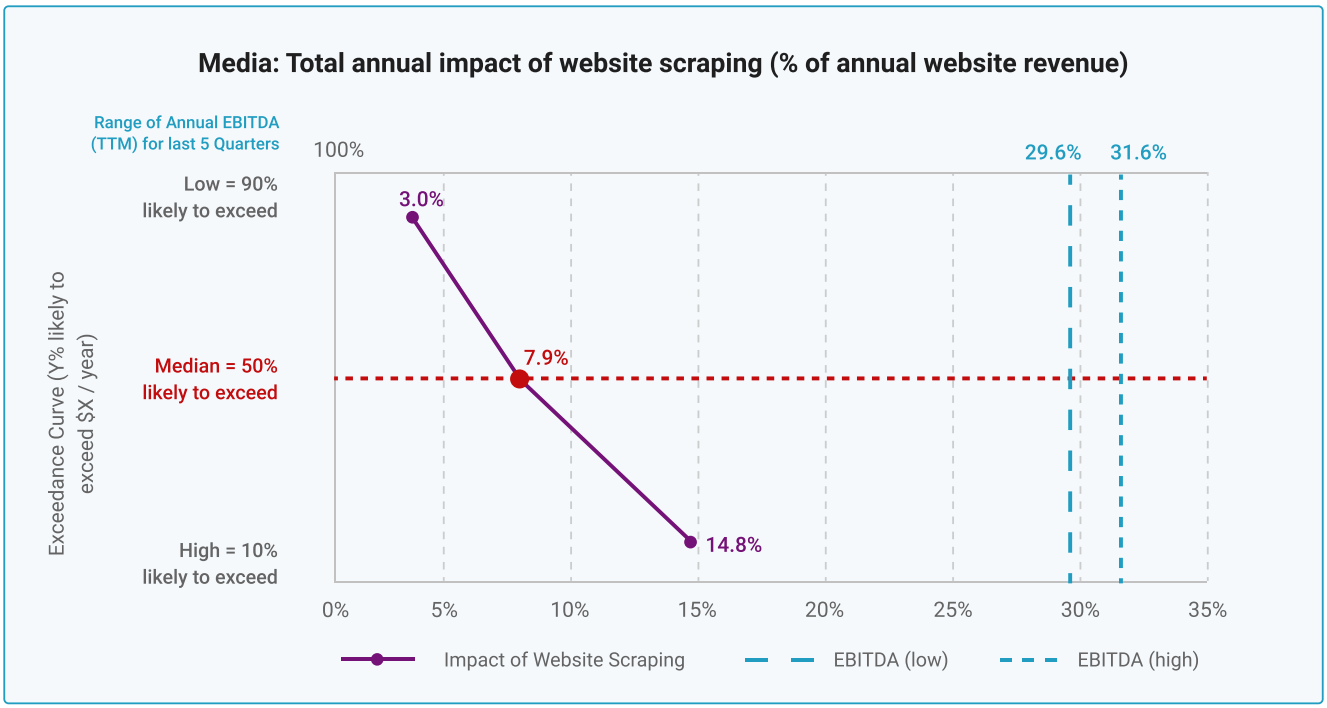


The research covers three industry sectors: e-commerce, travel, and media. It highlights the percentage of annual revenue lost as a result of website scraping.



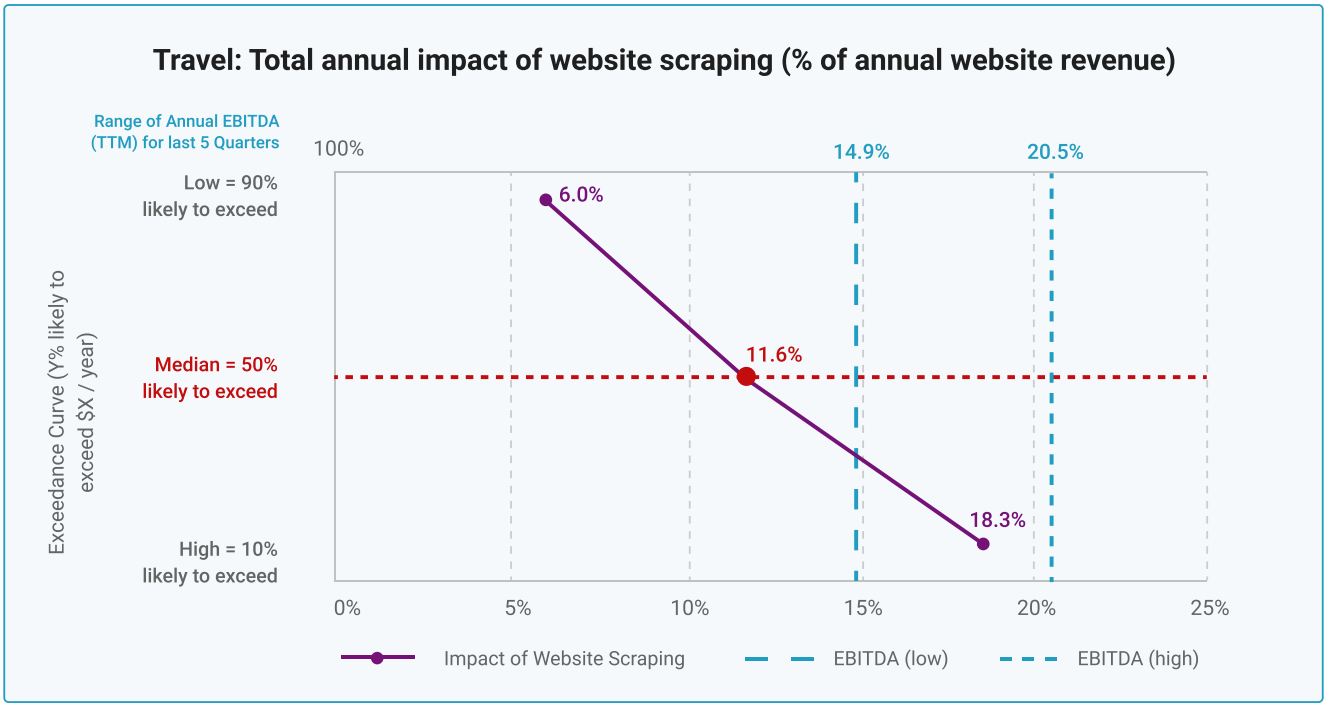
Source: Monte Carlo analysis; Aberdeen, May 2020

For the **e-commerce** sector, Aberdeen's analysis estimates that the annual business impact of website scraping is between 3.0% and 14.7% (8.1% median) of yearly website revenue. For every $1 of website profitability, there is an 80% likelihood that the annual business impact of website scraping will be between $1.23 and $1.46.



Source: Monte Carlo analysis; Aberdeen, May 2020

For the **Media** sector, the annual business impact of website scraping is between 3.0% and 14.8% (7.9% median) of annual website revenue. For every $1 of website profitability, there is an 80% likelihood that the annual business impact of website scraping will be between $0.47 and $0.50



Source: Monte Carlo analysis; Aberdeen, May 2020

For the **Travel** sector, the annual business impact of website scraping is between 6.0% and 18.3% (11.6% median) of annual website revenue. For every $1 of website profitability, there is an 80% likelihood that the annual business impact of website scraping will be between $0.89 and $1.23.

That means that in 2020 the average business impact of website scraping can be calculated from median values as **9.2% of annual website revenue.**

Is this an acceptable risk for a website? Each owner can decide independently and take necessary measures for the prevention of possible loss.

3. Web scraping automation and measures against it

Web scrapers use different automation tools for collecting requested data:

• **Scripts for parsing web pages based on system utilities** (e.g., **curl** or **wget**) are the essential tool in the scraper set. They can vary from 10-line snippets that detect required blocks using regular expressions up to the fully customized code that allows changing required headers and deceiving simple fingerprinting methods. The received response is parsed in order to make a readable database.

• **Scripts and programs using HTTP request libraries** (e.g., Python's requests library). The programming languages themselves, as well as their libraries, have many built-in methods for working with query parameters and parsing responses - that is why a developer can customize a bot in a more flexible way covering obvious bottlenecks in its behavior. Like the previous category, they often mimic 'good' bots (search, indexing, or advertising).

• **Headless browser combined with a testing framework and a content parser.** This is the most popular and flexible tool of web scrapers due to a broad palette of ready-made implementations. As basic frameworks for web testing, Selenium and Puppeteer are the most used and well-documented tools for web scraping. The blogs are complete with ready deployments based on a headless Chrome, Firefox, or PhantomJS initially intended for web testing but used by web scrapers to obtain significant data arrays. The main advantage of such a scraping approach is a browser-like behavior, including running JS elements, transferring screen and window parameters, as well as cursor position, usage of Canvas and WebGL, and processing of visible and invisible page elements. Such solutions require more sophisticated detection tools, as well as the following variant.

• **Puppeteer combined with a headless Chrome** is a particular case of the previous variant, which is worth noticing because of its vast opportunities for web scraping. Puppeteer is a Node library developed by Google as a web testing orchestration tool for headless Chrome or Chromium. In comparison with Selenium, Puppeteer gives the following opportunities:

- Intercept and change browser requests and responses allow to obfuscate bot's behavior and conceal automation.
- Work with JS in a swifter manner decreases the number of errors and development costs, as well as allows to implement more challenges to cheat anti-scraping solutions.
- Operation with a headless Chrome, Chromium or a headful browser and its plugins allows to fully imitate a real user.

This case is tough to detect and requires a very sophisticated approach.

• **Scraping-as-a-Service and scraper platforms.** Technically, these solutions combine the approaches mentioned above, depending on a task and a target resource. The legally operated platforms have opportunities to use residential proxy networks and cloud capacities for a botnet consisting of thousands of headless browsers. The efficiency of these solutions depends on the value of scraped data.

Based on these assumptions, the anti-scraping measures face several challenges. The first challenge has already been mentioned above: a regulatory status of scraping practices varying from fully legal to allowed but indistinct. So far, web scraping is not an offense; each website owner will have to take an independent decision in terms of mitigating its risks.

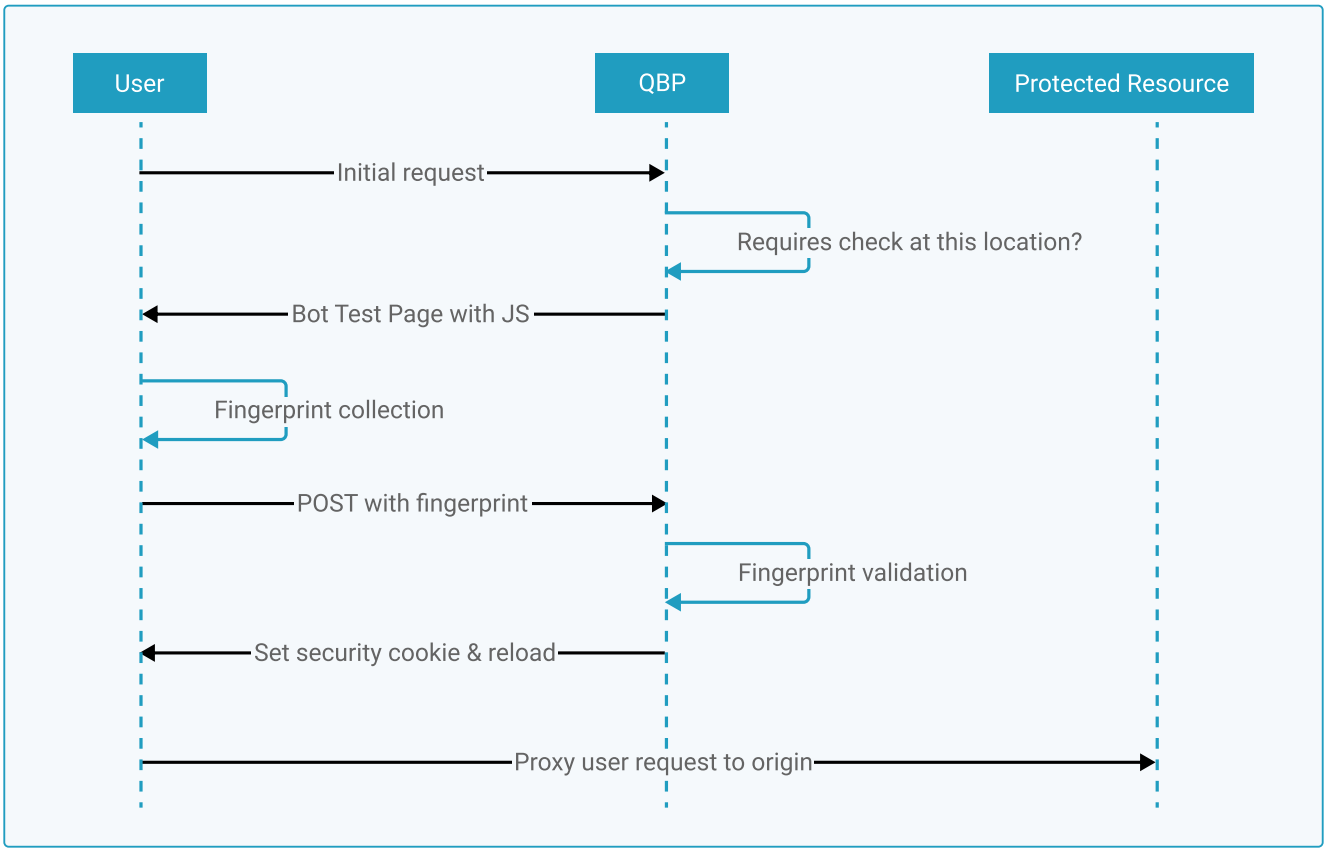
The second challenge is connected with the first one: considering the indistinct regulatory status of web scraping, companies offering services to counter it have to constantly struggle against new know-hows. In fact, scrapers and anti-scrapers participate in the 'arms race' or the race between a round and an armor. Here, the round is gaining even more solutions and implementations - and scraper's costs for developing a new parser cannot be compared with higher protector costs. There **is no system or approach that can guarantee 100% protection from bots.** Currently, quite a resource-consuming composition of components can be tweaked by a scraper in the way that it will be able to take advantage of any detection methods existing on the market. Nevertheless, approaches offered by specialized vendors can significantly decrease the number of bots visiting the resource and, correspondingly, the negative business impact.

4. The combined approach of the Qrator Bot Protection

Qrator Bot Protection (QBP) is a built-in element of the Qrator platform developed to protect websites against automated data scraping. It is included in the delivery set of the platform and can be turned on and set in a special section of the Qrator UI.

For the locations where Web-based users are expected by the protected location, QBP checks the environment of a browser addressing a protected resource and generates a tracking cookie for browsers considered as trusted. At the same time, QBP restricts access to the resource for suspicious browsers.

Qrator Bot Protection Basic flow



For the APIs used by native mobile app users (iOS & Android), QBP supports several protection methods:

- enabling browser-based authentication (BBA) for app users on the login stage, checking the environment and the browser used to perform BBA;
- validating security tokens which the users have to sign their requests with, filtering out the attempts to access the API directly.

QBP checks can be customized using rule sets, e.g., host or host group, definite URI or URI regular expression, blacklisted IP addresses.

The QBP basic flow is illustrated above.

A user corresponding with a predefined rule set will receive an HTTP 401 response page with a built-in JS script for fingerprinting. If the user has a browser (i.e., it is not a script or request generator), this browser executes the fingerprinting script and adds the generated fingerprint into the POST request body to be sent into a definite endpoint controlled by QBP.

QBP validates the fingerprint received in the body, and in case of successful validation, proxies the initial request to the server of the protected resource, adding the user's tracking cookie to it. The tracking cookie allows the checked user to gain access to the protected resource without validating further requests during a session.

QBP can operate in two modes:
• In the **Monitoring mode**, QBP proxies a user's request further in case of any validation result. If a browser sends any response to the QBP, it will not receive an error code, but validation details will be recorded in the event log. The event log can be viewed anytime by the QBP operator.

• In the **Blocking mode**, badly fingerprinted browsers or applications without JS support (including scrapers without the usage of browsers) will receive a customizable block page. A legitimate user will see the checking page first (a blank or a customized 401 page), and after validation will get the requested page.

The product's interface supports permissions for good bots, QA and other cases requiring bypassing the checks. Trusted IP addresses, CIDR lists, geozones and header rules can be specified to set up these scenarios.

Another useful parameter is the A/B distribution which adjusts the percentage of the user base affected by the mitigation algorithms. Using this feature makes tasks like adding new domains, applying changes and testing the rule pages smooth and controllable.

QBP can also operate with different borderline cases when protected domains are connected with unprotected ones: following recommendations will help to set the protection for partly-protected infrastructures:

• **Cross-domain access and authorization.** For instance, a D1 domain is protected, and users can gain access inside the resource only after the validation. But at the same time, a D2 domain is not protected, and users can generate requests to D1 URLs from the pages of D2. D2 users are not validated by the D1 protection and do not receive tracking cookies - that means they will not be able to get data from the D1. If there are such scenarios between the resources, it is recommended to set corresponding exceptions for D1 pages and an opportunity to be validated after authorization using the QBP tools. It is also recommended to exclude API endpoints that allow addressing to the protected D1 from the unprotected D2.

• **Cross-origin resource sharing.** In case of exchange with CORS requests between protected D1 and unprotected D2 domains, requests will not include current user cookies. That is why

users will receive an error page or a malfunctioning validation page. In order to exclude such scenarios, it is recommended to pass cookies in CORS requests and place corresponding domains in the same domain instance of Qrator.

• **Webview** and **iframe.** If resource pages are used by a mobile application in the webview mode or as iframes in external resources, there may be specific corner cases in which the page does not refresh and complete the validation routine. In case passing and setting the required cookies through the iframe, it is recommended to exclude such pages from the QBP validation mechanism.

5. Conclusion

Web scraping is intended to solve many tasks: scientific, commercial, or social. But any scraping method can be easily used in harmful or even criminal ways with a noticeable negative business impact. It is impossible to prohibit everyone from gaining access to open and intelligible data because they act as a service per se (e.g., social networks) or serve as a source for other services (e.g., e-commerce, financial data sources, etc.). It is also impossible to provide 100% protection from web scraping. Nevertheless, every website owner can minimize its impact by using different methods and services and setting rules for visitors in order to prevent any harmful activity.

Qrator Bot Protection is a solution that can help online business owners define these rules and their usage criteria. It gives an opportunity to monitor user compliance with the pre-set rules, as well as to take different decisions regarding violators.

The web scraping landscape is constantly changing in terms of threats, business requirements, the know-how of scrapers, and protection methods. Qrator Bot Protection will be adopted to these continually changing game rules.

+420 602 558 144
mail@qrator.net

LinkedIn Facebook Medium