2021 Palestinian International Conference on Information and Communication Technology (PICICT) | 978-1-6654-3651-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/PICICT53635.2021.00038

Mitigating Web Scrapers using Markup Randomization

Noor Bolbol
Information Technology
Islamic University
Gaza, Palestine
noorbolbol8@gmail.com

Tawfiq Barhoom
Information Technology
Islamic University
Gaza, Palestine
tbarhoom@iugaza.edu.ps

***Abstract*—Web Scraping is the technique of extracting desired data in an automated way by scanning the internal links and content of a website, this activity usually performed by systematically programmed bots. This paper explains our proposed solution to protect the blog content from theft and from being copied to other destinations by mitigating the scraping bots. To achieve our purpose we applied two steps in two levels, the first one, on the main blog page level, mitigated the work of crawler bots by adding extra empty articles anchors among real articles, and the next step, on the article page level, we add a random number of empty and hidden spans with randomly generated text among the article's body. To assess this solution we apply it to a local project developed using PHP language in Laravel framework, and put four criteria that measure the effectiveness. The results show that the changes in the file size before and after the application do not affect it, also, the processing time increased by few milliseconds which still in the acceptable range. And by using the HTML-similarity tool we get very good results that show the symmetric over style, with a few bit changes over the structure. Finally, to assess the effects on the bots, scraper bot reused and get the expected results from the programmed middleware. These results show that the solution is feasible to be adopted and use to protect blogs content..**

Keywords—Web scraping, web crawler, randomization, markup HTML, content security.

I. Introduction

Web scraping, also known as web extraction, is the set of techniques to extract certain kinds of information from the World Wide Web (WWW) and save it in a structured format for later retrieval or analysis [1]. This is accomplished either manually by a user or automatically by a bot or web crawler. Web scraping is related to web indexing, whose task is to index information on the web with the help of a bot or web crawler. Web scraping comprises three primary and intertwined phases: 1) website analysis, 2) Website Crawling to fetch the desired links, 3) Data extractor to fetch the data from the links, and storing that data into a structured form [2]. It primarily used in price scraping, article extracting from blogs, job portals, financial information sites, and so on. The data available on the Web comprises structured, semi-structured, and unstructured quantitative and qualitative data in the form of Web pages, HTML tables, Web databases, emails, tweets, blog posts, photos, videos, and so on. Practitioners can leverage this data to better understand their customers, formulate strategies based on their findings, and, ultimately, improve organizational performance [3]. This technique may be used in an ethical and unethical way, where the policy and copyright of the websites decide the legality of the use. The unethical scrapers bot gather and exploit web content, to republish content with no overhead, or to undercut prices automatically, this act compromise the ownership rights of the content and classified as a data theft crime.

In 2017 the statistics show that over 55% of web traffic is from bots, scrapers, and spammers, where only 45% is from humans. They return this results to the rapid rise of using web crawling because of the increase in the volume of time-sensitive and dynamic data posted to social networking and the web [4].

To overcome and mitigate the bad effects of these scrapers, many researchers study the problem and proposed several solutions that differ in their effect and efficiency. Some of these techniques, like paper [5], focus on detecting malicious visitors by using machine learning to classify the user's behavior. other solutions focus on protecting the content before it crawled by the bots. Diab in his paper [6] proposed a solution to protect the data by using randomization of the CSS class names and also, randomizing the Markup which changes the HTML.

In our paper, we shed the light on the web scrapers that are directed to extract articles from blogs and copying their content to another target. The proposed solution mitigates these bots and makes the automatic copying for the content more complicated and needs human participation. The mechanism based on two steps, the first one to tick the bots by adding a random number of duplicated URLs within the list of the articles URL on the blog page, then the next step is to protect the article content itself by adding random spans contains randomly generated strings within the article body, which will not affect the appearance of the page.

The paper is organized as follows, Section II lists and briefly describes related works, Section III explains the proposed solution, Section IV reports the extracted results, and we draw conclusions in Section V.

II. Related Works

Most of the previous studies and solutions focus on mitigating and preventing web content scraping by identifying the scraping bots and blocking them from access the content by analyzing the bot's behavior.

The paper [6] aims to provide a way to prevent web scraping based on CSS and XPath. The proposed solution is based on renaming all CSS class names with randomly generated names of 16 characters and creating a dictionary for mapping original names with the newly generated ones.

Another part of the solution is preventing XPath-scrapers by renaming CSS attributes as works in the previous par and, adding empty HTML tags to the file to confuse the scrapers. The HTML page is synchronized with the new class names and cached on the server disk to minimize the required time to re-generate names for every request. The cached pages to the client's browsers. To achieve this purpose they used Cron Jobs to automate the randomization for the pages to keep them fresh and at the same time keep them safe from prediction and detection.

For evaluating the proposed solution, they evaluate the experience on 10 websites from different categories based on three criteria:

1. Similarity: for measuring this they used Romero's and Marıa's approach that compares the web pages visually by transformation and compression, then calculates the similarity by their suggested formulas.
2. Processing time: the required time to generate randomization and render the new webpage on the browser, is a critical point to be measured, the more time affects user experience.
3. File size: comparing the size of the file before and after applying randomization, to keep track of the usage of server resources.
4. The effectiveness of preventing scrapers by re-run web scraping after applying the randomization.

The results of the experiment show that the similarity between the new page and the original page is 97%-100%, and the file size reduced on some web pages while it increased on other webpages depending on the removal of unnecessary HTML tags. And for processing time is increased depending on file size and it ranges between 2 to 5 minutes. And when re-scrap the websites from the collected dataset they found that it did not scrap any single data, which indicates the effectiveness of the proposed solution.

Haque and others in their paper [5] discussed briefly the anti-scraping applications, and propose an anti-scraping application by combining multiple techniques. The proposed solution follows this procedure when a user requests a webpage a filter first check if the user is from the blacklist and block the request if he is from the gray list, a CAPTCHA presented to be solved, and for protecting the valuable data, they propose to convert it from text to an image to prevent copying it, also, the rest HTML page is randomized using an automated randomizer, finally, the session logged in the database for analysis purpose. They proposed to use two Cron Jobs, a daily one to perform frequency analysis, interval analysis, and URL analysis. And a weekly one to perform traffic flow analysis. This paper coincides with the studied paper in using markup randomization, but they did not provide an actual experiment or any measurement criteria for the effectiveness of their solution. On the other hand, they support their solution by using multiple anti-scraping applications. They proposed a way to historical analysis of the visitor's IP address and classify them into three types (Black-list, Gray-list, White-list), the blacklist blocked from accessing the website, while the grey list treated with multi-tier defenses such as CAPTCHA, Honeypots, and Honeynets.

This proposed solution has a drawback because the scraper developed itself continuously. Also, they proposed to make the model change the information to an image so that the scraper will not reach any valuable text but this can be easily broken using Optical Character Recognition (OCR) techniques, also, it affects the user experience. Another issue is with the point that model changes the markup randomly to stop the scraper from getting data using old CSS selectors, this way not efficient if the scraper uses an XPath-based scraper. Also, they did not apply a full view of the randomization technique and the caching issue.

Parikh and others in [6] used Machine Learning algorithms to create a tool that captures the signature of malicious users and detects scraper patterns in real-time by feeding the Apache logs to the Kabana visualizing tool and flagging the attacker patterns, and generate a graphical view for facilitating tracking by customers, also, save collected data into logs to feed the tool with new training data. The results show that this solution is not too good because the scraper becomes more intelligent which makes the distinction more difficult, and the data they feed is collected from unstructured data from the websites the quality of this data mainly affects the results of the algorithm. This paper tries to detect and prevent scrapers from accessing web page data, while the studied paper trying to protect the content itself.

Yue Li & Kun Sun in their paper [7], developed a PathMarker mechanism to detect internal web crawler bots that have valid user accounts who can access valuable data in the website, based on the difference in behavior between normal users and the crawlers. They notice that the crawlers access URLs in a similar pattern regarding the path (depth-first, width-first and, random access), while normal users have obvious access in the short-term and have no pattern in the long-term access.

Their tool working by adding an encrypted marker to the end of all URLs on the website, this marker records the page the referrer page, and the ID of the user who accesses the URL. The marked URL force crawler to repeatedly visit the same page that has different URL markers for different crawlers, which suppress the scraping efficiency of crawlers before they are detected. Also, they use the Support Vector Machine (SVM) algorithm to detect the access patterns and show the CAPTCHA mechanism for malicious users. The result shows the effectiveness of detecting all crawlers such as Googlebot (Sexton) and Yahoo Slurp (Yahoo).

This paper adopted the machine learning approach on detecting web crawlers besides the using of marking URLs with heuristical data, this paper differs from the studied paper on detecting the crawler before accessing the web pages.

Rojas in paper [8] found that using HTML templates facilitates the scraping process and makes the analysis of HTML structure much easier, so they proposed a technique to reduce the vulnerability of scraping and extracting massive content by generating a layer of randomization that allows producing unpredictable HTML structures. The randomization is based on a framework generator of random HTML tags oriented to presentation layers to generate random HTML tags within the template.

The Helper combines the generation of HTML tags and their respective identifying attributes in a random way for each record obtained from the database to add a layer of randomness to the traditional way of displaying the records of a database, this process will allow the page to show different structures after each request with the purpose that automated web content analysis by external programs has a much higher complexity. This solution is based on the Pattern called Template View.

They applied their proposed solution to a test website and notice the reduction of the extracted amount of content. They recommend taking into consideration the need to increase the infrastructure of web applications, in order not to affect the user experience. Also, consider the user interaction to not differ before and after applying the helper.

This thesis very close to the studied paper, which using randomization techniques in changing HTML structure, but they did not apply any evaluation criteria except re-scraping the website. Also, they did not provide any caching mechanism to the generated web page.

III. METHODOLOGY

In this section of our paper, we explain our proposed solution to mitigate the scraping of blog articles and protect content from automatic theft. It based on two steps:

1. To make crawling of articles URLs non-sufficient from the blog main page, we add a random number of duplicated anchors of URLs, these anchors inserted between the articles links, will not affect the appearance of the page.
2. to protect the article content itself in the article page from extracting and adding to another site, we add a random number of spans among the text of the article body, contains hidden random characters, this will not affect the appearance of the page but it will make the extracted text useless and have to be revised manually and edited before auto adding to another site, we suggest to use Diab's solution [6] in randomizing the class name "hide" to make it difficult to be removed.

Our proposed solution focuses on changing the markup randomly which mitigates content scraping. We do not adopt CSS-based scraping because we see that it is not too effective if the developer used class id instead of class names, so we can not depend on class name randomization. On the other hand, it is more efficient to change only the valuable piece of data instead of manipulating the whole website page's content.
his proposed solution is good so far but has a critical drawback in processing time, it takes 2-5 minutes to generate and synchronize the HTML page. so we go a little bit further with his solution to focus on blog protection by randomly adding some tags to confuse the bots without changing any CSS or XPath and without affecting the appearance.

We developed a simple web crawler and scraper using Python package Scrapy, to crawl articles from the blog main page, Fig. 1 shows the results, then get article content by XPath, we will use this scraper to check our proposed solution.

The scraper works as follow in Fig. 2:

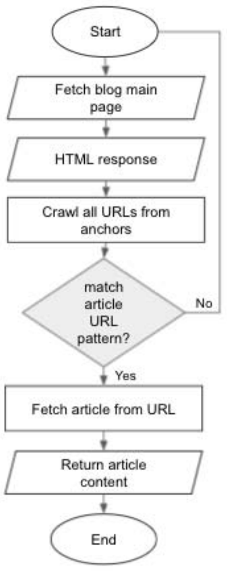
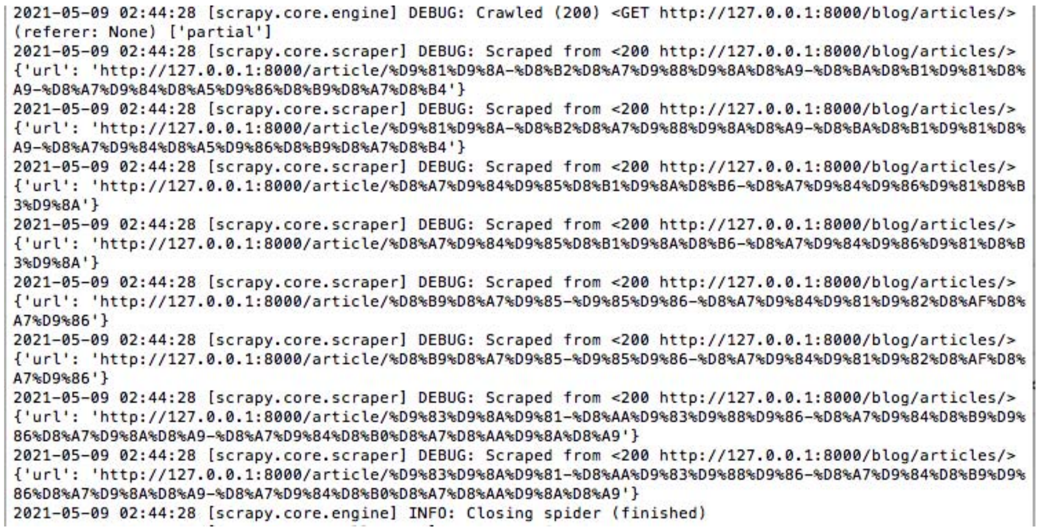


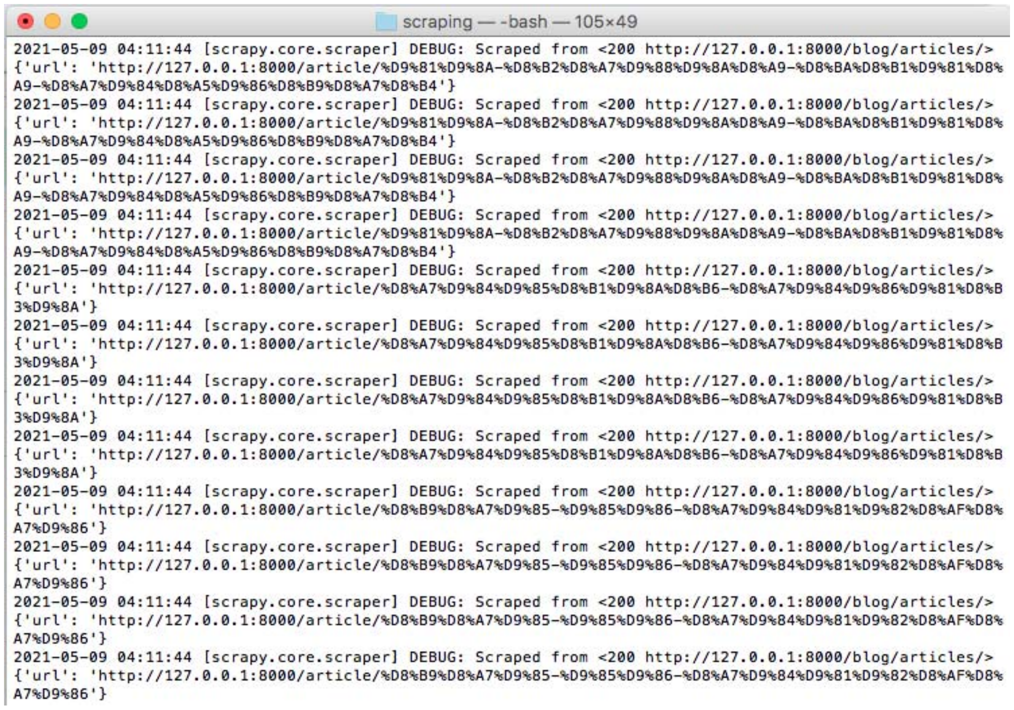
Fig. 2 scraper workflow

When a user requests the blog web page, the code generates duplicated URL after each record from the database, Fig. 3 shows the results, it works as the flow in Fig. 5 shows.



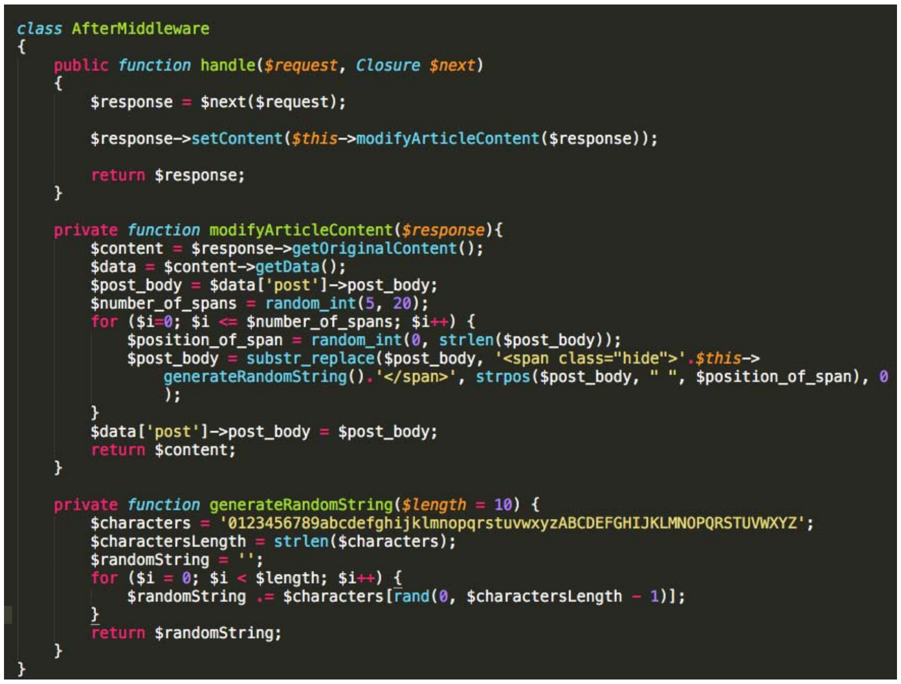
```
2021-05-09 02:44:28 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://127.0.0.1:8000/blog/articles/>
(referer: None) ['partial']
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%83%D9%8A%D9%81-%D8%AA%D9%83%D9%88%D9%86-%D8%A7%D9%84%D8%B9%D9%
86%D8%A7%D9%8A%D8%A9-%D8%A7%D9%84%D8%B0%D8%A7%D8%AA%D9%8A%D8%A9'}
2021-05-09 02:44:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%83%D9%8A%D9%81-%D8%AA%D9%83%D9%88%D9%86-%D8%A7%D9%84%D8%B9%D9%
86%D8%A7%D9%8A%D8%A9-%D8%A7%D9%84%D8%B0%D8%A7%D8%AA%D9%8A%D8%A9'}
2021-05-09 02:44:28 [scrapy.core.engine] INFO: Closing spider (finished)
```

Fig. 1 Result of crawled URLs before the solution.



```
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D9%81%D9%8A-%D8%B2%D8%A7%D9%88%D9%8A%D8%A9-%D8%BA%D8%B1%D9%81%D8%
A9-%D8%A7%D9%84%D8%A5%D9%86%D8%B9%D8%A7%D8%B4'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%A7%D9%84%D9%85%D8%B1%D9%8A%D8%B6-%D8%A7%D9%84%D9%86%D9%81%D8%B
3%D9%8A'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
2021-05-09 04:11:44 [scrapy.core.scraper] DEBUG: Scraped from <200 http://127.0.0.1:8000/blog/articles/>
{'url': 'http://127.0.0.1:8000/article/%D8%B9%D8%A7%D9%85-%D9%85%D9%86-%D8%A7%D9%84%D9%81%D9%82%D8%AF%D8%
A7%D9%86'}
```

Fig. 3 Result of crawled URLs after applying the solution.



```php
class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        $response->setContent($this->modifyArticleContent($response));

        return $response;
    }

    private function modifyArticleContent($response){
        $content = $response->getOriginalContent();
        $data = $content->getData();
        $post_body = $data['post']->post_body;
        $number_of_spans = random_int(5, 20);
        for ($i=0; $i <= $number_of_spans; $i++) {
            $position_of_span = random_int(0, strlen($post_body));
            $post_body = substr_replace($post_body, '<span class="hide">'.$this->
                generateRandomString().'</span>', strpos($post_body, " ", $position_of_span), 0
                );
        }
        $data['post']->post_body = $post_body;
        return $content;
    }

    private function generateRandomString($length = 10) {
        $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
        $charactersLength = strlen($characters);
        $randomString = '';
        for ($i = 0; $i < $length; $i++) {
            $randomString .= $characters[rand(0, $charactersLength - 1)];
        }
        return $randomString;
    }
}
```

Fig. 4 Middleware for modifying body

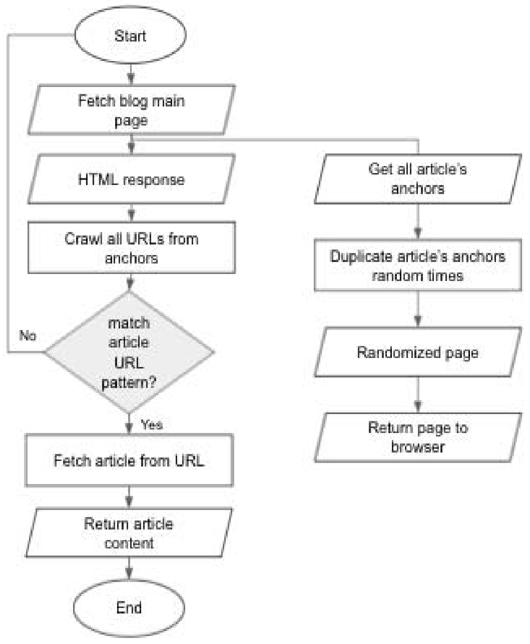


Fig. 5 Add duplicated anchors

The following DOM tree Fig. 6 describe the new added elements to the blog page, as the red rectangle show the duplicated articles' URL added in a random number within the articles list.

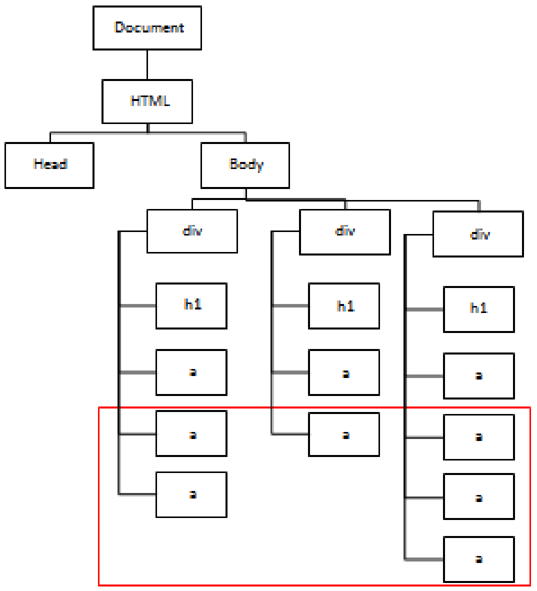


Fig. 6 Blog page DOM tree after applying the tchnique

To protect the content from being extracted and used on another website, we generate random characters among the article body, without affecting the appearance, the changes over the DOM tree of the article page, shown in a red rectangle in Fig. 7, a random number of span elements added within the article body.

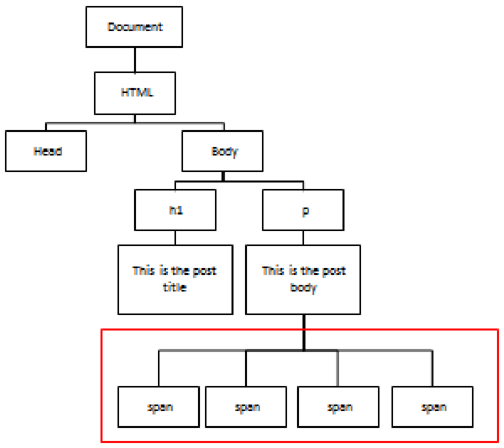
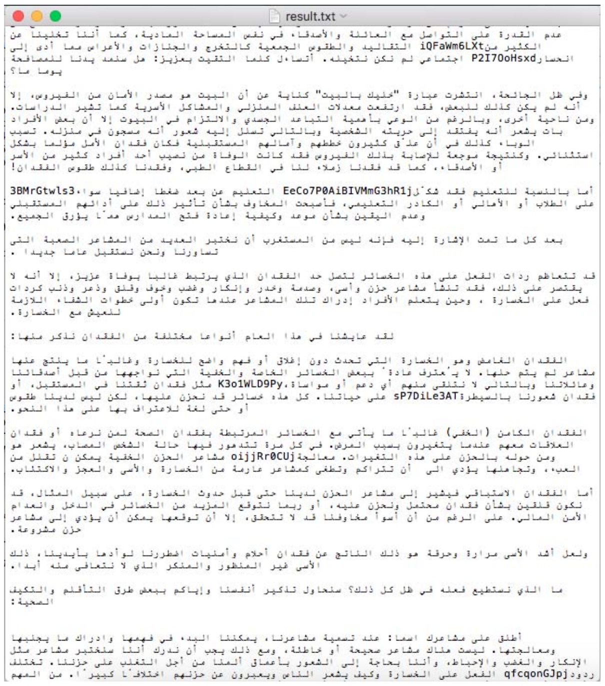


Fig. 7 Article page DOM tree after applying the technique

To apply this changes we developed a Middleware, shown in

Fig. 4, and use it on the article page only, it produced the results in Fig. 8. The techniques works as described in Fig. 9.



عدم القدرة على التواصل مع العائلة والأصدقاء في نفس المساحة المادية، كما أننا تخلينا عن الكثير منiQFaWm6LXt التقاليد والطقوس الجمعية كالتخرج والجنازات والأعراس مما أدى إلى انحسارP2I7OoHsxd اجتماعي لم نكن نتخيله. أتساءل كلما التقيت بعزيز: هل سنمد يدنا للمصافحة يوما ما؟

وفي ظل الجائحة، انتشرت عبارة "خليك بالبيت" كناية عن أن البيت هو مصدر الأمان من الفيروس، إلا أنه لم يكن كذلك للبعض، فقد ارتفعت معدلات العنف المنزلي والمشاكل الأسرية كما تشير الدراسات. ومن ناحية أخرى، وبالرغم من الوعي بأهمية التباعد الجسدي والالتزام في البيوت إلا أن بعض الأفراد بات يشعر أنه يفتقد إلى حريته الشخصية وبالتالي تسلل إليه شعور أنه مسجون في منزله. تسبب الوباء كذلك في أن علّق كثيرون خططهم وآمالهم المستقبلية فكان فقدان الأمل مؤلما بشكل استثنائي. وكنتيجة موجعة للإصابة بذلك الفيروس فقد كانت الوفاة من نصيب أحد أفراد كثير من الأسر أو الأصدقاء، كما قد فقدنا زملاء لنا في القطاع الطبي، وفقدنا كذلك طقوس الفقدان!

أما بالنسبة للتعليم فقد شكّلEeCo7P0AiBIVMmG3hR1j التعليم عن بعد ضغطا إضافيا سواء3BMrGtwls3 على الطلاب أو الأهالي أو الكادر التعليمي، فأصبحت المخاوف بشأن تأثير ذلك على أدائهم المستقبلي وعدم اليقين بشأن موعد وكيفية إعادة فتح المدارس همّا يؤرق الجميع.

بعد كل ما تمت الإشارة إليه فإنه ليس من المستغرب أن نختبر العديد من المشاعر الصعبة التي تساورنا ونحن نستقبل عاما جديدا .

قد تتعاظم ردات الفعل على هذه الخسائر لتصل حد الفقدان الذي يرتبط غالبا بوفاة عزيز، إلا أنه لا يقتصر على ذلك، فقد تنشأ مشاعر حزن وأسى، وصدمة وخدر وإنكار وغضب وخوف وقلق وذعر وذنب كردات فعل على الخسارة ، وحين يتعلم الأفراد إدراك تلك المشاعر عندها تكون أولى خطوات الشفاء اللازمة للتعيش مع الخسارة.

لقد عايشنا في هذا العام أنواعا مختلفة من الفقدان نذكر منها:

الفقدان الغامض وهو الخسارة التي تحدث دون إغلاق أو فهم واضح للخسارة وغالبًا ما ينتج عنها مشاعر لم يتم حلها. لا يُعترف عادةً ببعض الخسائر الخاصة والخفية التي نواجهها من قبل أصدقائنا وعائلاتنا وبالتالي لا نتلقى منهم أي دعم أو مواساة،K3o1WLD9Py مثل فقدان ثقتنا في المستقبل، أو فقدان شعورنا بالسيطرةsP7DiLe3AT على حياتنا. كل هذه خسائر قد نحزن عليها، لكن ليس لدينا طقوس أو حتى لغة للاعتراف بها على هذا النحو.

الفقدان الكامن (الخفي) غالبًا ما يأتي مع الخسائر المرتبطة بفقدان الصحة لمن نرعاه أو فقدان العلاقات معهم عندما يتغيرون بسبب المرض. في كل مرة تتدهور فيها حالة الشخص المصاب، يشعر هو ومن حوله بالحزن على هذه التغيرات. معالجةoijjRr0CUj مشاعر الحزن الخفية يمكن ن تقلل من العبء، وتجاهلها يؤدي الى أن تتراكم وتطغى كمشاعر عارمة من الخسارة والأسى والعجز والاكتئاب.

أما الفقدان الاستباقي فيشير إلى مشاعر الحزن لدينا حتى قبل حدوث الخسارة، على سبيل المثال، قد نكون قلقين بشأن فقدان محتمل ونحزن عليه، أو ربما نتوقع المزيد من الخسائر في الدخل وانعدام الأمن المالي. على الرغم من أن أسوأ مخاوفنا قد لا تتحقق، إلا أن توقعها يمكن أن يؤدي إلى مشاعر حزن مشروعة.

ولعل أشد الأسى مرارة وحرقة هو ذلك الناتج عن فقدان أحلام وأمنيات اضطررنا لوأدها بأيدينا، ذلك الأسى غير المنظور والمنكر الذي لا نتعافى منه أبدا.

ما الذي نستطيع فعله في ظل كل ذلك؟ سنحاول تذكير أنفسنا وإياكم ببعض طرق التأقلم والتكيف الصحية:

أطلق على مشاعرك اسما: عند تسمية مشاعرنا، يمكننا البدء في فهمها وادراك ما يجلبها ومعالجتها. ليست هناك مشاعر صحيحة أو خاطئة، ومع ذلك يجب أن ندرك أننا سنختبر مشاعر مثل الإنكار والغضب والإحباط، وأننا بحاجة إلى الشعور بأعماق ألمنا من أجل التغلب على حزننا. تختلف ردودqfcqonGJpj الفعل على الخسارة وكيف يشعر الناس ويعبرون عن حزنهم اختلافًا كبيرًا. من المهم

Fig. 8 Article body after applying the technique

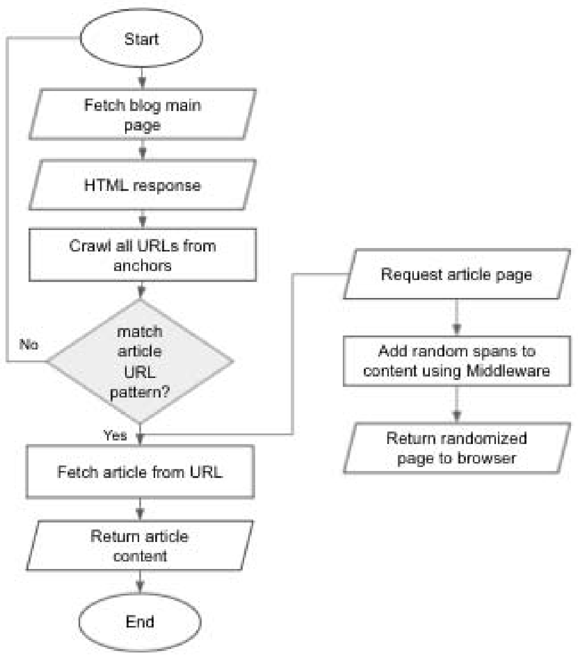

Start
Fetch blog main page
HTML response
Crawl all URLs from anchors
match article URL pattern?
No
Yes
Fetch article from URL
Return article content
End
Request article page
Add random spans to content using Middleware
Return randomized page to browser

Fig. 9 Add random spans within the article

IV. Experiments and Results

To evaluate the proposed solution effectiveness, we apply it to a website and re-scrape the web pages using the developed scraper and notice that the result matches the expectation. The proposed solution is not responsible for preventing scraping, but it makes the automatic scraping and extracting content to use it in another resource not feasible.

To assess the effect of the solution on web pages, we calculate the file size before and after, and we found that the size still approximately the same. And this thing logical because all the changes were on the next level, which does not affect the size very much as shown in TABLE I.

TABLE I. Table 1 The file size before and after the solution

File size (KB)	Before	After
Blog page	95	96
Article page	92	93

In addition, we calculate the processing time before and after applying the solution, and we found that the time was not affected too much, and still in the acceptance range as shown in . .TABLE II

TABLE II. Table 2 The processing time before and after the solution

Processing time (s)	Before	After
Blog page	1.95	2.23
Article page	2.27	2.42

For checking the web page similarity we use a python package called HTML-similarity to evaluate the structural and style similarity and found that the style did not affect, while the structure has a little bit effect as we see in TABLE . .III

TABLE III. Table 3 The similarity before and after the solution

Similarity (%)	Structural Similarity	Structural Similarity
Blog page	100	100
Article page	96	100

V. Conclusion

In this paper, the proposed solution for mitigating the bot scrapers from copying and reusing blogs content to keep it from unethical reuse, our technique applied on a local PHP project, and by using the Laravel framework we created a middleware to apply changes on the requested pages just before the response to the request. The experiment shows that the solution does not affect the efficiency and the appearance of the pages, while it gave very good results when we re-scraping the pages using the Scrapy tool. Therefore, we encourage using this technique due to the good results it returned.

For future work, we plan to continue the experiment to enhance the performance and reduce the processing time as possible, also, improve the solution by using honeypot anchors to keep tracking of bot behavior to be analyzed and classified using machine learning algorithms and block any potential attackers.

References

[1] B. Zhao, "Web Scraping," 2017, pp. 1–3.

[2] V. Krotov and L. Silva, *Legality and Ethics of Web Scraping*. 2018.

[3] E. Ferrara, P. De Meo, G. Fiumara, and R. Baumgartner, "Web Data Extraction, Applications and Techniques: A Survey," *ACM Comput. Surv. (Under Rev.*, vol. 70, Jul. 2012, doi: 10.1016/j.knosys.2014.07.007.

[4] M. Catalin and A. Cristian, "An efficient method in pre-processing phase of mining suspicious web crawlers," in *2017 21st International Conference on System Theory, Control and Computing, ICSTCC 2017*, Nov. 2017, pp. 272–277, doi: 10.1109/ICSTCC.2017.8107046.

[5] A. Haque and S. Singh, "Anti-Scraping Application Development," pp. 869–874, 2015.

[6] A. Diab and T. Barhoum, "Prevent XPath and CSS Based Scrapers by Using Markup Randomizer.," *Int. Arab. J. e Technol.*, vol. 5, no. 2, pp. 78–87, 2018.

[7] S. Wan, Y. Li, and K. Sun, "PathMarker: protecting web contents against inside crawlers," *Cybersecurity*, vol. 2, no. 1. 2019, doi: 10.1186/s42400-019-0023-1.

[8] E. B. Castañeda Rojas, "Propuesta de patrón de diseño de software orientado a prevenir la extracción automatizada de contenido web," *Pontif. Univ. Católica del Perú*, Nov. 2016, Accessed: Jul. 04, 2021. [Online]. Available: http://repositorio.pucp.edu.pe/index/handle/123456789/145777.