Automated Email Harvesting: Techniques, Risks and Security Implications

Mihai APOSTOL, Razvan DUȚESCU
National Institute for Research and Development in Informatics – ICI Bucharest
mihai.apostol@ici.ro, razvan.dutescu@ici.ro

Abstract: This study investigates the techniques and implications of web scraping for the extraction of email addresses from online sources. A Chrome web extension was developed, utilizing the Selenium testing framework to facilitate automated browsing and data collection. The targeted websites demonstrated minimal or inadequate defenses against web scraping, exposing significant vulnerabilities in their protection mechanisms. The relative ease of acquiring a substantial number of email addresses highlights the potential for phishing attacks, thereby raising critical security concerns. The article examines the technical aspects of web scraping, browser plugin development, and automated web application testing. Furthermore, it outlines best practices and strategies to defend against such data extraction methods, emphasizing the importance of robust web security measures to mitigate the risks associated with unauthorized data collection.

Keywords: web scraping, data gathering, automated testing, web browser plugins, web browser extension, email address scraping.

INTRODUCTION

The availability of data on the Internet and access to information certainly benefits society in all areas: e-books, software applications, open-source, Internet banking, shopping and many others. Every day new commercial companies are established with more or less possibilities, most of them with the aim of selling a product or a service. To sell a product or service requires advertising, especially at the beginning, the vast majority do not have an infrastructure available to maintain web applications necessary to carry out activities.

Listing products or services on different buy-sell sites, listing on platforms like Golden Pages or business listings are some of the immediate solutions to get things moving and not only that; The fact that a company or a company's services/ products are listed on multiple platforms brings more trust to that company. For example, if a company has its unique registration code (CUI) listed on the Internet, it can be checked with a specialized entity (for example the Trade Register)

https://doi.org/10.54851/v6i2y202402

if that company exists, has activity and other useful information including contact details.

Phishing cyber attacks involve sending malicious emails and electronic messages that contain links to web applications that mimic the web applications of legitimate companies. These attacks aim to obtain various access data, banking data and other information that would prove useful in illegal activities, fraud, blackmail and more (Jagatic, 2007).

Social engineering is a malicious activity performed by employing psychological manipulation techniques to trick victims into making security mistakes or revealing sensitive information (Sadiku, 2016).

Many companies, particularly newly established ones, often lack fundamental training in cybersecurity, rendering them vulnerable to phishing attacks and social engineering [Aurelian 2021]. Additionally, numerous websites that provide platforms for company listings fail to implement adequate protections for their data.

Data available online should be better protected from the point of view of its retrieval. Access to this data is generally free and available to anyone with very few restrictions that can be easily overcome. These data are generally protected by a form of authentication, confirmations such as the „I'm not a robot" checkbox, a captcha, or a click on a special button that reveals the requested information. These restrictions (except for authentication) were intended to block bots or web crawlers, but browser automation technology has evolved and it is not difficult for such a script to bypass these barriers.

This paper addresses techniques for collecting data available online, similar to the concept of web scraping, using a browser extension-type web application and web browser automation technologies. The data that will be extracted is data preferred by cybercriminals, namely email address, name, phone, and address/office.

MAIN CONCEPTS

In this chapter, we will explore three fundamental concepts relevant to web automation and data extraction: Web Scraping, Web Extensions, and Automated Testing. The section on Web Scraping will discuss various methods used to extract data from online sources, differentiating between manual and automated approaches, and highlighting the ethical concerns associated with its misuse. In the Web Extension segment, we will examine how browser plugins add or modify functionality on web pages, while also addressing the security vulnerabilities these extensions may introduce. Finally, the Automated Testing portion will cover the role of automation frameworks in web application testing, detailing their benefits, key features, and comparative capabilities across popular tools like Selenium, Cypress, Playwright, and Puppeteer.

Web Scraping

Web scraping, also called web data mining or web harvesting, is the process of building an agent or application that can automatically extract, analyze, download, and organize useful information from the web (Khder, 2021).

There are several methods of extracting data usually from databases, files and similar sources. Where data is available online, there are several methods by which this data can be extracted. One method is to search and save the data manually in an XLS file, but it is inefficient and time-consuming. Another method is to write or use a web scraping application.

The terms Web Crawling and Web Scraping are often used interchangeably because they both refer to the concept of data mining, but there are differences between the two terms. The differences between the two can be seen in Table 1.

Table 1. *Web Crawling and Web Scraping differences*

Property	Web Crawling	Web Scraping
Data extraction	Extracts all the content of a large number of websites	Extracts individual content from a website specific to the site structure
Implementation scale	Big	Any scale
Information type	Generic	Specific

Cybercriminals also use these web scrapers, as they have the ability to extract enormous amounts of information from the Internet such as phone numbers, email addresses, residential addresses and many other types of data that can be used to initiate phishing attacks and social engineering.

Web extension

A web extension refers to a plug-in for web browsers and is a software program (or software module) that applies various filters, adds functionality and controls the way web pages are displayed. They are developed using web technologies such as HTML, CSS, JavaScript and JavaScript APIs (Mehta, 2016).

Web extensions generally have the role of adding additional features and functionality to a website such as translating text from one language to another, downloading content in a specific format, recording certain activities, and more. Extensions can also be used to remove unwanted features and functionality such as ads, alert windows, pop-up ads and other aspects that the user wants removed.

Some extensions can be malicious in nature and although most of them come from unofficial sources, some still manage to reach official virtual stores like Google Chrome Web Store. For example, in 2018, four malicious extensions were discovered. The extensions were meant to create sticky notes, but in reality, these extensions were part of a click fraud scam, where each click generated revenue from rewards (Goodin, 2018).

Browser extensions are an interesting target for scammers because many extensions have massive user bases and communities and they are automatically updated. This means that if a user has downloaded a harmless extension, it can be updated later to become malicious; this update will be sent immediately to the user without them noticing anything. A similar scenario was possible due to phishing attacks targeting developer accounts of some web extensions. These accounts were compromised and extensions were updated with malicious packages that reached over a million users (Goodin, 2017).

Automated testing

Automated testing of web applications is a process in which various tools and software tools are used to evaluate security, behavior, performance, existing bugs and other elements that may affect a final web application (Monica, 2014). Automation is the most effective way to increase efficiency and runtime in web application testing for several reasons (Smada, 2018):

- Manual testing is expensive and takes a lot of time;
- Reduces human error;
- It is difficult to test web applications that are available in several languages;
- They do not require human intervention, they can be done unattended.

Automated testing of a web application involves entering data into the fields, completing application forms, extracting and downloading page content, and actions like click on a button, anchor or similar elements.

There are numerous automation frameworks available, many of which serve the same fundamental purpose, but they differ in aspects such as supported languages, method of invocation, and the implementation of certain functions.

Cypress is a free and open-source framework that can be installed locally, enabling users to write, run, debug, and record automated tests. It offers two download options: a direct download or via npm. The direct download lacks the test recording service and is intended for users wanting a quick trial. As a JavaScript framework, Cypress supports Chrome, Firefox, and Edge browsers, and is executed directly in the system browser. Its synchronous methods and ability to handle network traffic are among its key features (Cypress, 2024).

Selenium is the most widely used automated web application testing framework, also free and open-source. It supports multiple programming languages, including Java, Python, C#, Perl, and Ruby, and is compatible with Windows, Linux, and macOS. Selenium consists of three components: Selenium IDE, a Chrome and Firefox plugin for recording, editing, and debugging tests; Selenium Grid, a server that enables the execution of tests in parallel across multiple machines and browsers remotely; and Selenium WebDriver, which evolved from Selenium Remote Control to address previous limitations. WebDriver supports Python, Ruby, C#, and Java, but requires a specialized browser driver, such as Firefox Driver, Chrome Driver, or Safari Driver. It cannot handle window elements directly, though tools like Sikuli address this limitation (Selenium, 2024).

Playwright, an open-source framework developed by Microsoft, supports Chrome, Firefox, and WebKit browsers. It is compatible with Windows, Linux, and macOS, and supports Python, Java, and C#. Playwright simplifies test creation by eliminating the need for wait functions, as it has a built-in wait mechanism. Its key capabilities include intercepting network activity, simulating mobile devices, generating mouse and keyboard events, uploading and downloading files, and running parallel tests (Playwright, 2024).

Puppeteer is a library designed for Node.js, providing control over Chrome and Firefox browsers. It can automate various tasks typically performed manually, such as generating screenshots, auto-filling forms, and testing web extensions. Puppeteer operates using JavaScript and its derivatives, such as TypeScript. One of its notable features is that the Node.js installation includes a version of Chrome optimized for use with Puppeteer. Additionally, its synchronous methods and parallel testing capabilities make it a powerful tool for web automation (Puppeteer, 2024).

The most notable differences between these frameworks are structured in Table 2, after their most important capabilities.

Table 2. *Automated framework differences*

	Cypress	**Selenium**	**Playwright**	**Puppeteer**
Supported browsers	Chrome, Firefox, Edge	Chrome, Firefox, Internet Explorer, Edge, Safari	Chrome, Firefox	Chrome, Firefox
Programming languages	JavaScript	Python, C#, Java, Ruby, JavaScript, Kotlin	Python, C#, Java	JavaScript
Multiple browser instance	No	Yes	Yes	Yes
Multiple tabs	No	Yes	Yes	Yes
Independent driver	No	Yes	No	No
Execution speed	Fast	Slow	Fast	Fast
Parallel testing	Yes	Yes	Yes	Yes
Automated waiting	Yes	No	Yes	Yes
Install method	Local, npm	Local, pip	npm	npm

Although the Selenium framework has lower script execution times and the absence of automatic waiting (by default, the script will have more lines of code), it offers greater freedom and a much wider field of testing possibilities. It is the most used automated testing framework, it benefits from a huge community and complete documentation.

PREVIOUS WORK - WEB SCRAPING TECHNIQUES

There are numerous techniques to do web scraping on web pages (Lotfi, 2024). The most popular one is HTML parsing which involves extracting data from an HTML structure. The popularity of this method is derived from the straightforwardness of the approach as well as the number of libraries such as BeautifulSoup in Python or Jsoup in Java (Morina, 2022).

Another popular technique for web scraping is using a regular expression(regex) to identify patterns and extract text from within the HTML structure. This method has one advantage over the others in that most programming languages have a built-in library for them. Examples include the 're' library in Python, 'java.util.regex' for Java or 'Regexp' for Ruby. Using a regular expression provides flexibility in handling different text patterns but it does require knowledge of regex patterns and can be harder to maintain with increasingly more complex patterns (Gunawan, 2019).

An email extractor (Ketan et al., 2023) is a type of software used to extract email addresses from HTML pages, resulting in a huge list of email addresses. In Ketan's paper, an email extractor based on AI was proposed. Natural language processing algorithms can be used to extract emails by analyzing the text and identifying patterns that match the structure of an email address. For the data extraction, the Python packages beautifulSoup and requests are used. The Python pandas library is used for extracting and saving the data in a specific format. This scraper is also capable of validating (checking for the email's existence) and sending bulk emails. Unfortunately, due to sending email functionality, the tool can be used for malicious intentions like spamming and phishing emails.

Besides traditional web scraping techniques, web data extraction can be achieved using AI algorithms. In (Patnaik et al, 2021) paper, a web data extraction system is proposed, using deep learning, you only look once (YOLO) algorithm and Tesseract LSTM (long short-term memory) to extract product details, which are detected as images from web pages. This system solves the problem of having a core data extraction engine and can adapt to dynamic changes in website layout. This system has an image detection precision of 97% and a character extraction precision of 99%. It is built for gathering product data from e-commerce websites (Amazon for example) but this can be used for extracting any kind of data, including email addresses.

ParseHub is a popular application designed for web cramping. It has the facility to point and click on a web page to extract the desired information. Data can be downloaded in CSV or JSON format. It can recognize patterns and manipulate text using regular expressions. The application provides cloud hosting and can schedule data collections (ParseHub, 2024).

Octoparse Is a free software application for Windows (XP and above) that transforms unstructured or semi-structured data from websites into a structured dataset without the need to write code. It can download CVS, Excel, HTML, TXT or databases (MySQL, SQLServer, Oracle) format. Simulates web browsing behavior such as opening a web page, logging into an account, entering text, pointing and clicking on web elements, etc (Octoparse, 2024).

Importo.io is a web data integration SaaS (software as a service) platform that enables the conversion of unstructured web data into a structured format by extracting, preparing and integrating web data for consumption in analytical platforms or used in business, sales or marketing applications. Provides a visual environment for automating the web data extraction and transformation workflow. After specifying the URL of the target website, the web data extraction module provides a visual environment for designing automated data collection workflows, going beyond HTML parsing of static content to automate end-user interactions, providing data that would otherwise not be immediately visible. Once extracted, the software provides comprehensive data preparation capabilities that are used to harmonize and web data cleaning and provides a library of Excel-like functions that allow the end user to build custom formulas that can be used to enrich the dataset. It has its own visualization and dashboard to help business analysts get the insights they need and also provides APIs that provide full access to everything that can be done on their platform, allowing direct integration of web data into your applications (Import.io, 2024).

Webhose.io API provides easy-to-integrate, high-quality data and metadata from hundreds of thousands of online sources, forums, blogs, reviews, news, and more. Available via either query-based API or firehose, the Webhose.io API provides low latency with high-coverage data, with an efficient dynamic ability to add new sources.

Webhose's machine-readable data streams are simple, easy to integrate, and its granular filtering capabilities and scalable data plans are currently used by hundreds of organizations around the world (Webhose.io, 2024).

There are plenty of web crawler extensions for all types of browsers. They have fewer capabilities compared to an application run by an operating system but meet the basic Chrome requirements, for example, Scraper Crawler V3 is available (demo version) in the Chrome web store and is easy to use.

EXPERIMENTAL SETUP

The following experiment illustrates the ease with which a substantial amount of information, including email addresses and telephone numbers, can be obtained for potential phishing attacks through the use of automated web testing frameworks. A Chrome plugin was developed and integrated with Selenium to automate the collection of such data, highlighting the vulnerabilities in web security that can be exploited by malicious actors.

The experimental setup has two elements: a web browser plugin and a web testing automation script.

The browser plugin is compatible with the Chrome web browser. It uses Google storage API for data storage and has the advantage of cloud storage so the gathered data can be accessed from any device. It is written using Javascript and jQuery. The plugin is responsible for the data gathering from the webpage.

Figure 1 shows the browser extension with JSON results.

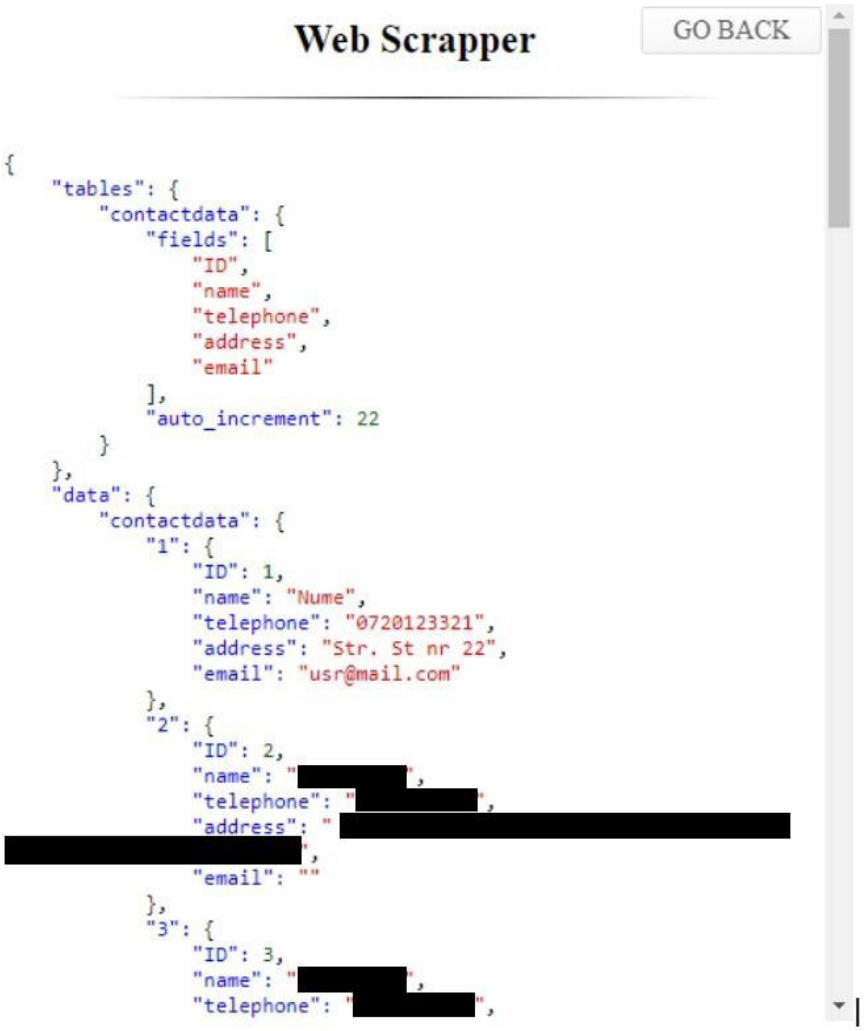

Figure 1. *Results in JSON format*

After a page has loaded, the extension follows several steps:

- checks if a database exists, if no database is found, one is created and a default line is inserted;
- get all tags from the page;
- checks in all tags for elements that have the "itemprop"' attribute;
- checks if "itemprop"' has the values „name", „email", „telephone" or „address";
- sanitize the values for unwanted characters like \n, new lines, commas (commas must be removed due to CSV delimiter) etc.;
- checks if the entry is already in the database by phone number, if not in the database, it creates a new entry;
- results are returned in JSON format but also it can be exported as CSV.

The web browser automation script is written in Python using the Selenium library. This library comes with its own Chrome web browser named Chrome webDriver. In order to use the above web browser extension, the plugin must be first packed to a .crx file.

The automation script follows several rules for navigation:

- browser launching;
- wait for the page to be fully loaded and then accept the cookies policy. This is located in an iframe element so the function switch_to.frame() was used;
- a base URL and a list of keywords are defined and used for the search;
- because the search is based on GET requests, we can concatenate the URL with a keyword and access this new URL. We do this for every keyword in the list and while a "next" button is present on the page, click on it and access the next page.

While the selenium script is running, the plug-in gathers all the data from every page. When the automation script has ended its execution, it automatically clicks on the generated download button containing the results in the .csv file. Without the automation script, the browser plugin requires manual browsing.

RESULTS

We have tested the automation script using 5 keywords. If we manually navigate to that page and perform manual searches, we notice that there is a maximum of 20 results per page. All results returned have the full name, phone number and address, but not all have the email address available.

```
======= AUTOMATION ENDED =======
======= TIME ELAPSED ======= 73.19373369216919
======= Pages vizited ======= 45
========================================
```

Figure 2. *Automation script results*

The script's navigation time is approximately 1.6 pages/second, which means that around 32 entities data are collected per second as seen in Figure 2.

There are some keywords that return even hundreds of pages of results. If we let this automation script run for 24 hours, which is 86400 seconds, we will end up with about 2764800 entries. Even if only 10% of them have a declared email address, 250000 valid emails are still a huge number.

The downside of using the Selenium framework is that it cannot properly handle certain load times. This problem generates a consequence that has a non-deterministic element: When the script is started, it may or may not return an error at the moment when the search is made to detect the cookie module confirmation button.

In comparison to the results from related work, the experiment presented here demonstrates a similar level of efficiency in data extraction but differs significantly in its focus and implementation. For instance, the AI-based email extractor proposed by Ketan et al. (2023) leverages natural language processing (NLP) algorithms to extract and validate email addresses with high accuracy, while also offering bulk email-sending functionality. In contrast, the experiment described in this study emphasizes the use of a browser plugin integrated with Selenium to automate the collection of both email addresses and other personal information such as phone numbers and addresses. While Ketan's method is more advanced in terms of AI integration and email validation, the experiment here demonstrates the sheer volume of data that can be extracted through a more straightforward automation approach, achieving a collection rate of 32 entities per second.

Similarly, Patnaik et al. (2021) utilized deep learning techniques for web data extraction,

targeting dynamic content like product images from e-commerce platforms. While their method, powered by the YOLO algorithm, excels in handling visual data and adapting to changing layouts, the approach in this paper focuses on structured HTML elements using simpler but highly efficient techniques. Patnaik's system achieves impressive accuracy (97% image detection precision), but the browser plugin and automation script in this study have the advantage of high-speed extraction over an extended period. The experiment shows that within 24 hours, approximately 2.76 million entries can be gathered, even if only a fraction includes email addresses, underscoring the potential scale of phishing attacks. However, limitations such as handling page load times and occasional errors with Selenium highlight the trade-offs between speed and reliability compared to more sophisticated AI-driven methods.

CONCLUSIONS

The recommendations outlined in this section are derived from a combination of insights gained through experimental results, practical experience with automated web frameworks, and an analysis of findings from related literature. This integrative approach ensures that the proposed measures are well-informed and address both practical and theoretical considerations in the context of web security and automation.

Unlike other online data collection methods, our method is more secure and the risk of the target platform restricting access is lower because it cannot detect that an automation script is running. Accessing using a browser requires time to load and process documents until the next document needs to be downloaded. This time, which is really wasted, is capitalized by the fact that other methods download more documents in a much shorter time and this can be detected by the target application.

Web extensions are hard to block, especially if they are not present on official extension platforms. An extension can be detected if the web extension ID is known. Once detected, several solutions can discourage the use of extensions on that web application:

- The use of means such as alerts, which can be configured so that after pressing the confirmation button that closes the alert window, a new window appears immediately, all with the same error message/request to disable the extension;
- Redirection to another page;
- Access restriction;
- Analyzing the DOM, detecting script insertions and blocking them.

Protecting web applications against scripts is generally difficult to achieve, especially since some developers do not consider that the data that will be available on the application they are developing will be collected, perhaps fully and easily. From the point of view of the web application development process that has the specific profile of this problem, the following solutions and mitigations can be considered:

- Using third-party software such as Barracuda Web Application Firewall. The Barracuda Web Application Firewall embeds a hidden link in the response. The embedded link is not displayed on the browser, so a human browsing the web pages through a common browser should never see and click the hidden link. Therefore, any request that attempts to access the hidden link is identified as an automated bot or scraper. (BarracudaCampus, 2024)
- Development of frameworks that can modify the tag properties of a document (id, class, itemprop, etc.) at each request. For example, changing the name of a class with a random string. These frameworks would negatively impact performance but offer very good protection against scripts;
- Implementation of captcha modules. They require data from the user (clicking on various images, solving a small puzzle, inserting a text, etc.) or confirmation from the user that often their windows

intentionally load later. These are hard to detect by automation applications and most frameworks have synchronous methods, which makes waiting time hard to control. Modules of the „I'm not a robot" type are generally easily solved because they require at most two clicks;

- Contact data should benefit from a minimum of protection against computer robots. They should only be accessed by registered users with a confirmed account on that platform;
- Monitoring network DNS traffic and blocking IPs that flood servers with requests in a very short time (Choi, 2007);
- Development of scripts that can analyze the server logs. If it is not possible to monitor the network traffic, the logs record every request received;
- Strengthening network security;
- Hiding the source code using various technologies that have the possibility of compilation and/or obfuscation.

It is noticeable how easily tens of thousands of email addresses can be obtained and the enormous amount of data that can be obtained in just a few days with minimal resources. Unfortunately, most phishing campaigns are successful because users do not carefully check links in email messages, the sender's email address, attached files and many other aspects. Anyone who interacts with computers, especially in the workplace where the consequences can be much more serious, needs to have a minimum knowledge of cyber security. Companies can take measures and organize training sessions dedicated to cyber security, participation in dedicated courses or other similar activities.

While automated web testing frameworks such as Selenium and Puppeteer are invaluable for legitimate purposes like website testing and development, they are equally prone to misuse for malicious activities, such as unauthorized web scraping and data harvesting. The versatility and accessibility of these tools make it challenging to effectively prevent their exploitation, as they can closely mimic human behaviour and bypass common security measures. Despite the implementation of protective strategies like bot detection and rate limiting, fully mitigating the risks posed by the malicious use of these frameworks remains difficult without affecting their legitimate applications.

REFERENCE LIST

BarracudaCampus. (2024) *Documentation, Training, and Certification for Your Barracuda Networks product.* https://campus.barracuda.com/download/pdf/article/48660515 [Accessed July 2024].

Choi, H., Lee, H., Lee, H. & Kim, H. (2007) Botnet Detection by Monitoring Group Activities in DNS Traffic, *7th IEEE International Conference on Computer and Information Technology*. doi: 10.1109/cit.2007.90. [Accessed June 2024].

Chrome (2024) *Puppeteer.* Chrome for Developers. https://developers.google.com/web/tools/puppeteer [Accessed July 2024].

Cypress. (2024) *Introduction to Cypress.* Cypress docs. https://docs.cypress.io/guides/core-concepts/introduction-to-cypress [Accessed July 2024].

Goodin, D. (17 January 2018) Google Chrome extensions with 500,000 downloads found to be malicious. *Ars Technica*. https://arstechnica.com/information-technology/2018/01/500000-chrome-users-fall-prey-to-malicious-extensions-in-google-web-store/ [Accessed June 2024].

Goodin, D. (3 August 2017) After phishing attacks, Chrome extensions push adware to millions. *Ars Technica* . https://arstechnica.com/information-technology/2017/08/after-phishing-attacks-chrome-extensions-push-adware-to-millions/ [Accessed June 2024].

Gunawan, R., Rahmatulloh, A., Darmawan, I. & Firdaus, F. (2019) Comparison of web scraping techniques: Regular expression, HTML dom and xpath. *Proceedings of the 2018 International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018)*. doi:10.2991/icoiese-18.2019.50 [Accessed June 2024].

Import (2024) *Welcome!* https://docs.import.io/ [Accessed July 2024].

Jagatic, T. N., Johnson, N. A., Jakobsson, M. & Menczer, F. (2007) Social phishing. *Communications of the ACM*. 50(10), 94-100. doi: 10.1145/1290958.1290968. https://www.researchgate.net/publication/220424040_Social_phishing [Accessed May 2024].

Nimase, K., Thakur, S. and Gaonkar, S. (2023) AI based e-mail scraper and sending tool. *International Journal for Research in Applied Science and Engineering Technology*, 11(4), pp. 3749–3755. doi:10.22214/ijraset.2023.50679 [Accessed May 2024].

Khder, M. (2021) Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. I*nternational Journal of Advances in Soft Computing and its Applications*, 13(3), pp.145–168. doi: https://doi.org/10.15849/ijasca.211128.11 [Accessed August 2024].

Lotfi, C., Srinivasan, S., Ertz, M. & Latrous, I. (2021) Web scraping techniques and applications: A literature review. *SCRS CONFERENCE PROCEEDINGS ON INTELLIGENT SYSTEMS*, pp. 381–394. doi:10.52458/978-93-91842-08-6-38. https://www.researchgate.net/publication/367719780_Web_Scraping_Techniques_and_Applications_A_Literature_Review [Accessed June 2024].

Mehta, P. (2016) Creating Google Chrome Extensions. *Berkeley, CA: Apress*. doi:https://doi.org/10.1007/978-1-4842-1775-7 [Accessed June 2024].

Morina, V. & Sejdiu, S. (2022) *Evaluating and comparing web scraping tools and techniques for data collection.* https://www.researchgate.net/publication/371141292_The_Benefit_and_Risks_for_Scraping_Based_on_Python [Accessed June 2024].

Octoparse (2024) https://dataapi.octoparse.com/DataApi/en-US/ [Accessed July 2024].

ParseHub (2024) Documentation. *ParseHub Help Center*. https://help.parsehub.com/hc/en-us/categories/203678627-Documentation [Accessed July 2024].

Patnaik, S.K., Babu, C.N. and Bhave, M. (2021) Intelligent and adaptive web data extraction system using convolutional and long short-term memory deep learning networks. *Big Data Mining and Analytics*, 4(4), pp.279–297. doi:https://doi.org/10.26599/bdma.2021.9020012 [Accessed May 2024].

Playwright (2024) *Installation.* Getting Started. Playwright. https://playwright.dev/docs/intro [Accessed July 2024].

Selenium (2024) *The Selenium Browser Automation Project.* Selenium.dev. Available at https://www.selenium.dev/documentation/en/ [Accessed July 2024].

Smada, D.M., Rotuna, C., Boncea, R. and Petre, I. (2018). Automated Code Testing System for Bug Prevention in Web-based User Interfaces. *Informatica Economica*, 22(3/2018), pp.23–32. doi:https://doi.org/10.12948/issn14531305/22.3.2018.03 [Accessed July 2024].

Stoica, A. (2021) Social engineering as the new deception game, *Romanian Journal of Information Technology and Automatic Control*. 31(3), pp. 57-68. ISSN 1220-1758. https://doi.org/10.33436/v31i3y202105 [Accessed August 2024].

Sharma M. & Angmo R. (2014) Web based Automation Testing and Tools, *International Journal of Computer Science and Information Technologies*. 5 (1), pp. 908-912. https://www.academia.edu/35698956/Web_based_Automation_Testing_and_Tools [Accessed June 2024].

Sadiku, M., Shadare, A. & Musa, S. (2016) Social Engineering: An Introducction. *The Journal of Scientific and Engineering Research*. 3, 64-66. Available at https://www.researchgate.net/publication/308315268_Social_Engineering_An_Introducction [Accessed April 2024].

Webz.io. (2024) *About Webz.io*. [online] https://docs.webz.io/reference/about-webzio [Accessed July 2024].



This is an open access article distributed under the terms and conditions of the Creative Commons Attribution-NonCommercial 4.0 International License.