Philip Raschke, Sebastian Zickau, Jacob Leon Kröger, Axel Küpper

Towards real-time web tracking detection with T.EX - The Transparency EXtension

Citation details
Raschke, P., Zickau, S., Kröger, J. L., & Küpper, A. (2019). Towards Real-Time Web Tracking Detection with T.EX - The Transparency EXtension. In Privacy Technologies and Policy (pp. 3–17). Springer International Publishing. https://doi.org/10.1007/978-3-030-21752-5_1.

Towards Real-time Web Tracking Detection With T.EX - The Transparency EXtension

Philip Raschke, Sebastian Zickau, Jacob Leon Kröger, and Axel Küpper

Service-centric Networking, Weizenbaum-Institut, Telekom Innovation Laboratories,
Technische Universität Berlin, Germany
{philip.raschke,sebastian.zickau,kroeger,axel.kuepper}@tu-berlin.de

Abstract. Targeted advertising is an inherent part of the modern Web as we know it. For this purpose, personal data is collected at large scale to optimize and personalize displayed advertisements to increase the probability that we click them. Anonymity and privacy are also important aspects of the World Wide Web since its beginning. Activists and developers relentlessly release tools that promise to protect us from Web tracking. Besides extensive blacklists to block Web trackers, researchers used machine learning techniques in the past years to automatically detect Web trackers. However, for this purpose often artificial data is used, which lacks in quality.
Due to its sensitivity and the manual effort to collect it, real user data is avoided. Therefore, we present T.EX - The Transparency EXtension, which aims to record a browsing session in a secure and privacy-preserving manner. We define requirements and objectives, which are used for the design of the tool. An implementation is presented, which is evaluated for its performance. The evaluation shows that our implementation can be used for the collection of data to feed machine learning algorithms.

Keywords: Web tracking, browsing behavior, data privacy, browser extension, data quality, machine-learning, classification algorithm

1 Introduction

There is no doubt that our Web browsing behavior is very sensitive. The websites we visit and the content we consume reveal information about our personality, our preferences, orientations, and habits. We give away our physical addresses, our phone numbers, and bank account information to use services or order goods. Simultaneously, the majority of websites nowadays integrates content from multiple external sources or third parties. Consequently, when visiting a website (also referred as first party) these third parties are given notice about our visit the moment our browser requests the external content. While our physical address, phone number, or bank account information is not disclosed to these third parties, a link to the website we visited is.

The reasons for websites to integrate external content are manifold. Services embed images, audio, or videos without having to host or being allowed to host the content on an own server. But also many third-party scripts are integrated for various reasons. They are in particular critical, since their integration enables the execution of third-party code on the user's machine. There, they can access and gather information of the device and send it to a server where it is aggregated and analyzed. This way, a malicious third party can track every mouse movement, every key stroke, and every change of the scroll position of a user on a *different* website even without his or her awareness.

While on paper this sounds like a severe data security and privacy threat, this technique is widely used in the field of targeted advertising and Web analytics to track user behavior across multiple websites. In fact, Web trackers are an inherent part of the modern Web, because of their economic value for content providers and publishers. Websites display advertisements provided by ad exchanges or advertising networks in exchange for a payment per view or click. This way, each user of a website generates revenue.

While there is a variety of browser extensions that promise to tackle the issue, they are mostly blacklist-based, i.e. manual effort is required to identify trackers, which are then blocked (often by the domain name). This has four major disadvantages: (i) trackers can easily change their domain name, (ii) websites may offer relevant content or services, while also tracking user behavior (Amazon, Google, etc.), (iii) blacklists can be wrong, not complete, or outdated, and (iv) blocking requests to domains might create errors that prevent access to the desired content of the first party. The latter also occurs in the opposite causal direction, i.e. first parties block users from their content, if they block requests to third parties. Another conceptual flaw of blacklists is that they are not transparent themselves by providing little to no information on the third party in question and why it is blocked or not.

Consequently, an automated approach to detect Web trackers is desirable. This is a classification problem, which can be solved with machine learning techniques. However, machine learning approaches require rather large amounts of training data, which ideally is real data. However, researchers in this field often use bots to generate this data by crawling the Alexa.com top K websites. While this method produces large amounts of data rather quickly, it has a major drawback: it is not *real* data. These bots open the website, wait until it is finished loading, and then open the next in the list. These bots cannot log into websites like Facebook or Twitter, which even have implemented countermeasures for artificial users of their services. Even worse, the front page of these services are very limited and only offer a login form. It can be safely assumed that most of the third-party communication takes place after the login. By using bots, tracking of user interactions like moving the mouse, pressing a key, or scrolling is completely neglected.

For this reason, we present *T.EX: Transparency EXtension (T.EX)*, a secure browser extension to enable client-side recording, storage, and analysis of individual browsing behavior. With this tool researchers can generate data sets with

real users in a secure, privacy-preserving, and user-friendly way. In this paper, we define requirements concerning security, privacy, and usability and explain how they were met. In addition, the extension provides data visualization capabilities allowing (experienced) users to assess their browsing behavior and the third-party communication involved in it.

The remainder of this paper is structured as follows: Section 2 defines the objectives and requirements of the tool. Section 3 elaborates on the limitations and the derived design decisions. Section 4 gives an overview of related work and assesses whether suitable solutions already exist. Section 5 presents the implementation of the tool. In Section 6, we evaluate the tool with regard to the specified objectives. Finally, a conclusion is given including an outlook.

2 Objectives and requirements

As stated above, the main objective of the tool is to enable the generation of *real* user data in a secure, privacy-preserving, and user-friendly manner by allowing users to record browsing sessions. On this basis, we derive the following objectives:

Obj1 The tool needs to be able to monitor Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) traffic, including header information, parameters, and the body.

Obj2 An accurate differentiation between first and third party must be realized. The first party should not be identified only by its host name but rather by the actual page (HTTP path) the user visited.

Obj3 The network traffic must be persistently stored for a certain amount of time. This data must be securely (i.e. encrypted) stored on the user's device, so no other (malicious) software on the user's machine can access it.

Obj4 The extraction of data must be in a privacy-preserving manner, i.e. only relevant data should be collected. Furthermore, no external servers must be involved.

Obj5 The user must be able to completely delete the data at any time. There should be a means to prove the erasure of the data.

Obj6 Furthermore, the user must be able to export the data in a machine-readable format.

Obj7 The user must be able to disable the recording of network traffic at any time. Ideally, the user can be given a guarantee or proof that the recording is stopped.

Obj8 Usage of the tool should be user-friendly to the extent that the perceived Quality of Experice (QoE) is not impacted by it.

Obj9 The tool must offer data visualization capabilities so that users can review the recorded data before they export it. A search function enables users to check if any sensitive information are contained within the data set.

3 Limitations

Unfortunately, the above defined objectives cannot be realized without constraints. In this section, we infer limitations from these objectives and elaborate on consequent design decisions for the tool.

In order to realize Obj1, HTTP and HTTPS traffic needs to be intercepted. Obviously, this is a severe data security risk and infringement of the user's privacy. For this reason, the collected and recorded data must remain on the user's device (see Obj4). However, intercepting HTTPS traffic on the network layer is not possible without aggressive intervention. A *man-in-the-middle* attack could be used in order to intercept the encrypted traffic, but this would put the user's overall data security at risk.

Fortunately, we can rely on capabilities offered by browser vendors. Experienced users or system administrators have the expertise to obtain the data using the browser's developer tools like Google Chrome's *DevTools* or the *Inspector* of Firefox. However, the data, that is logged there, is separated from other browser sessions (tabs). Consequently, for a holistic view, an aggregation of the data is required. The user would need to open the corresponding tool before the begin of each browsing session in each tab. The log is cleared with every new page the user visits, so a checkbox needs to be ticked to persist the log (in each tab). To export the recorded data, only Firefox' *Inspector* offers a complete export of the data, while Chrome's *DevTools* only offer an option to export one request at a time. Collecting data using this method is cumbersome and error-prone, which violates Obj8. Further inspection of this method also revealed that Obj2 is violated, since the exported data either does not contain the first party (Chrome) or only gives the host name of it (Firefox).

Clearly, a more sophisticated method is required. Luckily, HTTP and HTTPS traffic can be logged using Chrome's or Firefox' extension Application Programming Interface (API). So, Obj1 can be best implemented in a browser extension. In fact, we found no alternative approach to realize Obj1 without aggressively interfering with the user's device. Using the extension API also allows us to identify the first party including the HTTP path (see Obj2). Besides an *initiator* field in the traffic log, it is possible to map a request to a certain open and active tab of which the URL can be used.

To persistently store the data like stated in Obj3, a sophisticated database like *MySQL* or *MongoDB* would be ideal, however this would require users to install additional software on their device (violation of Obj8) or to transmit the data to an external server (violation of Obj2). Browser extensions are able to store data in the so-called *local storage*, which offers limited storage capabilities. The local storage is a key-value store, thus complex queries cannot be easily expressed. Furthermore, the local storage is not encrypted, thus malicious software on the user's device could easily gain access to it. Therefore, encryption must be implemented within the browser extension. However, inconvenient key-pair generation and management must be avoided in order to not violate Obj8.

In order to realize a collection of data in a privacy-preserving manner (Obj4), only the outgoing traffic is recorded. This way, we follow a data minimization

approach. The HTTP response, besides the actual content the user consumes, contains cookies and identifiers that are assigned to the user and which are used for subsequent requests. By neglecting the HTTP response, we miss these assignments. However, we assume the preserved privacy is of higher value than the benefit gained from the HTTP responses. Moreover, it is not sure whether the accuracy of a classification algorithm to detect Web trackers would be increased if the HTTP response is taken into consideration. It would be interesting to investigate this in a separate study.

Since the HTTP body is used to transmit sensitive data like passwords, messages, photos or videos, recording it can be highly sensitive. Therefore, it is not recorded by default but the user is able to enable this feature at own risk. The reason why we do not completely exclude it, like we do with the HTTP response, is that we could observe Web trackers using it for passing identifiers to their servers.

The local storage can be cleared at any time; therefore, the user is given a button to trigger the erasure of all data (Obj5). Moreover, the local storage is file-based, i.e. its content can be found in plain text in files on the user's machine. Thus, to ensure the erasure of all personal data, the user can additionally delete the corresponding files. The path to these files is static, it can be given to the user so he or she can find it.

To export the data in a machine-readable format (Obj6) the whole local storage must be queried, requests must be decrypted, and saved to a dedicated file. Since data in the local storage is in JSON format, it is reasonable to export it as such. Due to the diverse structure of the recorded data, an export in CSV is rather unhandy.

Disabling the recording (Obj7) can be realized with a set of means: by implementing blacklists (or whitelists), by offering a button to start and stop recording at any time, or by disabling the extension completely. The latter is undoubtedly the safest and easiest way to guarantee that the recording is disabled. Blacklists or whitelists determine on which websites recording should be disabled or enabled respectively. This approach, however, requires users to invest some effort for preconfiguration, which might violate Obj8. A button to start and stop recording is rather easy to implement, but offers no advantage compared to enabling or disabling the extension, since this can be triggered with one click as well.

To achieve Obj8, all other objectives must be realized by involving as less user effort as possible. This means that the usage of the extension itself is realized in a user-friendly manner. But furthermore, the usage of the extension should not impact the perceived QoE while browsing the Web, i.e. websites should not take longer to load or that CPU and memory consumption drastically increase so that other applications are affected.

The visualization of the data (Obj9) can be done in the browser using Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, and Scalable Vector Graphics (SVG). To highlight the communication flows, we chose a graph representation of the data. A search function is provided to users

allowing them to query the data for personal information they do not want to be included in a resulting data set, which is further processed.

4 Related work

Trackers enjoy a long presence in the history of the Web. In fact, they exist almost as long as the Web itself. Lerner et al. [11] proved the presence of Web trackers in 1996 by examining and analyzing the Web Archive. The Internet, as a distributed system, is built upon interconnections of nodes, thus, third parties are conceptually nothing to despise. However, for the precise personalization of displayed advertisements, personal data is required, which is often collected without a users awareness using Web tracking techniques. One could argue that the most severe issue with third-party content is not its presence but users unawareness of it. A study by Thode et al. [14] shows that users' expectations regarding third-party tracking heavily differ from reality. With the General Data Protection Regulation (GDPR) [7] coming into effect in May 2018, this circumstance becomes problematic, since it requires the processing of personal data to be transparent.

Bujlow et al. [2] published a sophisticated survey on all known Web tracking techniques to date. Most modern and often more accurate methods mostly rely on third-party scripts that are executed on the user's device to obtain a set of data items to generate a so-called *browser fingerprint*, which is sufficient to uniquely identify the user among other users.

Today Web trackers are subject to extensive studies due to the threat they impose on our data privacy. A very sophisticated study was conducted by Englehardt et al. [6] in 2016, who aimed to measure and analyze the extent of third-party presence on one million websites. Therefore, they designed and developed the tool OpenWPM to measure and record HTTP traffic. Yet, OpenWPM uses Selenium to crawl the top one million websites, which is a framework to simulate and automate user interactions. Thus, their measured data is not real user data. Regardless of the data quality, they found third-party scripts present on nearly all considered websites. Their results further show that only few third parties are present on a high number of first parties. This is clear evidence for data monopolies of the most prominent Web trackers. However, this circumstance is also an advantage: one has to identify and block the few most prominent third parties only to effectively protect oneself from Web tracking on the most popular websites at least. This is one of the reasons why the blacklist-based approach is so popular: it is very effective.

There are many browser extensions for all major browsers that follow this approach. Their promise is to protect users from unintended and unauthorized third-party information disclosure. Browser extensions like Ghostery [10], UltraBlock - Privacy Protection & Adblocker [16], Crumble [4], or Privacy Badger [13] are very popular tools with millions of users. However, only Privacy Badger tries to identify Web trackers based on their prominence in addition to blacklists. Privacy Badger blocks a third party if its presence is observed on three distinct first

parties. An additional challenge of these browser extensions is to maintain the same level of user-perceived QoE after the extension has been installed. From a user's perspective, blocking third-party requests is very beneficial, since loading times are decreased and computing resources are spared, as a study of Kontaxis and Chew [8] confirms.

However, the above presented browser extensions give little to no information on the tracking third party itself nor technical details about the process of data exposure. However, there are browser extensions that give more information: uMatrix [17] and uBO-Scope [15]. The extension uMatrix provides the user with insights on the type of HTTP requests issued to the corresponding third parties. While, to our knowledge, the extension uBO-Scope is the only one that accurately gives information on the extent of presence of a specific third party during the current browsing session. A high presence of a third party is indicated with red in the extension's pop-up window.

Nonetheless, all the above presented browser extensions rather aim to identify and block tracking activities than serving as tool to assess data flows to third parties. They offer limited data visualization capabilities and no recording options, which makes it difficult to analyze or further process the measured data. The browser extension closest to the objectives of T.EX is Firefox' Lightbeam [9], which has strong visualization features (Obj9), but fails to give more insights on the communication that has taken place and the third parties itself (Obj1). Lightbeam allows to export the recorded data in machine-readable format (Obj6), yet the the exported information does not include the first party with its HTTP path (Obj2).

The idea to use machine-learning techniques to identify Web trackers was proposed by Bau et al. [1] in 2013. They elaborate on useful data sources and how to obtain labeled training sets. Following the paper's position, there were several publications of researchers in the following years describing supervised or unsupervised classification of Web tracking activities. In 2014, Metwalley et al. [12] present an unsupervised approach that leads to successful results. Their algorithm is able to detect 34 Web trackers that have never been documented before. Similar results are achieved by Wu et al. [18] in 2016. They use a supervised approach and detect 35 new tracking parties. Despite their successful revelation of new Web trackers, both research groups use crawlers to generate the data with which they feed their machine-learning algorithms.

The importance of proper data quality is highlighted by the publication of Yu et al. [19], who achieve remarkable results with regard to accuracy and performance of detecting Web trackers. The authors are a research group from the Cliqz browser development team, which is a German browser vendor of the samenamed browser Cliqz [3]. Through their product, they were able to use browsing data of 200.000 users for their algorithm. This way, they were able to outperform their commercial competitor Disconnect.me [5], which is also used by Firefox.

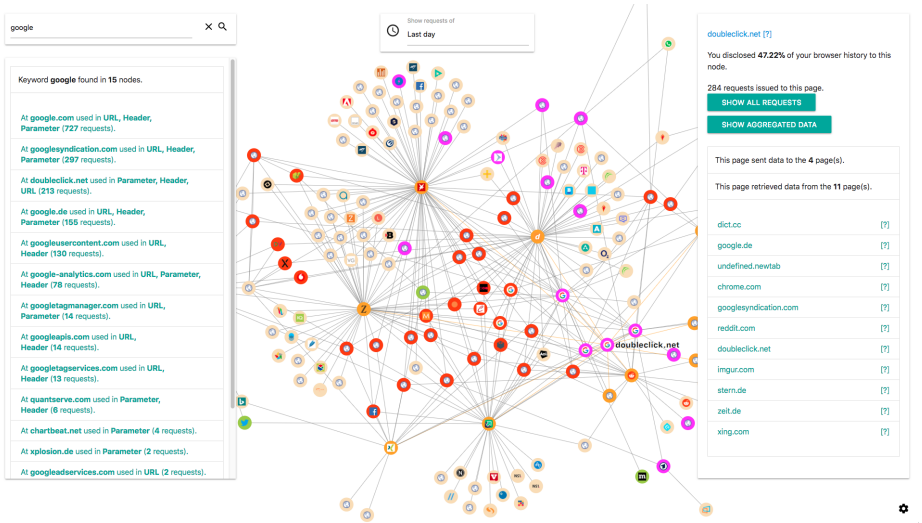

Fig. 1. The user interface of the browser extension including a graph, a search feature, and further information on the third parties. Highly connected nodes are colored red to indicate third parties with high extent of presence on other websites.

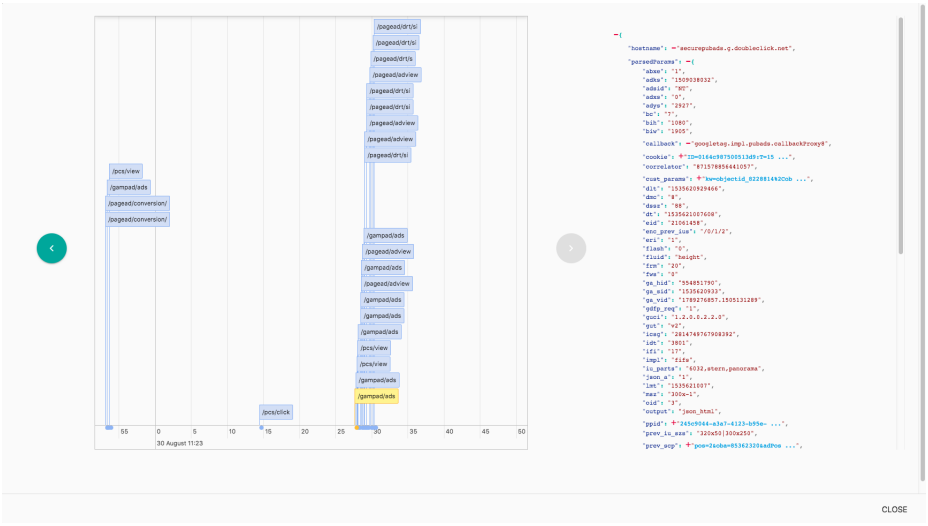

Fig. 2. Records visualized on a timeline enabling users to investigate requests initiated by a certain website to a certain third party. By selecting an event, users can see the corresponding record including all recorded data.

5 Implementation

This chapter presents the implementation of T.EX and explains how the individual objectives were realized. T.EX has been implemented for Google Chrome, however it is planned to port the implementation to Mozilla Firefox. Since the offered browser extension APIs of the two browser vendors are based on the WebExtension APIs, it can be expected that most of the code can be reused for the implementation of a Firefox extension.

5.1 HTTP and HTTPS traffic logging and recording

To intercept and log HTTP and HTTPS traffic, the interface *webRequest* is used. Chrome and Firefox emit an event *onBeforeRequest* before a request is issued. Extensions can subscribe to the event by adding a listener to it. Both browsers provide extensions with valuable information on the issued request, including all necessary information on the target t of the request, search parameters S, request headers H, form data F and even data in the request body B. Interestingly, determining the source s of a request requires more effort in Google Chrome. While Firefox emits the initiator of a request in the *originUrl* field, Chrome only gives information on the source in an optional field called *initiator*. To retrieve the source even if the field is not set, a query of open tabs with the *tabId* is required. A logged event is called *record* r, which is defined as follows:

$$r \in R := (s, t, S, H, F, B) \tag{1}$$

$$kv := (key, N) \in S \cup H \cup F \cup B \tag{2}$$

$$v, kv \in N \tag{3}$$

5.2 Persistent storage of records

Records need to be persistently stored in order to enable an assessment of them later in time. The local storage of browsers is rather limited with regard to performance and expressiveness of queries. The local storage is a so-called key-value-store that allows to load values for certain keys or a set of keys, yet does not offer possibilities to query ranges. Each key has to be unique and queried explicitly. This means in practice that the local storage cannot be queried to return records that have been recorded in the last seven days for example. Furthermore, it is not advisable to get or set values in a high frequency, since the local storage can be easily overwhelmed, which directly leads to a bad QoE.

For this reason, two strategies are implemented: the aggregation of records into chunks and the writing of chunks into the local storage in a defined interval i. This way, the local storage is less demanded and the work load is evenly distributed over time. However, these strategies raise the question of appropriate keys that can be used for the chunks, so that they can be queried later in time.

To enable this, we implement a chain of chunks C, i.e. each chunk c is pointing to the last chunk and the key of the most recent chunk is stored in a global field

called *currentId*. Each chunk retrieves a timestamp ts, which is used as key for the chunk.

$$c \in C := (ts, lastId, R_{[ts-i,ts]}) \tag{4}$$

$$currentId = ts \tag{5}$$

Eventually, this implementation enables queries of chunks in a certain time range. Moreover, this implementation allows the erasure of old chunks after a predefined time. Given that the local storage by default is limited to 5.24 megabytes, this feature is crucial. Both Chrome and Firefox have the extra permission *unlimitedStorage*. Extensions that ask for the privilege are allowed to store more data. Nonetheless, an implementation that does not rely on the permission is desirable.

5.3 Encryption and decryption of chunks

Since the local storage resides on the user's machine unencrypted, encryption needs to be implemented in order to ensure data security. Otherwise, a malicious application on the user's device could gain access to this data and gain valuable information like passwords, the browser history, email addresses, bank account information and suchlike. Without encryption, T.EX would rather constitute a severe risk than contribute to improved data security and privacy.

To implement encryption, the user is prompted to generate a key pair ($pubKey$ and $privKey$) after the installation of the browser extension. This requires the user to enter a password *pwd*. The generated private key is encrypted with the entered password using the Advanced Encryption Standard (AES). The generated public key and the encrypted private key $encPrivKey$ are then stored in the local storage.

To encrypt chunks, a random key $aesKey$ is generated that serves as symmetric key for the encryption. This random key is used for the whole browsing session until the browser is closed. This key is encrypted with the public key so that only the private key can decrypt it. This encrypted symmetric key $encAesKey$ is stored along with the encrypted chunk in the local storage. To decrypt chunks, the user is prompted to enter the password to decrypt the private key, which is then used to decrypt the symmetric key to eventually retrieve the chunks.

5.4 Data visualization

As it can be seen in Figure 1, data flows are represented by a graph $G := (V, E)$, which illustrates connections between visited websites (green-colored nodes) and involved third parties (beige or red-colored nodes). Red-colored nodes are highly connected nodes that retrieve data from various websites and Web applications. For the coloring, a rather simple rule-based approach was used for the beginning. However, it is planned to extend the coloring function at a later point in time.

Algorithm 1 Set-up and encryption of chunks

1: $privKey, pubKey \leftarrow generateKeyPair()$
2: $pwd \leftarrow$ user-entered password
3: $encPrivKey \leftarrow encrypt(privKey, pwd)$
4: $save(encPrivKey, pubKey)$
5: $c = (ts, lastId, R_{[ts-i,ts]})$
6: $aesKey \leftarrow generateRandomKey()$ for each session
7: $encAesKey \leftarrow encrypt(aesKey, pubKey)$
8: $c' \leftarrow (ts, lastId, encrypt(R_{[ts-i,ts]}, encAesKey), encAesKey)$
9: $save(c')$

Algorithm 2 Decryption of chunks

1: $encPrivKey \leftarrow$ load from local storage
2: $pwd \leftarrow$ password prompt
3: $privKey \leftarrow decrypt(encPrivKey, pwd)$
4: $c' \leftarrow$ load from local storage
5: $aesKey \leftarrow decrypt(c'_{encAesKey}, privKey)$
6: $c \leftarrow (ts, lastId, decrypt(R_{[ts-i,ts]}, aesKey)$

A more gradient color function is currently researched to highlight only the Web trackers in the graph.

$$G := (V, E) \tag{6}$$
$$V := \{r_s, r_t | r \in R\} \tag{7}$$
$$E := \{(r_s, r_t) | r \in R\} \tag{8}$$

Users can search for keywords that might appear in URLs, headers, or parameters. Purple-colored nodes (as seen in Figure 1) are nodes that contain the keyword in the record. By clicking on a node the user is able to retrieve more information on the corresponding node such as to which nodes data has been sent to or from which nodes data was retrieved. For further investigation of the occurred communication, the user can investigate requests to or from one node, which are visualized on a timeline. By selecting an entry on the timeline the record is visualized (see Figure 2).

6 Evaluation

The aim of this section is to evaluate whether the usage of T.EX implies an unneglectable impact on the user-perceived QoE while browsing the Web. Therefore, we investigate whether the loading time of a website noticeably increases, when using T.EX. We measure loading times by recording key events: *onDOMContentLoaded* and *onCompleted*. Both events occur strictly sequential, i.e. the *DOMContentLoaded*, which indicates that the Document Object Model (DOM)

is fully built, always occurs before *DOMContentCompleted*, which indicates that also all referenced resources are fully loaded and initialized. From a user's perspective, the first event occurs close to the moment when the user is able to see the website. In contrast to the latter, which is triggered when the loading indicator of the browser disappears.

Analogously, we measure the resource consumption (i.e. CPU and memory usage) during a website request and loading in order to learn the impact of the browser extension on hardware resources. For this purpose, we request and compute CPU and memory usage in a determined interval (so-called tick each 50 milliseconds). Besides CPU and memory usage, we further evaluate the disk space consumption of T.EX on a general level to find out how fast the extension reserves disk space for its purpose.

As stated above, we open websites with and without T.EX activated. We additionally repeat the procedure with a different, comparable browser extension activated in order to be able to assess the performance of T.EX in comparison with other extensions. For this purpose we identified Privacy Badger as good candidate, since it uses the same APIs to analyze traffic in real-time. However, we know that Privacy Badger decreases loading times of websites, while we expect T.EX to increase loading times. This is due to Privacy Badger preventing HTTP requests from occurring, thus saving time to load, while T.EX logs, processes, and stores HTTP requests. For both hardware resources are used. With this evaluation procedure we aim to put the increased hardware usage of T.EX into perspective.

As appropriate websites for the test, we use the German news site *spiegel.de* and the front page of *google.de*, which differ in the amount of third-party content they integrate. While accessing *google.de* triggers *only* 23 requests, which only request content from Google servers, requesting *spiegel.de* involves more than 400 requests to more than 50 third parties. We expect hardware usage and loading times to increase linearly with the number of involved requests, thus we selected two websites that are rather bipolar in that respect. The experiment was conducted on a machine with an Intel Core i7 (2.2 GHz quad-core) and 16 GB memory. The machine was connected to the Internet via a 1 Gbit Ethernet connection. The experiments were repeated three times each to detect anomalies.

The results of the experiment are depicted in Figure 3. The rows represent the corresponding runs without T.EX activated (top), with T.EX activated (middle), and with Privacy Badger activated (bottom). In each run the CPU usage (left column), memory usage (middle column), and loading times (right column) were measured.

By comparing the individual results displayed in the first column, an increase of CPU usage is clearly observable. The CPU is working much closer to capacity and maintains this level during the whole time the website is loaded. The reason for the CPU demand of T.EX is found in the steady encryption of records in the background. Thus, disabling the encryption would gain performance, yet would constitute a violation of the extension's main objectives. Additional CPU capacity is used, since requests are preprocessed before they are stored in the

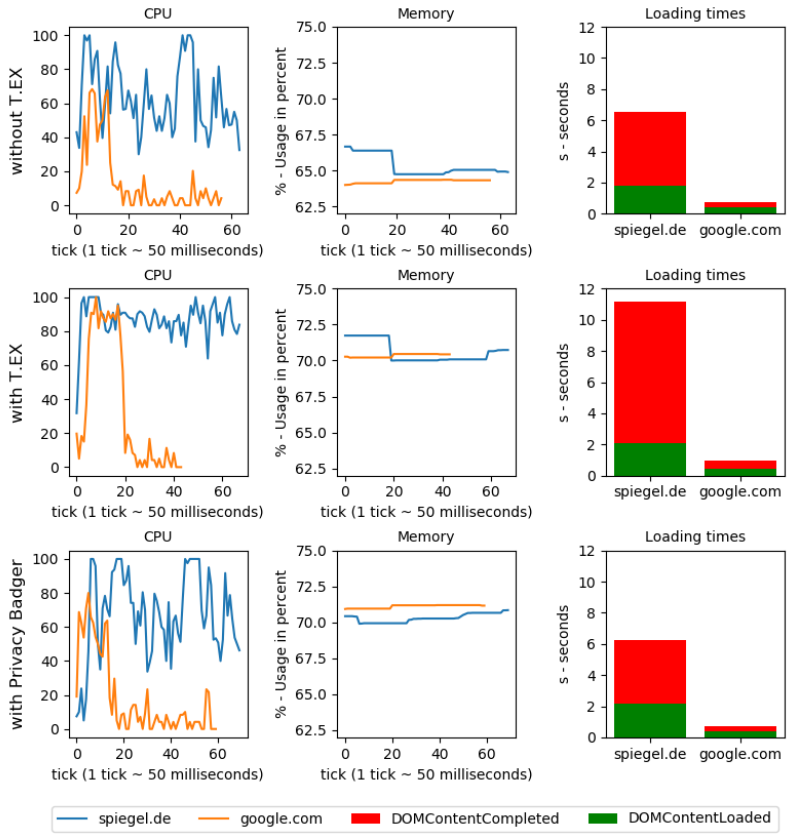

Fig. 3. The results of the evaluation: the first column shows the CPU usage, the second column the memory usage, and the third column the loading times. The first row represents the measurements without T.EX activated, the second row with enabled T.EX, and the last row with Privacy Badger activated.

local storage. This preprocessing could be executed at a later point in time, for example, when the browser is in the idle state for a certain amount of time , i.e. the browser is currently not used by the user.

The memory consumption is rather consistent with our expectation: the usage is increased fairly but not excessively. Comparable browser extensions like Privacy Badger that perform similar tasks show the same level of memory consumption. The perceived QoE should not be affected to much by this circumstance. In contrast to the loading times, which seem to be strongly affected by the usage of T.EX. When comparing the third column in Figure 3, it is noticeable that the loading time is drastically increased, when T.EX was activated. This does not apply on the *DOMContentLoaded* event, but on the *DOMContentCompleted* event. Note that the page is usable much earlier, so that the user can already interact with it, before the DOM content is fully loaded. Yet the performance of T.EX with regard to loading times requires improvement. It is also noteworthy that the performance for the loading times of *google.de* are comparable to the performance achieved in the other runs. Consequently, the drastic increase of the loading time occurs on websites with massive third-party involvement. An exponential increase relative to the number of involved third parties could be ruled out.

Finally, we aim to investigate the disk space consumption. While it can be measured easily by simply checking how big the local storage files are, it is rather difficult to define a rule to estimate the storage usage. In general, it heavily depends on the usage and browsing behavior of the user. In a dedicated three-hour lasting session, we were able to collect 80 megabyte of data, while on a different machine that is exclusively used during office hours (then extensively), we collected almost 700 megabyte in a single month. Nonetheless, it must be stated that the storage requirements imposed by the usage of T.EX exceed the requirements of other browser extensions. Therefore, users of T.EX must be aware that the recording of browsing sessions is storage intensive.

7 Conclusion & outlook

This paper presents T.EX a browser extension to provide transparency to experienced users or system administrators, who want to record and analyze communication flows to external third parties while browsing the Web. Therefore, objectives and requirements have been defined and their implementation has been presented. T.EX will serve as tool to conduct measurements and obtain real user data in a secure and privacy-preserving manner, which might contribute to more accurate machine learning models to identify Web trackers and tracking activities in real-time. We evaluated T.EX by measuring its impact on the performance to derive consequences on the user-perceived QoE. Our results show that T.EX achieves performance, which is comparable to other privacy browser extensions like Privacy Badger. However, it has an impact on the loading times of certain websites that cannot be neglected. The issue will be investigated in

future works. Furthermore, we will use T.EX to collect data that will be used to identify trackers and their tracking activities.

Acknowledgments

Supported by the European Union's Horizon 2020 research and innovation programme under grant 731601.

References

1. J. Bau, J. Mayer, H. Paskov, and J. Mitchell, A Promising Direction for Web Tracking Countermeasures, W2Sp, 2013.
2. T. Bujlow, V. Carela-Espanol, B. R. Lee, and P. Barlet-Ros, A Survey on Web Tracking: Mechanisms, Implications, and Defenses, Proceedings of the IEEE, vol. 105, no. 8, pp. 14761510, 2017.
3. Cliqz - Der sichere Browser mit integrierter Schnell-Suche [Online]. Available: `https://cliqz.com/` [Accessed: 4-Feb-2019].
4. Crumble Online Privacy, Stop Tracking. [Online]. Available: `https://chrome.google.com/webstore/detail/crumble--online-privacy/icpfjjckgkocbkkdaodapelofhgjncoh`. [Accessed: 4-Feb-2019].
5. Disconnect [Online]. Available: `https://disconnect.me/`. [Accessed: 4-Feb-2019].
6. S. Englehardt and A. Narayanan, Online Tracking, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS16, no. 1, pp. 13881401, 2016.
7. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), OJ L 119, 4.5.2016, p. 1-88.
8. G. Kontaxis and M. Chew, Tracking Protection in Firefox For Privacy and Performance, In IEEE Web 2.0 Security & Privacy, Jun. 2015.
9. Firefox Lightbeam Add-ons for Firefox. [Online]. Available: `https://addons.mozilla.org/de/firefox/addon/lightbeam/` [Accessed: 4-Feb-2019].
10. Ghostery Makes the Web Cleaner, Faster and Safer! [Online]. Available: `https://www.ghostery.com`. [Accessed: 4-Feb-2019].
11. A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016, Usenix Security, 2016.
12. H. Metwalley, S. Traverso, and M. Mellia, Unsupervised Detection of Web Trackers, in 2015 IEEE Global Communications Conference (GLOBECOM), 2014, pp. 16.
13. Privacy Badger — Electronic Frontier Foundation. [Online]. Available: `https://www.eff.org/privacybadger`. [Accessed: 4-Feb-2019].
14. W. Thode, J. Griesbaum, and T. Mandl, I would have never allowed it: User Perception of Third-party Tracking and Implications for Display Advertising, Re:inventing Information Science in the Networked Society. Proceedings of the 14th International Symposium on Information Science (ISI 2015), Zadar, Croatia, 19th–21st May 2015, vol. 66, no. May 2015, pp. 445456, 2015.
15. uBO-Scope: A tool to measure over time your own exposure to third parties on the web. [Online]. Available: `https://github.com/gorhill/uBO-Scope`. [Accessed: 4-Feb-2019].

16. UltraBlock - Block Ads, Trackers and Third Party Cookies. [Online]. Available: https://ultrablock.org/. [Accessed: 4-Feb-2019].
17. uMatrix: Point and click matrix to filter net requests according to source, destination and type. [Online]. Available: https://github.com/gorhill/uMatrix. [Accessed: 4-Feb-2019].
18. Q. Wu, Q. Liu, Y. Zhang, P. Liu, and G. Wen, A machine learning approach for detecting third-party trackers on the web, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9878 LNCS, no. 4, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds. Cham: Springer International Publishing, 2016, pp. 238258.
19. Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol, Tracking the Trackers, in Proceedings of the 25th International Conference on World Wide Web - WWW 16, 2016, no. AUG., pp. 121132.