Towards an automatic tool for detecting third-party data leaks on websites

Robin Carlsson[1], Panu Puhtila[1] and Sampsa Rauti[1,*]

[1]University of Turku, Finland

Abstract

Everyday tasks are increasingly completed with the help of various web-based services, and many users with little technical know-how are using these services. Due to this development, online privacy has emerged as a paramount concern when developing web services. One particular privacy concern involves third-party services such as analytics services that are nowadays commonplace on almost any website. In the current study, we explore the possibilities of automating the data collection in scientific research on personal data leaks related to third-party analytics tools, and build a proof-of-concept implementation of a tool that uses automated traffic analysis to record and analyze potential leaks of personal data to third-party services. The current implementation of the tool is intended to detect URL leaks, and to specifically inspect how this happens in the search functionalities found on the analyzed websites. Our findings indicate that the automation of this kind of data collection is very effective, and could potentially increase the quality of the research significantly as it allows for faster and more wide-spread data collection.

Keywords

Data leaks, online privacy, web security, robotic process automation

1. Introduction

With the rapid advancement of digitalization, reliance on electronic services for everyday tasks keeps growing. Web technologies in particular have become popular due to their accessibility, scalability and ease of installation and maintenance. At the same time, there are various essential web services which involve processing sensitive personal data, posing a risk of data disclosure to third-party entities like analytics services. To address privacy concerns, the General Data Protection Regulation (GDPR) has been established. The GDPR regulates the processing of personal data and provides individuals with greater control over their sensitive information. As per the requirements of the GDPR, users must always be properly informed about personal data collection.

Many users are likely aware of analytics services embedded into websites to some extent, but often they do not fully understand what kind of personal, potentially very sensitive data is sent to third-party services when visiting websites. Privacy policy documents that are supposed to

SQAMIA 2023: Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, September 10–13, 2023, Bratislava, Slovakia

*Corresponding author.

crcarl@utu.fi (R. Carlsson); papuht@utu.fi (P. Puhtila); sjprau@utu.fi (S. Rauti)

0009-0003-7255-0239 (R. Carlsson); 0009-0004-6418-1063 (P. Puhtila); 0000-0002-1891-2353 (S. Rauti)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3588/p12.pdf

shed light on the nature of collected personal data regularly use vague and unclear expressions, often failing to inform users correctly [1, 2]. Many web service developers also seem to be oblivious to dangers of using excessive amounts of third-party services on their websites. The situation calls for increased attention to the privacy of web services and implementation of effective tools to assess potential data leaks.

This paper describes an automated tool for detecting data leaks on web pages[1], and discusses both its potential uses and further development possibilities in the future. The motive behind the development of this tool has risen out of the needs of the IDA[2] research project. As a part of the project, we have conducted several studies on third-party data leaks and their connection to user-given consent to data collection, or the lack of such, happening on numerous popular websites. In such studies, the acquisition of datasets for analysis is often slow and tedious, involving lots of manual and repetitive work. Large portions of this labor could be automated to allow for the collection of larger datasets in the same timeframe and make the data collection process more systematic, enhancing the quality of data.

The rest of the paper is structured as follows. In Section 2, we present a brief survey of the related research and development that precedes our own work. Section 3 gives an overview of third party services, mainly web analytics tools, the reasons why they are used at the websites and how this impacts user privacy. In Section 4, the conceptual design of the data leak detection algorithm is explained and the details of the actual implementation of the tool are laid bare. In Section 5, we take a look at the potential future challenges and possibilities related to the development process. Finally, in Section 6, we present our definitive conclusions.

2. Third-Party Services and Online Privacy

2.1. Data Collection by Third-Party Services

There has been a shift towards digital platforms and online business models over the past few decades, accelerating significantly in recent years. Organizations have turned to web analytics to gain insights about their customers, optimize operations and make data-driven decisions. Today, various different third-party analytics services are embedded into websites in order to analyze behavior of users, measure performance of websites and provide demographic information about website visitors [3, 4].

One important use of web analytics is conversion tracking [5, 6]. In online marketing, conversion refers to a desired action taken by a website visitor, and represents a successful outcome of a marketing strategy or the desired engagement with a website – something that adds value to the company. With web analytics, websites can set conversion goals and track user actions that indicate successful conversions. Examples include successfully persuading a user to submit a form or make a purchase.

Many noncommercial pages use the same third-party analytics services and track "conversions" in the same manner, although the term conversion takes a more generic meaning here. As Bekos et al. note, "actions the website has configured to be tracked are defined as

[1]The tool was developed by the first author in 2023 and is available on Gitlab at https://gitlab.utu.fi/crcarl/ida-selenium-search-bar-tool

[2]Intimacy in Data-Driven Culture (https://www.dataintimacy.fi/en/)

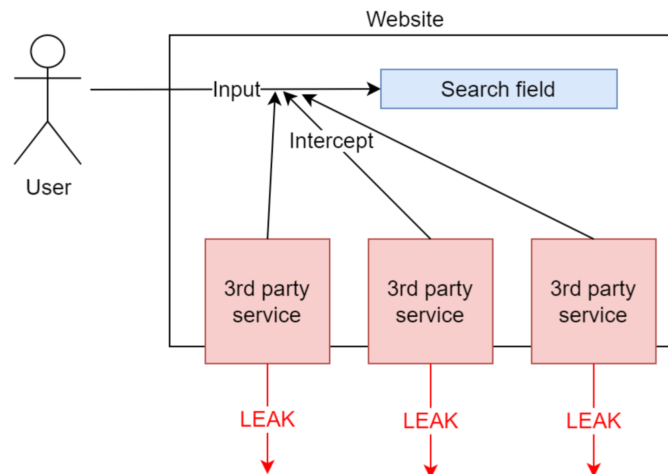

Figure 1: Data collection by third-party services.

conversions." [7] Understood in this sense, using the search function can also be considered a conversion. The visits to individual pages are also often tracked. They can be used as a part of the process called funneling, which refers to tracking and analyzing the steps that users take on a website to achieve a certain goal or conversion. Landing on an important page can be considered a conversion as well.

The fact that these actions are being tracked, often by several analytics services, naturally causes privacy concerns. Sensitive searches, visits on delicate web pages, and several private actions (say, purchasing a specific prescription medicine in an online pharmacy) leak to third parties. Figure 1 depicts this situation. Third-party analytics services, shown as red boxes, capture identifying information on the user (such as IP addresses and device identifiers) as well as previously mentioned contextual data on the user's visit and actions on the website. The collected data is then sent to external servers of analytics companies, such as Google or Meta. In a way, this setting resembles a man-in-the-browser attack [8], in which a third party stealthily spies on network traffic without the user noticing anything. Usually, the web developers and the organizations operating websites do not fully understand that such serious data leaks are taking place in their services.

2.2. An example illustrating the need for automatic data leak detection in privacy research

The challenge in researching personal data leaks described previously in larger scale is that suitable automatic tools for this purpose seem to be missing. To illustrate the need for automatic data leak detection in more detail, we shall take a look at a previous study conducted on the personal data leaks in online pharmacies operating in Finland [9]. In this paper, it was observed that 70% of the studied pharmacies leaked sensitive information to third parties, and 35% leaked highly sensitive health related data such as the specific prescription medicine the customer is ordering. The data leaks happened mostly through analytics tools, for example Google Analytics,

deployed on the pharmacy websites for the exact purposes described earlier.

The data collection for the study was conducted as follows: first, a researcher navigated to the selected online pharmacy website, opened the Google Chrome Developer Tools (devtools from this point onwards) and cleared the cache, after which the page was reloaded. Then the devtools were made to record all traffic that happened between the online pharmacy website and any third parties. Then the researcher proceeded to make a dummy purchase: they searched for a given prescription medicine, followed the links given by the sites' search function to the specific medicines page, added it to their cart and proceeded to the check-out. No actual purchases were made in the experiments, so the data trail ends at the point where the customer would have to have the law-mandated chat with an online pharmacist before the medicine purchase.

During the test sequence explained above, on every step of the way, sensitive personal data could potentially be leaked to third parties. In the tests the search term (usually a medicine name) was regularly leaked. Data on visiting the product page of the given medicine was leaked. Moreover, data on actually initializing an order for a specific prescription medicine was leaked, even without making any actual purchase.

This example illustrates both the cumbersome way this kind of research data is gathered and the need for this kind of research in the first place. User anonymity becoming compromised is an ongoing and widespread issue, and one which happens all too much hidden from the public scrutiny. Automating large parts of the procedure of detecting such behavior in websites allows researchers to conduct essentially similar surveys on much larger scope in much smaller time, yielding better statistical information on how prevalent these kinds of data leakages are.

3. Related Work

There are other applications designed to perform functions similar to our tool, such as Website Evidence Collector (WEC), which was developed at the behest of the European Data Protection Supervisor (EDPS)[3]. Another somewhat similar tool was developed and used in research by Wesselkamp et al.[10] as a browser extension. The big difference to what our application or the WEC are doing, compared to the ERNIE browser extension, is that ERNIE is ultimately meant to be used by the proprietors of the websites to help them understand how tracking happens in their own domains, so as to better comply with data privacy regulations. We have used the WEC in previous studies [9], but found it wanting in several aspects. For one, WEC alone can not process multiple websites with one command. Although there is a third-party distribution of it available which can do this[4], inability to analyze several pages was not the only shortcoming which led to the development of a tool of our own. The primary feature our research needed, and WEC lacked, was the ability to automate the interaction with the website being inspected, which is the whole idea of the tool discussed in this paper. Another important reason was that while it is possible to make WEC move from one page to the next, the network traffic taking place on individual pages is not differentiated with enough clarity in the report it generates.

Another roughly similar application is OpenWPM (abbreviated from Open Web Privacy

[3]https://edps.europa.eu/edps-inspection-software_en
[4]https://github.com/ovh/website-evidence-collector-batch/blob/master/src/index.js

Measurement) which was developed by the Englehardt as a part of his doctoral thesis [11] at the Princeton University. The tool he devised is, like ours, based on the idea of automatically processing through websites and detecting the user tracking systems that are being used. However, OpenWPM is more concerned with categorizing the different forms of tracking technologies used, especially the different forms of fingerprinting techniques that are used in tracking the user, unlike our tool which is aimed at identifying the actual data leaks that happen in the websites, and specifically in their search functionalities. Englehardt was inspired in his research by earlier similar tools such as FPDetective, which was developed by Acar et al. [12]. FPDetector is apparently quite similar application to the later OpenWPM, although perhaps slightly more limited in scope and modularity. Another somewhat similar application is TrackingObserver [5], developed by the University of Washington. It is a modifiable Google Chrome extension designed to automatically detect user tracking. It is designed to be customizable, and exposes several APIs for tracking detection, measurement, and blocking. The tool can be further modified with installable add-ons.

4. Tool Design & Implementation

4.1. Algorithm Design

The algorithm implemented by this tool is designed to process a list of websites, inspect each page in the list for certain elements (div tags, input fields and interactable elements which have the term "search" in their HTML tag attributes), add found elements to a list, then loop through this list and attempt to interact with each element. If the element is an input field, this means inputting a pre-determined search term. If the element is clickable, it is clicked. If the interaction succeeds, the algorithm records all traffic between the website and third parties, then moves to the next website on the list. This process is presented in detail in the state diagram for the algorithm, shown in Figure 2. The algorithm is presented below in the pseudo-code notation.

```
INITIALIZE web browser
READ list I from input file

FOR each website URL in list I:

    # Search for input/interactive elements with specific attributes
   IF input/interactive element with id/name/class indicating search EXISTS:
        APPEND element to list A/B
    FOR each div element:
            IF div with class/id/name indicating search EXISTS:
                FOR each child of div:
                    IF child is input/interactive element:
                        APPEND child to list A/B
                    ELSE IF child is a div:
                        RECURSIVE call (Inspect this div element)
```

[5]https://chrome.google.com/webstore/detail/trackingobserver/obheeflpdipmaefcoefhimnaihmhpkao

```
    # Process lists A and B
    WHILE there are unprocessed items in list A OR list B:
        # Process input elements in list A
        FOR each element in list A:
            TRY:
                INPUT predetermined search term and COMMIT
                IF successful:
                    BREAK
                ELSE:
                # Process input elements in list A
                FOR each element in list B:
                    TRY:
                        INTERACT with element
                        WAIT for predetermined time
                        INPUT search term
                        IF successful:
                            BREAK

    WAIT for predetermined time
    RECORD traffic

TERMINATE algorithm
```

4.2. Implementation

The actual implementation of the algorithm is developed in the Python programming language and Selenium[6], which is a popular technology for browser automation and web testing. Selenium allows robotic process automation (RPA) [13] in the web environment, which makes it possible to automate interactions with web browsers, enabling developers to simulate user actions, such as navigating through web pages, clicking buttons, and filling and submitting forms [14]. Selenium's core component is Selenium WebDriver that provides a programming interface for interacting with web browsers and allows the developers to automate browser actions by writing code in their preferred programming language. Selenium WebDriver, designed to facilitate the development of browser automation applications, makes it quite effortless to build software that, for example, loops through chosen web pages and performs actions within them, as Selenium offers an ample package of classes and functions designed specifically for these purposes. It also has a decent level of integration with the Google Chrome Developer Tools, which allows for the easy automation of the network recording. In addition to Selenium, the application imports

[6]https://www.selenium.dev/

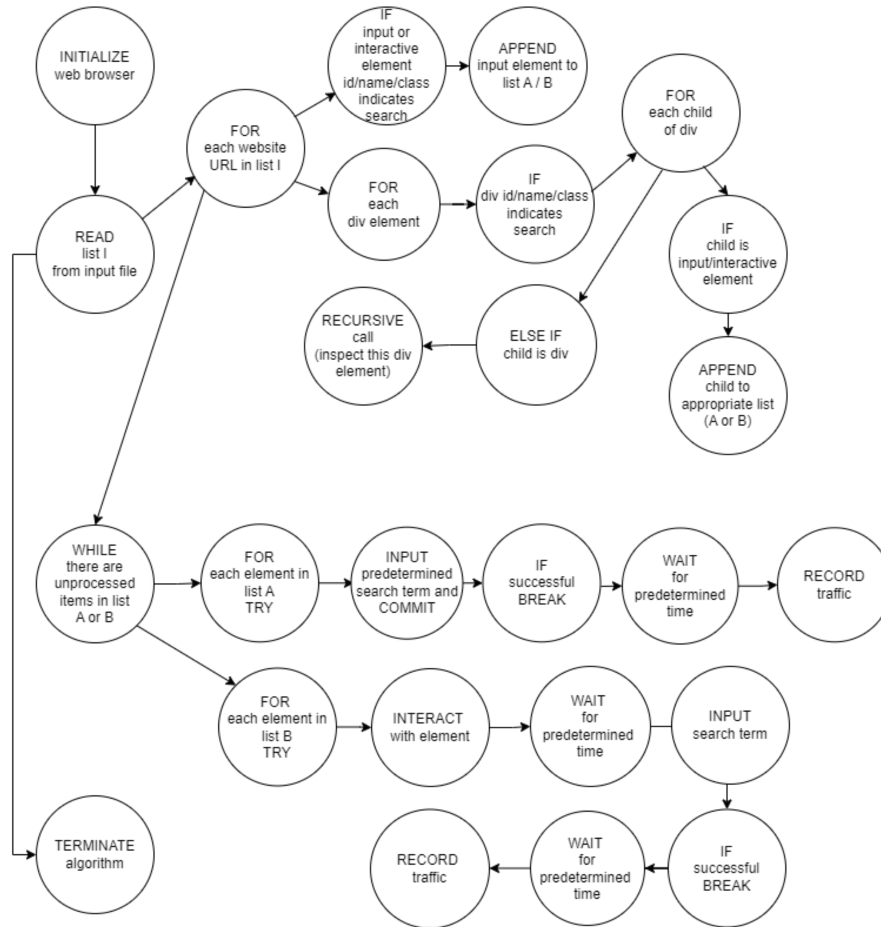

Figure 2: A state diagram of the algorithm.

the pychrome package which is also used in recording the traffic. The current implementation consists of only 298 lines of code in total, which speaks to the efficiency of the used technology.

On the start of the application the software initializes the Selenium WebDriver to use Google Chrome. It then configures Google Chrome Developer Tools to enable performance logging, and initiates the main loop which consumes a list of website URLs' which is provided from inputted .txt file. Then the application goes through the list and attempts to open each website in the list by using Selenium WebDrivers' `get(url)` -function. If successful, the application attempts to perform certain actions on each website in certain sequence. First, it initiates function `find_inputs()` which attempts to look for `input`, `form` or `div` -elements with a correct identifying attribute (either `id`, `name` or `class` which contains the term "search", "query" or "haku") in the web page and append them to a list. If the element it finds is a `div`, it creates a list out of its' children, which is recursively looped until it finds an input inside one of the list items. Then it attempts to run function `find_buttons()`, which looks for interactive or `div` -elements, referred to as buttons from this point onwards, currently defined as `button`, a and

```
    },
    "loaderId": "B00A2B438E2D442A197F6DF3139F710B",
    "redirectHasExtraInfo": false,
    "request": {
        "headers": {
            "Referer": "https://helsinki.mll.fi/",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTM
            "sec-ch-ua": "\"Not.A/Brand\";v=\"8\", \"Chromium\";v=\"114\", \"Google Chrome\";v=\"11
            "sec-ch-ua-mobile": "?0",
            "sec-ch-ua-platform": "\"macOS\""
        },
        "initialPriority": "Low",
        "isSameSite": false,
        "method": "GET",
        "mixedContentType": "none",
        "referrerPolicy": "strict-origin-when-cross-origin",
        "url": "https://consent.cookiebot.com/uc.js?cbid=6e9dd9d8-0273-48fb-b271-1069202b6c6c&conse
    },
    "requestId": "7377.78",
    "timestamp": 223918.547182,
    "type": "Script",
    "wallTime": 1686535260.661192
}
```


Figure 3: An example of an outputted log file.

span tags (the three most commonly used clickable elements) with proper identifying attribute (same as in the inputs part) and where there is no string "close" present in the attribute, and appends them to a list. If the element it finds is a div, it creates a list out of its' children, which is recursively looped until it finds an interactive element inside one of the list items.

After this phase has been completed the application attempts to process these lists in the following manner; first, if there are only input fields but no buttons found, it tries to initiate the function input_search_term(), which commits search with a predetermined arbitrary term. If there are both inputs and buttons, it attempts to first use the input_search_term and then the function click_search_button(), which attempts to interact with found button elements in such a way as to first click the element, then if this succeeds wait a second (for the search input to appear) and then input the same search term as in the previous step. If there are only interactive elements it attempts to use only click_search_button(). If either input_search_term() or click_search_button() or both yield results, in other words that activating them does not result in error, the ensuing traffic is recorded by Google Chrome Developer Tools, which is then outputted in JSON format in to a log file (see Figure 3). If there are no suitable elements it moves to the next website in the parameter list. When all websites in the parameter list are processed in this fashion the application execution ends.

The current implementation of the application has been implemented and tested on macOS, but it is effortless to write a version of it for other operating systems and devices too, as the difference in practice is just two lines of code which define the version of the WebDriver used.

4.3. Test Results

The application was tested by conducting several runs with it. In these test runs, the application was given a batch of websites, after which the test results, i.e. the log files the application created were analyzed. Various characteristics were determined, such as the success rate and

several and reasons for occasional failures. In our initial test, the success rate of the application was roughly 90% out of a test batch of 64 websites (58/64). While 90% may not sound like a high percentage at first, the tool still reduces manual work in our data leak analysis by 90% and is found to be very useful in research. Here, success refers to cases in which the application managed to commit search using an input field or managed to click an interactive element and then carry out the search. A test run was also considered a success if the application managed to open the web site, but no search related elements were found (as they did not exist at the website). The time the application used in interacting with the website was around 15–40 seconds, depending on how many elements it had to process through.

In our subsequent tests, which were conducted with a batch of 309 websites, the success rate of the application dropped somewhat, but was still 79.3% (245/309). The reason for this drop in the success rate was due to several problems which did not arise during the initial tests, mainly due to the nature of the larger batch size which happened to contain lots of similarly built websites, which of course produced the same problems. In these later tests, the median time the application used for processing one website was 43 seconds. The instances in which the application did not work as expected were because of several reasons:

- *Pop-up elements.* In some of the studied websites, there were pop-up elements, such as cookie consent banners or other notification windows that were overlayed with the search elements of the web page, which prevented the application from clicking the search button.
- *Empty search.* In some of the studied websites, the application proceeded to push buttons that initiated an empty search for some reason.
- *Predetermined search terms.* Some websites use input fields that allow the user to search with only predetermined terms, and inputting anything else just causes the search to not to commit. Regardless of whether the inputted term is hard-coded into the program or inputted by the user, this sometimes leads to situations in which the application cannot function correctly.
- *Responsiveness of the web pages.* In some of the tested websites, the responsiveness of the website layout forced the website to open in mobile view, mainly due to the reason of the default size of the browser window opened by the application, which sometimes situated the search functionality outside the reach of the application.
- *Unusually complex search functionality.* Some websites used search functionalities that were more complex than others, and sometimes this caused the application to fail.

Possible solutions to these situations are further explored in Section 6.

The log files outputted by the application are currently written only in JSON format, which is not the best possible option for human readers, but is sufficient for the current needs of the research. The application goes through the log file recorded by the Google Chrome Developer Tools and picks the values of the "method" key in network object, which it then prints to the JSON file. In addition to this, the JSON file consists of several technical details, and notifications on when the interception of data by third parties happened.

5. Challenges and Future Improvements

The most obvious challenge in developing a data leak detection tool we have described previously is in defining what kind of HTML tags it should attempt to inspect for purposes of finding interactable elements or input fields, as in the modern website ecosystem these are not necessarily obvious. For example, large amounts of elements that appear as "buttons" in websites are actually links, or even `span` type elements. This is largely because the classical `button` element comes with predetermined appearance and functionality, which in the modern, constantly evolving landscape of web development does not necessarily serve the purpose it had in the past. Thus the developers often choose to use more flexible alternatives, since it is easy to masquerade other elements to look like "buttons" with CSS styling and to add interactivity to them with JavaScript.

Another major developmental challenge lies in the fact that the elements being looked for might not, and indeed often do not, have a correct `id`, `name` or `class` attribute that is needed to properly identify the element in question to be a "search" element. In some cases they might not have any kind of verbal indicator of their nature, for example being represented only by a looking glass icon, which is not even named correctly, to describe their nature. This is especially challenging in the cases where the search input field is hidden behind a collapsible element, accessible only through an interactive element that lacks any identifiable attributes. In some cases the elements sought for are not present in the DOM (Document-object model) [15] at all (in other words, they do not exist yet) until some interactive element is clicked, which makes it very hard to detect the sought-after element if the connected interactive elements have misleading attributes.

While the application worked well in our testing, it had several functionality issues that demand further development. In some of the tested websites the application could not click the search button it detected as there was another element, usually a cookie banner, overlaid above it in some way. Another problem was that in some cases clicking the search button initiated an empty search, which produced no results. The third issue was that sometimes the websites deploy search fields that accept only certain inputs, and this can not be detected properly by automation. In addition to this, there were other issues that were slightly more specific, such as the occasional responsiveness attributes interfering with the application, complex search functionalities which the application could not use. Several different solutions to these challenges have been considered, and we plan to implement them in the future.

For the reasons above, there have been considerations on adding machine-learning aspects to this application in the future. For example, elements whose search functionality is indicated only by an icon, could be identified with a properly taught neural network. Using artificial intelligence could also help in avoiding some of the pitfalls mentioned in the preceding paragraph. Another area where the artificial intelligence technology could be found useful is in automating the categorization and analysis of the results gathered from websites, thus hastening the actual research considerably. Furthermore, another avenue of further development would be a mechanism to detect the language of the website in question, and then add the "search" also in this language to the criteria of elements to be looked for. Currently, the application outputs the data it collects only in `JSON` format, which is human-readable but still onerous to analyze. This can be developed further for it to produce an easily readable output like Web Evidence

Collector mentioned in Section 2 does.

The current implementation of the algorithm is very specifically meant to study only the data leaks happening in the search functionality of the websites in question. However, it would be quite easy to use this algorithm and codebase to, for example, make the application just go through the given websites and click all clickable elements, and then record what kind of data was leaked to third parties while doing so. With minor alterations this tool could be used also in other research projects, basically in any studies that demand large amounts of statistical data about websites, such as how prevalent certain content types, elements or cookies are. The basic structure of the algorithm, in other words that it loops through a given list of websites and performs actions in them, could be easily turned into retrieving all kinds of information from the sites, which could be put to actual use in many non-technical sciences that study human behavior on the internet. For example, this concept could be used to scrape all posts with certain keywords or topics from a list of discussion forums or social media platforms at once, or find all items with a specific keyword from several different webstores. It could also be implemented in such a way as to access video streaming websites and search for specific types of videos based on their tags. However, in order to be useful for other than computer science research interests, the implementation should include a graphical user interface, as it currently has none. Adding the GUI and some modularity, for example the parametrization of the terms the algorithm uses in detecting the wanted types of elements to interact with, to make the application more usable for other research interests has been proposed in our internal discussions, and will be implemented in the future.

6. Conclusion

In the current paper, we have presented a proof-of-concept implementation of a tool employing automated traffic analysis to record and analyze potential leaks of personal data to third-party services. This makes it easier to systematically gather large datasets for scientific research on potential data leaks caused by third-party services planted on the websites. The algorithm in its current form provides a good basis on which to improve further in the future, for example by adding machine learning to aid in searching for the wanted elements and in empowering the analysis of the results. Currently, the test results indicate that the application is not infallible, but still quite capable of detecting correct interactable elements from websites, activate them and record the traffic with third parties if it happens. We intend to continue the development of the application in the future, improving its usability and performance. The objective is to make it a useful tool not just for the needs of our data leak research project, but for other scientific interests as well.

Acknowledgments

This research has been funded by Academy of Finland project 327397, IDA – Intimacy in Data-Driven Culture.

References

[1] J. Bhatia, T. D. Breaux, J. R. Reidenberg, T. B. Norton, A theory of vagueness and privacy risk perception, in: 2016 IEEE 24th International Requirements Engineering Conference (RE), IEEE, 2016, pp. 26–35.

[2] J. R. Reidenberg, J. Bhatia, T. D. Breaux, T. B. Norton, Ambiguity in privacy policies and the impact of regulation, The Journal of Legal Studies 45 (2016) S163–S190.

[3] F. Palomino, F. Paz, A. Moquillaza, Web Analytics for User Experience: A Systematic Literature Review, in: International Conference on Human-Computer Interaction, Springer, 2021, pp. 312–326.

[4] V. Kumar, G. A. Ogunmola, Web analytics for knowledge creation: a systematic review of tools, techniques, and practices, International Journal of Cyber Behavior, Psychology and Learning (IJCBPL) 10 (2020) 1–14.

[5] A. Huidobro, R. Monroy, M. A. Godoy, B. Cervantes, A Contrast-Pattern Characterization of Web Site Visitors in Terms of Conversions, in: Technology-Enabled Innovations in Education: Select Proceedings of CIIE 2020, Springer, 2022, pp. 31–51.

[6] B. Chitkara, S. M. J. Mahmood, Importance of web analytics for the success of a startup business, in: Data Science and Analytics: 5th International Conference on Recent Developments in Science, Engineering and Technology, REDSET 2019, Gurugram, India, November 15–16, 2019, Revised Selected Papers, Part II 5, Springer, 2020, pp. 366–380.

[7] P. Bekos, P. Papadopoulos, E. P. Markatos, N. Kourtellis, The Hitchhiker's Guide to Facebook Web Tracking with Invisible Pixels and Click IDs, in: Proceedings of the ACM Web Conference 2023, 2023, pp. 2132–2143.

[8] K. Curran, T. Dougan, Man-in-the-browser attack, International Journal of Ambient Computing and Intelligence 4 (2012).

[9] R. Carlsson, S. Rauti, S. Mickelsson, T. Mäkilä, T. Heino, E. Pirjatanniemi, V. Leppänen, Several online pharmacies leak sensitive health data to third parties, Accepted to WorldCIST 2023 (2023).

[10] V. Wesselkamp, I. Fouad, C. Santos, Y. Boussad, N. Bielova, A. Legout, In-depth technical and legal analysis of tracking on health related websites with ernie extension, in: Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society, WPES '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 151–166. URL: https://doi.org/10.1145/3463676.3485603. doi:10.1145/3463676.3485603.

[11] S. Englehardt, et al., Automated discovery of privacy violations on the web, 2018.

[12] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, B. Preneel, Fpdetective: dusting the web for fingerprinters, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 1129–1140.

[13] W. M. Van der Aalst, M. Bichler, A. Heinzl, Robotic process automation, Business & information systems engineering 60 (2018) 269–272.

[14] B. García, M. Gallego, F. Gortázar, M. Munoz-Organero, A survey of the selenium ecosystem, Electronics 9 (2020) 1067.

[15] J. Stenback, P. Le Hégaret, A. Le Hors, Document object model (dom) level 2 html specification, W3C, 2003.