Preventing Web Scraping
Best Practice
October 2014

PREPARED BY

Paola Di Cretico
Creative Digital Ideas

Paola Di Cretico

Table of Content

1. Document Objective

The aim of this document is to outline the best practice to prevent and mitigate web scraping

2. Definition

Web scraping, known as content scraping, data scraping, web harvesting, or web data extraction, is a way of extracting data from websites, preferably using a program, or bot (short for web robots) that sends a number of HTTP requests, emulating human behavior, getting the responses and extracting the required data out of them.

Common methods to address web scraping attacks are:

» ***Session opening*** detects an anomaly when either too many sessions are opened from an IP address or when the number of sessions exceeds a threshold from an IP address. Also, session opening can detect an attack when the number of inconsistencies or session resets exceeds the configured threshold within the defined time period. This method also identifies as an attack an open session that sends requests that do not include an ASM cookie.

» ***Bot detection*** investigates whether a web client source is human by limiting the number of page changes allowed within a specified time.

» ***Session transactions anomaly*** captures sessions that request too much traffic, compared to the average amount observed in the web application. This is based on counting the transactions per session and comparing that to the average amount observed in the web application.

3. Best Practice

Unfortunately, there is no efficient way to fully protect our website from data scraping. This is so because data scraping programs (also called data scrapers or web scrapers) obtain the same information as our regular web visitors.

It is impossible for traditional network security devices such firewalls, intrusion detection and prevention, or even application layer firewalls to detect or block them as sophisticated scraping tools mimic user search patterns, however there are developing technical counter measures for detecting the practice.

Hence, to mitigate the risk of our site being scraped a combination of the following measures should be implemented.

3.1 Concurrent Login (session opened)

Scrapers are more likely to open an unusually high number of concurrent connections and access many more pages than most users.

It is recommended to add user capabilities that allow checking the details of active sessions at any time, monitor and alert the user about concurrent logons, provide user features to remotely terminate sessions manually, and track account activity history (logbook) by recording multiple client details such as IP address, User-Agent, login date and time, idle time, etc.

Proposed changes

All external users access should be limited to 2 concurrent login.

3.2 Source IP Address Changes

Scrapers are more likely to open an unusually high number of connections changing frequently the IP addresses.

It is recommended to add user capabilities that allow tracking account activity history (logbook) by recording multiple client details such as IP address, User-Agent, login date and time, idle time, etc.

Recommended actions: implementing the following security controls for sessions where the source IP address changes:

- Log source IP address changes during application sessions.
- Notify the user of potentially suspicious activity.
- Possibly require the user to re-authenticate to ensure the change is legitimate.

3.3 BOT Traffic

There are different web robot types:

- ***White bots (good)*** like search engines (Google, Bing and Baidu) help drive more customers to the site and therefore increase revenue. They also help monitor the site availability and performances (Akamai site analyzer, Keynote, Gomez) as well as pro-actively look for vulnerabilities (Whitehat, Qualys)

- ***Black bots (bad)*** send additional traffic to the site that may impact its availability and integrity. Bad bot traffic can drive customers away from the site, negatively impacts revenue and the web site reputation. For example Hackers trying to bring down a site with a DDoS attack or exposing / exploiting vulnerabilities. Competitors or other actors scrapping a site to harvest pricing information to be used for financial gains.

- ***Grey bots (neutral)*** don't necessarily help drive more customers to the site, nor do they specifically seem to cause any arm. Their identity and intent is more difficult to define, they usually present characteristics of a bot but are usually non aggressive. Such traffic would only occasionally cause problem due to a sudden increase of the request rate.

Identifying bot traffic

Dealing with bot traffic can be challenging and pro-active measures should be taken to prevent any negative impact on the site. Monitoring bot activity is key. The one thing that all bots have in common is that they only request base HTML pages, which usually contain valuable information but are also more process intensive for the web server to generate. Bots generally never request any of the embedded objects (images, JavaScript, Cascaded style sheets) just because the client doesn't need to render the full page.

- *White bot* traffic is usually predictable. It will have a specific header signature and will come from IPs belonging to the companies managing the bot. It is possible to control what these bots can request on the site through robot.txt or through the administration interface of the service managing the bot activity.
- *Black bots* header signature will widely vary from exactly mimicking a genuine browser or search engine request to something that will present several anomalies with missing headers or atypical headers being present in the request. Black bots may also send requests at a higher rate.
- *Grey bot traffic* can be more challenging to identify since it generally present the same characteristics as black bots.

Mitigating bot traffic

Once bot traffic is identified, the next step is to decide what to do with the black and grey bot traffic. The type of action taken may vary depending on the business needs:

- Deny the traffic: this is the default but least elegant solution; client will receive a HTTP 4xx or 5xx response code or a CAPTCHA. This will give the bot operator a clear indication that such action is not allowed on the site and that they've been identified by some security service or device. Bot operators could vary the format of the request and see if they can stay under the radar.
- Serve alternate content: the content served could vary from a generic "site unavailable" page to something that looks like a real response but only containing generic data. This strategy may slow down the bot operator and keep them in the dark as too why they cannot access to the data they want.

- Serve a cached / stale / static / version of the content: This is the best strategy of all but not always necessarily possible to implement, some content just cannot be cached or stored as static data on an alternate origin, because of compliance concerns or its dynamic nature. It could potentially take the bot operator some time to realize the data they are getting is worthless, an attacker running a DDoS against the site would also get discourage and move on to a different target.

3.4 Rate limiting

Advanced scraping utilities are very adept at mimicking normal browsing behavior but most hastily written scripts are not. However, automated clients will expose themselves by requesting Web content after a consistent time period. Bots will follow links and make web requests at a much more frequent and consistent rate than normal human users.

To rate limit an IP address means allowing the IP a certain amount of searches in a fixed time frame before blocking it. This might be difficult to implement as many of our users are likely to come through proxy or large corporate gateways which often share with thousands of other users.

Solutions:

- Manually compile and maintain a white list, since IP addresses change over time. (Hard to maintain)
- Investigate each “potential scraper” before blocking them.

Rate limiting technique usually work for a little while as most scrapers are used to distribute themselves over a large number of accounts or IP addresses in order to avoid detection.

3.5 Use Cookies or JavaScript to verify clients program is standard Web browser

Simple scraping tools cannot process complex JavaScript code or store cookies. To verify that the client is a real web browser, it is best practice to inject a complex JavaScript calculation and determine is JavaScript is correctly computed.

This technique works well against normal bots, but not against advance scraping tools as Connotate that have built-in JavaScript interpreter or any other scraping tools built on Node.js (see headless browser section).

3.6 Implementing CAPTCHA

Completely Automated Public Turing Tests to Tell Computers and Humans Apart (CAPTCHA) exists to ensure that user input has not been generated by a computer. This has been the most common method deployed because it is simple to integrate and can be effective, at least at first. It works with less advance scrapers who are not motivated to alter their scraping tools and techniques.

There are two ways to circumventing CATCHA tests: 1) by using OCR (Optical Character Recognition) software or 2) by employing labor in low cost countries to manually solve them.

By using increasingly complex algorithms, programmers have managed to get a 5% success rate at solving even the reCAPTCHA test which is one of the hardest CAPTCHA challenges out there[1]. An interesting side effect of this may be that using a reCAPTCHA test may significantly increase the scraping related traffic to our websites as they will need 20 searches instead of one at a 5% success rate. Less effective programmers will require more attempts.

The fact that CAPTCHA challenges can be circumvented is however not the primary objective against using them, it is that they degrade the usability of a website. The harder CAPTCHA challenges are troublesome even for humans to solve and used in the wrong place on a website may significantly lower the visitor numbers.

There are ways of limiting these effects by using CAPTCHA in conjunction with other means to detect scraping. The most basic example is to only send CAPTCHA tests to clients making more than a certain number of requests; this will help most users of the website including scrapers as they will not have to fill out as many CAPTCHA tests. Another method is to send CAPTCHA challenges to IP addresses geographically located in places where we normally do not have many visitors. Many websites are country or language specific, and we can block off countries that normally harbor the open proxies or anonymizing services that scrapers use.

All implementations of CAPTCHA tests naturally come with the challenge of keeping whitelists up-to-date. Almost all websites have partners, friendly bots, and other allowed automated users of the website.

The problem is that Captchas can be beaten with a little work and more importantly, they are a nuisance to end users that can lead to a loss of business.

[1] Will a CAPTCHA test stop scraping? – Sentor Blog 15 January 2014.

3.7 Headless Browsers

A headless browser is a web browser without a graphical user interface. In other words it is a browser, a piece of software, that access web pages but doesn't show them to any human being. They're actually used to provide the content of web pages to other programs.

The headless browser is significant because it understands web pages like a browser would – with the caveat that browsers all behave slightly differently. Headless browsers, for example, are able to parse JavaScript. They can click on links and even cope with downloads.

Google: A proposal for making AJAX crawlable

Many headless browsers software are built on Node.js, a very powerful platform built on Chrome's JavaScript runtime.

Here's a list of crawlers you can get to deploy on Node.js https://nodejsmodules.org/tags/spider

It is recommended to test our site scrapability to PhantomJS [phantomjs.org] which is a scriptable headless WebKit with a JavaScript API to understand how much data can be scrapped.

HTMLUnit

From Wikipedia: **HtmlUnit** is a headless web browser written in Java. It allows high-level manipulation of websites from other Java code, including filling and submitting forms and clicking hyperlinks. It also provides access to the structure and the details within received web pages. HtmlUnit emulates parts of browser behavior including the lower-level aspects of TCP/IP and HTTP.

A sequence such as getpage(url), getLinkWith("Click here"), click(), allows a user to navigate through hyperlink text and obtain web pages that include HTML, JavaScript, Ajax and cookies. This headless browser can deal with HTTPS security, basic http authentication, automatic page redirection and other HTTP headers; it allows Java test code to examine returned pages either as text, an XTML DOM, or as a collection of forms, tables and links.

The most common use of HtmlUnit is test automation of web pages, but sometimes it can be used for web scraping, or downloading website content.

Version 2.0 includes many new enhancements such as a W3C DOM implementation, Java 5 features, better XPath support, and improved handling for incorrect HTML, in addition to various JavaScript enhancements, while version 2.1 mainly focuses on tuning some performance issues reported by users.

--

There is no way to prevent scrapers using this type of technology from scraping our site. The best solution will be to implement an intelligent monitoring tool to detect and block abnormal bot behavior.

3.8 Converting data into Images

Some scrapers will only access html files and ignore images and other binary files. Advance scraping tools, such as Mozenda, gives clients the ability to store images as well as html files. In addition, some well determined scrapers (using Django) can use OCR to parse information out of the image.

Embedding content in an image or other media objects will probably slower the site for the average user; not to mention all the extra time it takes to update the content on our site.

Another way that would at least make scraping by bots harder (but still not prevent people from taking screenshots) would be to save the images on our server in a scrambled or encrypted form and then use some client-side logic (in Javascript for example) to unscramble them.

The downsides of this approach are that the clients will need to have the ability to do the unscrambling technically and performance-wise in the first place and since the unscrambling logic would run on the client, there is always the possibility for dedicated people to reverse-engineer it.

Please note: Listing data as an image without a text alternate is in violation of the Americans with Disabilities Act (ADA).

3.9 Block Copying

By disabling JavaScript and CSS style text can be blocked from being copied.

Many scrapers will use automated scraping tools that include an interpreter in the back end that will read and copy the text.

Alternately, there are a number of Firefox application settings held in the prefs.js and user.js files of Firefox's package contents that can be easily viewed in the clients' browser by typing about:config in the address bar.

Then, it is sufficient to modify the dom.event.clipboardevents.enabled preference by changing its value from "true" to "false" to allow scrapers paste and copy the selected content.

Hence, Block copying can be beaten with a little work and more importantly, they are a nuisance to end users that can lead to a loss of business.

3.10 Block View Source

JavaScript Encryption is by far the most popular way to try to hide the source code.

It involves taking our code, using a custom made function to "encrypt" it somehow, and then putting it in an HTML file along with a function that will decrypt it for the browser.

Many browsers provide alternative ways around this. Some allow users to save the page, decrypted for easy viewing later. Others, like FireFox, include tools like the DOM Inspector, which allows users to easily view and copy the XML of the page, decrypted.

Hence, Block View Source can be beaten with a little work and more importantly, they are a nuisance to end users that can lead to a loss of business.

3.11 Account management

To mitigate web scraping vulnerability it is recommended to implement robust account management architecture to enabling us to uniquely identifying our users. Currently, no measures are in place to ensure registered users regularly update their profile information. Users (especially self-registered) shall be prompted to update their profile regularly.

Hard data stored in their profile, coupled with security questions might then be used to uniquely identifying them.

Any security questions or identity information presented to users should ideally have the following four characteristics:

- **Memorable**: If users can't remember their answers to their security questions, you have achieved nothing.
- **Consistent**: The user's answers should not change over time. For instance, asking "What is the name of your significant other?" may have a different answer 5 years from now.
- **Nearly universal**: The security questions should apply to a wide an audience of possible.
- **Safe**: The answers to security questions should not be something that is easily guessed, or research (e.g., something that is matter of public record).

Best practice for security questions

- Display the security question(s) on a separate page only *after* your users have successfully authenticated with their usernames / passwords (rather than only after they have entered their username). In this manner, you at least do not allow an adversary to view and research the security questions unless they also know the user's current password.

» If you also use security questions to reset a user's password, then you should use a *different* set of security questions for an additional means of authenticating.
» Security questions used for actual authentication purposes should regularly expire much like passwords. Periodically make the user choose new security questions and answers.
» If you use answers to security questions as a *subsequent* authentication mechanism (say to enter a more sensitive area of your web site), make sure that you keep the session idle time out very low...say less than 5 minutes or so, or that you also require the user to first re-authenticate with their password and then immediately after answer the security question(s).

4.11 2- Step Identity Verification Mechanisms

To mitigate web scraping vulnerability it is recommended to implement an extra layer of security on registration/login that in turn will enable us to uniquely identify our end users.

Multi-factor authentication (MFA) is using more than one authentication factor to logon or process a transaction:

- Something you know (account details or passwords)
- Something you have (tokens or mobile phones)
- Something you are (biometrics)

Authentication schemes such as One Time Passwords (OTP) implemented using a hardware token can also be key in fighting attacks such as CSRF and client-side malware. A number of hardware tokens suitable for MFA are available in the market that allow good integration with web application.

Two-step verification is a process involving two subsequent but dependent stages to check the identity of an entity trying to access services in a computer or in a network with just one factor or secret, whilst there is no proof obtained that the bearer of the unit is identical to the owner of the unit.

2-level identity verification mechanism adds an extra layer of security to users account, drastically reducing the chances of having personal information stolen and unlawful access to your site.

It is best practice to deploy a Mobile app (or SMS) identity verification step to identity the users.

4. Issues on handling scraping prevention in-house

To be able to mitigate scraping issues in-house we need:

» Development resource to continuously develop our countermeasures as scrapers will try and get around them.
» Analysis resources to analyse performance of the solution and suggest changes to dev.
» Architects to design the solution and keep up with what is happening around the internet to make sure the solution keeps working and do what it does.

It is important to emphasis that the resources will need to be committed on an ongoing basis.

5. State of the art anti-scraping tools

SENTOR
http://www.scrapesentry.com/

FIREBLADE
http://www.fireblade.com/

CLAREITY SECURITY
http://clareitysecurity.com/