BACHELOR THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

Misusing browsers' login managers for data exfiltration by third parties

Author:
Lucas van Kasteren
S1039270

First supervisor/assessor:
Assistant professor, Dr. Gunes Acar
g.acar@cs.ru.nl

Second assessor:
Associate professor, Dr. Ir. Erik Poll
erikpoll@cs.ru.nl

January 18, 2023

Abstract

In this thesis a research about data exfiltration by third-party scripts has been conducted. In detail, this entails abuse of browsers' internal login managers without the users' consent. The user fills out a login form on a website and saves their credentials in the browsers' login manager. When the user starts browsing on that particular website a third-party script is inserted which is autofilled by the browsers' built-in login manager. The script then obtains the users' email address and sends hashes of it to a third-party server. We run web crawls on 50000 websites to find evidence of such tracking. Tracker Radar Collector basic instrumentation is extended to detect whether sensitive data is being injected into the pages' context. A new collector is added which uses the `MutationObserver` API to detect DOM changes. Moreover, a bait technique is used to allow sensitive data injections into websites such that third-party scripts can access and exfiltrate the data. The analysis of our results reveals leaks via URLs and POST request bodies and discovers input field sniffs. However, no cases were found where login managers were abused for the purpose of third-party web tracking. We compare these results to previous research and discuss the limitations of our own research.

Contents

Chapter 1

Introduction

Nowadays, third-party web tracking can be found everywhere. In 2021 94% of desktop sites included at least one third-party resource and almost 46% of requests on desktop are third-party requests[5]. Third-party requests can be used for functionality which includes personalisation, site analytics, and targeted advertising. It is even claimed that preventing targeted advertising via third-party requests has a negative economic impact[14]. However, third-party request can also be used for tracking purposes. With this third-party tracking, an entity, other than the website directly visited by the user, is collecting personal identifiable information (PII) about the user. Examples of such trackers are cookies and browser fingerprinting.

The thesis is inspired by an earlier work[3] performed by Acar *et al.*. They investigated three data exfiltration attacks by third-party scripts. The attack that misuses the browsers' internal login managers is the foundation for this paper. Data was gathered by extending OpenWPM[20] to behave as if it has already interacted with a website. They found that two scripts, loaded from AdThink (audienceinsights.net) and OnAudience (behavioralengine.com), sent hashes of users' email addresses to its server for the purpose of third-party tracking. These scripts were found on 1,110 of the Alexa top 1 million sites in September 2017. After the publishing of Acar *et al.*'s work[3], these two companies stopped using the technique however, this does not mean that there are other companies who adopt this technique. Moreover, Senol *et al.* [23] found that users' email addresses are exfiltrated to tracking, marketing and analytics domains before form submission and without giving consent.

As seen above this tracking often happens without the users' consent or knowledge. PII leakage to third-party domains via different methods is a thriving topic the past few years. This proves the relevance of this thesis since pages that host tracking forms must respect the users' privacy and not leak, on accident or intentionally, user's PII to third-parties.

This research paper documents the development of a tool, which tries to detect login-manager abuse for email exfiltration based on previous work. In particular, we ask ourselves the question if browsers' built-in login managers are being abused by third-party scripts for the purpose of third-party web tracking without user awareness?

All major browsers have built-in login managers to save and automatically fill credentials. Every browser determines in their own way which login forms to autofill, but at least a username and password field must be available. Browsers' login managers automatically fill in previously saved credentials for known form fields. Scripts may misuse this feature and insert invisible login forms into the web pages' context. These scripts then trigger the browsers' built-in login managers and subsequently read the filled input field. Hence user credentials are obtained for the purpose of third-party web tracking.

To help achieve the goal of discovering this non-consensual tracking, DuckDuckGo's Tracker Radar Collector (TRC) web crawler is used. Tracker Radar Collector[11] is a Puppeteer-based[1] crawler used to detect and measure third-party trackers for their Tracker Radar[10]. While one can use TRC as it is, some extensions were performed to make it suitable for our research.

The final product will use a bait technique similar to the method used by Acar *et al.*[3] and use leak detection methods based on Senol *et al.*[23] and Englehardt *et al.*[12]. It is a practical analysis to explore web privacy vulnerabilities and address its side-effects.

Chapter 2 will go over previous work and explain differences and similarities with this thesis. Moreover, it will explain technical background and knowledge necessary to understand the design choices for our tool before diving into the actual research performed in chapter 3. Chapter 4 will then display the findings before getting to a discussion of the results and limitations in chapter 5. Finally, chapter 6 will conclude our research.

Chapter 2

Background and Related Work

In this chapter all necessary details to understand the design choices and the development process of the research are addressed. We will discuss web tracking and in particular third-party web tracking. Furthermore, we will discuss related work and dive into the details of previous studies conducted.

2.1 Background

Web tracking refers to the practice of collecting data about an individual's online behaviour. There have been many studies to forms of identifying online behaviour through web tracking. This entails tracking examples via cookies[13], browser fingerprinting[7], ETags[6], etc.. The gathered data can be used for different purposes such as website analytics and targeted advertising/marketing.

When talking about third-party tracking we refer to the use of tracking technologies by entities other than the website the user is actually visiting. For instance, when visiting a website which has ads displayed by third parties. These third parties can use tracking techniques to collect data about the user's behaviour. This advertising company can then show you targeted ads based on your behaviour. This may raise concerns about privacy since these third-party trackers allow companies to collect and use your personal data without the user's knowledge or consent[19].

As already briefly touched upon in the introduction section, we will explain the actual attack we are trying to detect in this research paper. First, a user fills out a login form on a website and saves their credentials in the browsers' login manager. Secondly, the user starts browsing on that particular website where a third-party script is present. This third-party script inserts an invisible login form which is autofilled by the browsers' built-in login managers. Hence, the script obtains the users' email address and sends hashes of it to a third-party server for the purpose of third-party tracking.

2.2 Related Work

Several research papers have been written on login-manager abuse for email exfiltration and personal identifiable information (PII) leaks via third-party tracking. The inspiration for this paper is the research performed by Acar *et al.*[3]. In this paper PII leakage is discovered by tracking scripts misusing login-managers. The discovery method uses a bait technique, which allows to inject sensitive user data into the context of real websites in such a way that third-party scripts can access and exfiltrate the data. This bait technique is also used in a similar way in this paper, more details can be found in chapter 4.

Similarly, Senol *et al.*[23] built a crawler which finds and fills email and password fields, monitors the network traffic for leaks, and intercepts script access to filled input fields. As a result, they found that users' email addresses are exposed to tracking domains before form submission in over 1800 websites in the EU and over 2900 websites in the US. Although the scope of this research is not a big as the one performed by Senol *et al.*[23], the methodology for detection PII leakage is highly similar between our studies.

Different researches about PII leakage to third-party domains have already been performed for quite some time. As discussed by Englehardt *et al.* leakage can be classified as intentional or unintentional/accidental[12]. The first practise occurs when the website intentionally leaks user's PII to third-party trackers, whereas in the second case leaked values are passed via HTTP referer headers[12]. They discovered that even the simple act of viewing emails can leak PII to third-parties. More specifically, in majority of the leaked cases they are intentional.

Moreover, in 2016, Starov *et al.* reported the first large-scale study of PII leakage via contact pages of the 100,000 most popular sites of the web[26]. They witnessed PII leakage towards third-parties in a variety of ways, including the leakage through third-party form submissions.

More recently, Dao *et al.* documented the first in-depth analysis of leaked PII in the users' sign-up and sign-in flow (authentication flows)[8]. One of the things they discovered with their analysis is that for 307 popular shopping sites, 42.3% leak the PII to third-party services.

Above research papers show that PII leakage to third-party domains via different methods is a hot topic in recent years. This also proves the relevance of this research paper since pages that host tracking forms must respect their users and not leak, on accident or intentionally, user's PII to third-parties.

Chapter 3

Research

To study browsers' built-in login manager abuse for the purpose of third-party web tracking we conducted a research based on the methods used by Acar *et al.* in their study[3]:

1. Extending Tracker Radar Collector (TRC) with a new collector through which injecting sensitive data into the page context is observed. The `MutationObserver` API is used to detect Document Object Model (DOM) changes and log these 'mutations'.

2. HTMLInputElement is instrumented to observe when input field objects are accessed. A bait technique with fake credentials acts as the browsers' built-in login manager. If the input field gets accessed we log the value to detect if the bait email was read.

3. A leak detection method to detect whether the username and password are sent to a third-party server. Further details of this method are explained in Section 4.3.

A high-level overview of the extension of TRC can be found in figure 3.1.

3.1 Developing the crawler

As mentioned in the introduction chapter, DuckDuckGo's Tracker Radar Collector (TRC)[11] is used to crawl the web. TRC is a Puppeteer-based[1] crawler which uses the Chrome DevTools Protocol to interact with the browser. TRC uses collectors to gather tracking data from websites, such as cookies, API calls and requests. Moreover, it uses conditional breakpoints which allows you to break inside a code block when a defined expression evaluates to true. If TRC hits such a breakpoint it logs data about the respective function or object accesses. When a website is crawled successfully, it creates a separate unique JSON file named after the website which contains all the crawled data. Additionally, for each crawl a metadata.json file

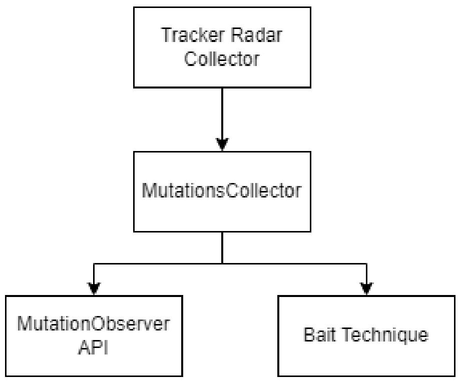

Figure 3.1: High-level overview of the extension of Tracker Radar Cllector

is created with high-level stats such as start- and endtime, used collectors and the system environment.

For TRC to work the way it is intended it to work, it is extended with another collector called `MutationCollector`. The code for this collector can be found in appendix A. This collector monitors DOM mutations to see if new login forms are injected into the page. To achieve this goal the `MutationObserver` API is used[25]. `MutationObserver` is a powerful API to detect changes in the DOM. At first, you specify a target node to observe (this can be any object in the DOM) for changes. It waits for a script or task to be completed, meanwhile recording all DOM mutations in a record queue. When the execution is completed, it outputs all observed mutations in an array by making use of a *callback* function[1] instead of just polling the DOM. The `MutationObserver` API is an effective way to detect DOM changes and is supported by all major web browsers and observes basic operations such as adding or deleting nodes and changing attribute values[16].

The `MutationObserver` specified in this research looks for invisible injected form and input fields into the DOM. It uses the root document node as target node to observe the full page. Moreover, it logs the dimension (width and height) of an injected input field to see if the form injected is invisible. It is likely to assume that when a form doesn't have a width and height position on the page, it is invisible. Finally, it tracks the form's action – the URL that processes the form data – to know whether this URL is also the one that is sending username/password to a third-party server and not a website using for instance a form builder[2].

When input fields are detected, the collector tries to fill the input field with fake login credentials to detect whether information from an input field is being read. If a field can be filled it triggers the breakpoint set on

[1]A function that is passed as an argument to another function such that it can be executed in that other function.

[2]A common form builder: `https://www.hubspot.com/products/marketing/forms`

the HTMLInputElement and logs the details of the input element access specified in the APICallCollector. These details contain the id, type and value of the input field tag. Moreover, it registers the initial URL and a timestamp of when the capture happened.

Besides checking for potential PII leaks in URLs, it is also checked whether PII leaks occur in POST request bodies. An attribute is added to the request collector to collect the POST data belonging to the request bodies. Extending it in this way is similar to the one used by Senol *et al.* in their research[23].

3.2 The dataset

To detect browsers' built-in login manager abuse for the purpose of third-party web tracking 50000 websites were crawled from the Tranco list[3] generated over the period 23 November 2022 to 22 December 2022 (30 days)[17]. This list aggregates the ranks from the lists provided by Alexa[4], Cisco Umbrella[15], Majestic[18], Quantcast[21] and Farsight[22]. The downloaded list uses only domains included in the Chrome User Experience Report[2], which contains actual URLs visited by Chrome users.

The entries were put in a text file as input for Tracker Radar Collector. To run the crawls a cloud-based server hosted on DigitalOcean[9] was used. DigitalOcean is an American cloud infrastructure provider headquartered in New York City with data centers worldwide. DigitalOcean offers a range of services, including virtual private servers (VPS), object storage, and managed databases. For this research a VPS is used, which in DigitalOcean is called a droplet. A droplet is a lightweight, isolated, and easily-scalable virtual machine that can be used to run a wide range of applications. On this droplet, the image used is Ubuntu 22.10 x64 with a 8GB RAM memory, 80GB of disk space and 4 AMD CPUs. To access this cloud server PuTTY was used. PuTTY is a free and open-source terminal emulator, serial console and network file transfer application[24]. It is used to connect to the Secure Shell (SSH) protocol of the DigitalOcean server.

Before the main crawl of 50000 websites was ran, a smaller crawl of 10000 websites was crawled[4] generated over November 2022[17]. It successfully crawled 9175 websites and was ran from New York City. However, since this crawl did not gave desired output a bigger crawl was started using the above mentioned 50000 websites. This was done to up our chances of finding a potential leak. This second crawl successfully crawled 46137 websites (92.3%) and was also ran from New York. Both of these crawls used the same Tranco configuration and used the same DigitalOcean setup mentioned earlier.

[3]Available at: `https://tranco-list.eu/list/JX5KY/50000`

[4]Available at: `https://tranco-list.eu/list/5YW6N/10000`

The command to run the crawls is:

```
npm run crawl -- -i ./crawl_lists/tranco_top10000.txt -o
./output_top10000/ -v -d requests,apis,mutations
```

This configuration is explained in the following points:

- `-i ./crawl_lists/tranco_top10000.txt` specifies the path to the text file with the list of URLs to crawl.
- `-o ./output_top10000/` specifies the output folder where the output files will be created.
- `-v` instructs the crawler to log additional information to the console while crawling.
- `-d requests,apis,mutations` instructs the crawler which collectors to use. It uses the request collector to detect leaks in requests. Moreover, the APICallCollector is used to log input element accesses and it uses the MutationsCollector to detect changes in the DOM for potential malicious scripts.

3.3 Leak detection

To detect whether potential malicious scripts send our email address and/or password to their third-party server we need to identify (potentially) encoded, hashed or obfuscated leaks. This is a challenge since email addresses might not be in plaintext but rather, encoded or hashed. For this reason a technique similar to Englehardt *et al.*'s[12] and Senol *et al.*'s[23] methods are used. Given a set of encoding and hashes, Englehardt *et al.*'s technique searches for email leakage in HTTP traffic[12]. Senol *et al.*'s improves thereafter this method through splitting content by separators and decoding the resulting strings, to search for different encodings of the search terms (email and password values)[23]. Senol *et al.* searches for potential leaks in the referrer header, cookies, URL and POST bodies of the requests[23]. However, using all these methods is out of scope for this research so only leak detection via URL and POST request bodies is used. Similarly to above mentioned previous work, we detect for a maximum of three encoding/hash layers. We use likely encodings and hashes as defined in Senol *et al.*'s research[23], which can be found in Appendix A.

To use this method on the collected JSON files a Python script has been written to apply the leak detection method. In this script a leak detector object is used to search for the 'fake credentials' in request URLs and POST bodies. Moreover, we log whether we find a sniffed value read from an input field. This is done to determine which source tried to read from an input field.

To fill a potential input field it must be logged if an invisible form has been injected. As discussed before this is captured by the MutationsCollector. However, to determine whether an input field leak is actually initiated by the same party as the one injecting the script we log the forms action attribute. The forms action attribute specifies where to send the form-data when a form is submitted. This is then compared to the URL/POST body which leaked the email address or the password. In the end this will give us a list of all input field sniffs, all leaks and all injected input fields.

Chapter 4

Results

This chapter contains the results of the research described in chapter 4. In section 5.1 the general results of the conducted research will be discussed. Next we make a comparison with previous work in section 5.2 and we finish by sharing other interesting findings from our study

4.1 General results

From the execution of our research, 46127 websites were crawled successfully out of our dataset of 50000 websites (92.3%). Each successfully crawled website created a corresponding JSON file with the data. When running the leak detection program on these JSON files 76 site leaks on 21 websites and 193 input field sniffs were found. From these leaks almost half of them came from the known third-party trackers domains `ct.pinterest.com` and `api.rlcdn.com` (15 leaks). The results of this can be found in table 4.1. In this table the type of leak can be found, including if the leak occurred in the URL or via POST request bodies. The encoding of the leak type can be found and if not present there was no encoding detected. Finally, one can find how many times a collection endpoint made a unique request on that particular website. In the table you can only find unique requests for the certain endpoints so therefore in total 38 unique site leaks were found.

From all the input field sniffs that were found, we found that websites having a leak also had input field sniffs. In most cases these were common form builders such as hsforms[1] or TrustedForms[2]. The input field sniffs also recorded sniffs of the known trackers `ct.pinterest.com`, `api.rlcdn.com` and `rs.fullstory.com`.

Moreover, in our research we used a bait technique with fake credentials to act as the browsers' built-in login manager. When looking at the mailbox of this bait email we did not receive any emails. This means our

[1]`https://www.hubspot.com/products/marketing/forms`
[2]`https://activeprospect.com/trustedform/`

bait credentials did not get tracked by third-party web trackers. In addition to this, no scripts that misused the browser login manager to extract user email addresses were found. This was confirmed by looking at the results of the `MutationsCollector` for each website which had a leak. From this we could confirm that no scripts with input elements for tracking purposes were inserted into the webpages context.

4.2 Comparison with previous work

Since this thesis is inspired by an earlier work[3] performed by Acar *et al.* a comparison needs to be made. In their study they found two scripts that misused the browser login manager to extract user email addresses. This was found on 1,110 of the Alexa top 1 million sites in September 2017. To compare this with our results no such scripts as discovered by Acar *et al.* were found[3]. In detail, this means that no websites were discovered where form or input fields were injected. Chapter 5, discussion, will explain why this might be the case.

On the other hand, it was discovered that `rlcdn.com` is the most prominent tracker domain that collects hashed email addresses. LiveRamp (the entity behind rlcdn) collected the MD5, SHA-1 and SHA256 hashes of the email address typed into login forms. Furthermore, we also found leaks of the tracker domain *fullstory.com*. Both these leaks are similar to what Senol *et al.* found in their research[23].

4.3 Other results of note

As can be seen in table 4.1 only email address leaks were found. However, we also discovered our bait password via a POST request body. This was discovered on the website `bikereg.com`. According to the leak detection program we found that the website made a request to itself which means it collects itself. This sounded hard to believe so the initiators of the specific request were inspected. We found that bikereg loaded scripts from CodePlex, Amazon CloudFront and the jQuery JavaScript library. All of these either enable users to upload and share their own software (CodePlex), speed up distribution of static and dynamic webcontent (CloudFront) or simplify HTML DOM tree traversal and manipulation (jQuery).

Moreover, when looking at the website source code we could not find a use case which purposely leaks to trackers. Hence, it is likely that this password which got detected by the leak detection program is not a request to a third-party domain for the purpose of web tracking. This for the reason that this request happened by the first-party domain and we excluded first-party domains from the results. Therefore, it is plausible that `bikereg.com` does not purposely leak passwords to third-parties.

Leak Type	Location	Encoding	Quantity	Website	Collection Endpoint
Email	URL	URL	4	www.bizzabo.com	https://ws.zoominfo.com
	URL	md5-sha1-sha256	2	www.buzzfeednews.com	https://api.rlcdn.com
	URL	URL	1	www.extensis.com	https://okt.to
	URL	md5-sha1-sha256	1	www.favecrafts.com	https://api.rlcdn.com
	URL	URL	2	www.getambassador.com	https://ws.zoominfo.com
	URL	sha256	2	www.gorgias.io	https://ct.pinterest.com
	URL	md5-sha1-sha256	2	www.searchenginejournal.com	https://api.rlcdn.com
	URL	md5-sha1-sha256	1	www.livestrong.com	https://api.rlcdn.com
	URL	sha256	1	www.logi.com	https://ct.pinterest.com
	URL	sha256	1	www.logitech.com.cn	https://ct.pinterest.com
	URL	sha256	1	www.logitech.com	https://ct.pinterest.com
	URL	URL	2	www.mbsy.co	https://ws.zoominfo.com
	URL	md5-sha1-sha256	2	www.metalinjection.net	https://api.rlcdn.com
	URL	sha256	2	www.metropolismag.com	https://ct.pinterest.com
	URL	URL	2	www.openshift.com	https://autocomplete.demandbase.com
	POST		1	www.bizzabo.com	https://forms.hsforms.com
	POST		1	www.bizzabo.com	https://rs.fullstory.com
	POST	URL	1	www.bikereg.com	https://www.bikereg.com
	POST		1	www.getambassador.com	https://.getambassador.com
	POST		1	www.purecars.com	https://rs.fullstory.com
	POST		1	www.chargebee.com	https://api.factors.ai
	POST	URL	3	www.veteransunited.com	https://create.leadid.com
	POST	sha1	1	www.veteransunited.com	https://api.trustedform.com
	POST		1	www.mbsy.co	https://getambassador.com
	POST		1	www.ex.co	https://rs.fullstory.com

Table 4.1: Email address leaks found in request URLs of given collection endpoints of the crawled websites. Including how many times the leak was found in a different request URL with that specific endpoint.

Chapter 5

Discussion

In this chapter a discussion on the results will be provided. In particular, limitations of our study will be discussed and possibilities for future work.

5.1 Limitations

When our research was conducted several limiting factors were discovered. These limiting factors needs to be discussed when interpreting the gathered results in chapter 4.

5.1.1 The dataset

We used 50000 websites in our crawl. Since no actual scripts were discovered, 50000 websites are not enough. To compare, Acar *et al.* crawled 1 million websites in their study which did find two scripts that misuse browser's built-in login manager to extract user email addresses[3]. These scripts were present on more than 1000 websites.

5.1.2 Crawling

In our research we choose to run a crawl on a cloud-based server hosted on DigitalOcean. This may be problematic since the crawler could have been marked as a bot when visiting websites. This might be a reason why 3873 out of 50000 website visits (7.7%) failed. However, Zeber *et al.* did point out that the difference between automated measurements techniques over human gathered data is rather small for third-party domains and URLs[27]. Since we do not have log files to verify why this happened it will remain unknown as to why these visits failed.

Another note is that we crawled from a server hosted in the US. This can have an impact on the amount of third-parties that were found on websites. This should be taken into account when using our results outside of the US.

5.1.3 Leak detection

In our leak detection method only likely hashes and encodings, specified in Appendix A, are used. This means that our leak detection method may still miss leaks that are not encoded or hashed using these likely encodings. Rather, there could have used custom encodings/hashes which we would have never been able to discover.

Another note is that in this research there has only been searched for leaks in URLs and POST request bodies. However, leaks can occur in other places as well such as cookies or referrer header. This is what Senol *et al.* did use in their research[23]. As mentioned in chapter 1, introduction, this was out of scope for this thesis but mentioning it can help future work.

5.2 Future work

When looking at future work there are several options which can be applied. The first part entails improving upon the used methodology. One can crawl more websites or use a different crawler configuration to get more complete data. This can mean that for instance one million websites can be crawled and instead of using Tracker Radar Collector one can use OpenWPM to get a more representative dataset. In addition to this, crawls can be run from the EU as well instead of only the US. This will give more of a comparison between the third-party trackers used in both regions.

However, not only the methodology can be improved upon for future work. The analysing of results can also be extended in certain ways. As already mentioned in the limitations, future work can focus on a better leak detection method such that leaks that got undiscovered up until now might be discovered in the future. Moreover, websites can be categorized to see if websites with similar leaks have similar tracker providers.

Chapter 6

Conclusions

In this thesis we studied browsers' built-in login manager abuse for the purpose of third-party web tracking. A list was collected of the Tranco top 50000 websites generated over the period 23 November 2022 to 22 December 2022 (30 days). We crawled this list using a modified version of DuckDuckGo's Tracker Radar Collector. We used the `MutationObserver` API to detect DOM changes and detect whether sensitive data is injected into the page's context. A bait technique acted as the browsers' built-in login manager to know when to log sensitive input reads. Following this we analysed the gathered data using a leak detection method.

Our findings using our leak detection method found 76 site leaks on 21 websites. More specifically, we found URL and POST request leaks of known third-party tracker domains such as `rlcdn.com` (LiveRamp) and `fullstory.com` (FullStory). However, we did not discover any third-party scripts that access and exfiltrate sensitive user data by misusing browsers' login autofill.

Moreover, we found 193 input field sniffs on our crawled websites. These results were compared to previous research conducted by Senol *et al.*[23] and Acar *et al.*[3] and showed similar results with the former.

Our bait technique which filled in fake credentials did not get tracked by third-party web trackers because no emails in the mailbox of this fake credential were received.

Finally, we discovered that some leaks were not requests to third-party domains for the purpose of web tracking but common form builders or an incidental collection of a request made by a first-party.

Bibliography

[1] Puppeteer. Puppeteer is a Node library which provides a high-level API to control headless Chrome or Chromium over the DevTools Protocol. `https://developer.chrome.com/docs/puppeteer/`.

[2] The Chrome User Experience Report . `https://developer.chrome.com/docs/crux/about/`.

[3] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. *Proceedings on Privacy Enhancing Technologies*, 2020(4):220–238, 2020.

[4] Amazon. Alexa Top Websites – Most Popular Sites List 2022. Alexa rank is a global ranking system that ranks millions of websites in order of popularity. `https://www.alexatopwebsites.com/`.

[5] HTTP Archive. Web Almanac, 2021. `https://almanac.httparchive.org/en/2021/`.

[6] Mika D Ayenson, Dietrich James Wambach, Ashkan Soltani, Nathan Good, and Chris Jay Hoofnagle. Flash cookies and privacy ii: Now with html5 and etag respawning. *Available at SSRN 1898390*, 2011.

[7] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Nordic conference on secure it systems*, pages 31–46. Springer, 2012.

[8] Ha Dao and Kensuke Fukuda. Alternative to third-party cookies: investigating persistent pii leakage-based web tracking. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 223–229, 2021.

[9] DigitalOcean, Inc. DigitalOcean. DigitalOcean, Inc. is an American cloud infrastructure provider with data centers worldwide. `https://www.digitalocean.com/`.

[10] DuckDuckGo. Tracker Radar. Tracker Radar is a best-in-class data set about trackers that is automatically generated and maintained

through continuous crawling and analysis. https://spreadprivacy.com/duckduckgo-tracker-radar/.

[11] DuckDuckGo. Tracker Radar Collector. Modular, multithreaded, puppeteer-based crawler used to generate third party request data for the Tracker Radar. https://github.com/duckduckgo/tracker-radar-collector.

[12] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. *Proc. Priv. Enhancing Technol.*, 2018(1):109–126, 2018.

[13] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299, 2015.

[14] Avi Goldfarb and Catherine E Tucker. Privacy regulation and online advertising. *Management science*, 57(1):57–71, 2011.

[15] Dan Hubbard. Cisco Umbrella 1 Million. The Cisco Umbrella 1 Million is a free list of the top 1 million most popular domains. https://umbrella.cisco.com/blog/cisco-umbrella-1-million.

[16] Junaid Iqbal, Ratinder Kaur, and Natalia Stakhanova. PoliDOM: Mitigation of DOM-XSS by Detection and Prevention of Unauthorized DOM Tampering. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–10, 2019.

[17] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.

[18] Majestic. The Majestic Million. Majestic Million is a FREE league table of the top 1 million websites in the world. https://majestic.com/reports/majestic-million.

[19] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE symposium on security and privacy*, pages 413–427. IEEE, 2012.

[20] OpenWPM. OpenWPM is a web privacy measurement framework which makes it easy to collect data for privacy studies on a scale of thousands to millions of websites. https://github.com/openwpm/OpenWPM.

[21] Quantcast. Quantcast. `https://web.archive.org/web/20200105223115/https://www.quantcast.com/top-sites/`.

[22] Farsight Security. Farsight. `https://www.farsightsecurity.com/`.

[23] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. Leaky forms: A study of email and password exfiltration before form submission. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1813–1830, 2022.

[24] Simon Tatham. PuTTY. PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. `https://www.putty.org/`.

[25] DOM Living Standard. MutationObserver. The MutationObserver interface provides the ability to watch for changes being made to the DOM tree. `https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver`.

[26] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. Are you sure you want to contact us? quantifying the leakage of pii via website contact forms. *Proc. Priv. Enhancing Technol.*, 2016(1):20–33, 2016.

[27] David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, and Martin Lopatka. The representativeness of automated web crawls as a surrogate for human browsing. In *Proceedings of The Web Conference 2020*, page 167–178. Association for Computing Machinery, 2020.

Appendix A

Appendix

Code snippet of the MutationCollector which was created to extend Tracker Radar Collector:

```
const puppeteer = require('puppeteer');
const BaseCollector = require('./BaseCollector');


class MutationsCollector extends BaseCollector{

    id() {
        return 'mutations';
    }
    /**
     * @param {import('./BaseCollector').CollectorInitOptions}
        options
     */
    init({log, url}) {
        /**
         * @type {any[]}
         */
        this._insertedNodes = [];
    }


     /**
     * @param {{cdpClient: import('puppeteer').CDPSession, url:
        string, type: import('./TargetCollector').TargetType,}}
        targetInfo
     */
    async addTarget({cdpClient, url, type}) {
        await cdpClient.send('Page.enable');
        await cdpClient.send('DOM.enable');
        const SOURCE_STRING = `
        const fillInputFields = function(fields){
            for(const field of fields){
                fillInputField(field);
            }
        };

        const fillInputField = function(/** @type {
            HTMLInputElement} */ field){
            //check type and fill accordingly
            if(field.type == 'email' && field.tagName != 'form')
                {
                field.value = 'hello@gmail.com';
            }
            else if (field.type == 'password'  && field.tagName
                != 'form'){
                field.value = 'myPwd1234';
            }
            else{
                console.log('No␣email␣or␣password␣fields␣to␣fill
                    ');
            }
        };

        var resultList = []; const observeTargets = function(){
            // Specify root document node as target node to
                observe full page
            const targetNode = window.document.documentElement;
            const config = {attributes:true, childList: true,
                subtree:true};
            const callback = function(/** @type {MutationRecord
                []} */ mutationList, /** @type {MutationObserver}
                */ observer){
                mutationList.forEach( function(mutation){
                    for(var i=0; i<mutation.addedNodes.length; i
                        ++){

                        if(mutation.addedNodes[i].nodeName == "#
                            text" || mutation.addedNodes[i].
                            nodeName == "#comment" ){
                            continue;
                        }
                        // @ts-ignore
                        var forms = mutation.addedNodes[i].
                            querySelectorAll('form,␣input');

                        if (forms.length == 0){
                            continue;
                        }
                        var elemDimension = forms[i].
                            getBoundingClientRect();
                        var parentFormAction = forms[i].form.
                            action;
                        //Push the mutation to the result list
                            together with its position and action
                        resultList.push(forms[i].outerHTML, ("
                            Form␣element␣width:␣" + elemDimension
                            .width), ("Form␣element␣heigth:␣" +
                            elemDimension.height),
                            parentFormAction);
                        //Call the function to fill the input
                            fields
                        setTimeout(fillInputFields,1000, forms);
                    }

                });

            };
            //Create the MutationObserver object to initiate
                callback
            const observer = new MutationObserver(callback);
            // Start observing target node
            observer.observe(targetNode,config);
        };


        //Start observering targets when the whole DOM content
            has been loaded
        window.addEventListener('DOMContentLoaded', (event) => {
            observeTargets();
        });
        `;
        await cdpClient.send('Page.
            addScriptToEvaluateOnNewDocument', {source: `console.
            log("INJECTED␣SCRIPT")`});
        await cdpClient.send('Page.
            addScriptToEvaluateOnNewDocument', {source:
            SOURCE_STRING});

    }

    /**
     * @param {Options} options
     */
    async getData(options) {
        this._options = options;
        this.page = options.page;
        this.finalUrl = options.finalUrl;
        var result = await this.page.evaluate(`resultList`);
        this._insertedNodes.push(result);
        return this._insertedNodes;
    }

}

module.exports = MutationsCollector

/**
 * @typedef {number} NodeId
 */

/**
 * @typedef Options
 * @property {string} finalUrl
 * @property {function(string):boolean} urlFilter?
 * @property {puppeteer.Page} page
 * @property {string} outputPath
 * @property {puppeteer.BrowserContext} context
 */

/**
 * @typedef NodeData
 * @property {Node} node
 *
 */
```

Likely encoding and hashes used for URL and POST bodies leak detection:

```
LIKELY_ENCODINGS = [
    'base64',
    'urlencode',
    'entity',
    'lzstring',
    'custom_map_1'
    ]

LIKELY_HASHES = [
    'md5',
    'sha1',
    'sha256',
    'sha512',
    'sha_salted_1'
    ]
```