Systematic Analysis of Browser History Evidence

Tobias Groß, Richard Dirauf and Felix Freiling
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Department of Computer Science, D-91058 Erlangen, Germany
Email: {tobias.gross,richard.dirauf,felix.freiling}@fau.de

***Abstract*—Traces of browser usage are an important piece of digital evidence in many cases. In the literature, it is usually assumed that the entries in the browser history and the browser cache reliably indicate which URL was accessed and at which time this was done. Using the market leaders Google Chrome and Mozilla Firefox as examples, and comparing our results with older versions of Internet Explorer, we show that this exact correspondence between stored URL and real URL on the one side and the stored timestamp and the real time of the action is not always true. On the contrary, it is rather common that browsers record the timestamp of a user action several seconds after the action really happened. It can even happen that browsers sometimes record a different domain from the domain that was actually visited. The basis for our insights was a large scale experiment using an automatic deployment of virtual machines, resulting in a dataset of considerable size.**

I. Introduction

Despite the increasing trend towards personal mobile devices, web browsers are still a dominant method for users to perform communication and access information in the Internet. Therefore the evidence generated by web browsers is also important in many criminal investigations. A common investigative questions is whether a named user actually visited a certain website at a particular time. Answers to this question can help to explain motivation (e.g., visiting extremist websites or downloading propaganda), prove or disprove communication (e.g., communicating with other users in forums or via web mail), or to check for alibis (e.g., user activity in time periods of claimed absence).

The main sources of evidence in web browsers are the *browser history*, a common feature to allow forward/backward navigation to visited websites, and the *browser cache*, a software component that stores recent web content for performance reasons. To use this data as evidence, it has to be (1) extracted from the concrete browser installation in question and then (2) interpreted correctly to make proper statements. While much is known on how to extract data from browser history and cache for common browsers like Mozilla Firefox and Google Chrome, much less is known on how to reliably interpret this data.

A. Related Work

Early work on the forensic analysis of web browsers focussed on Microsoft's Internet Explorer, for a long time the dominant browser on the web. For example, both Jones [1] and Jones and Belani [2] analyzed the structure of Internet Explorer activity files and developed the tool *Pasco* to parse these files. Boyd and Forster [3] focused on parsing the different date and time structures in these files. The interpretation of this data is rather straightforward. For example, Jones and Belani [2] refer to the access timestamp in the browser history as follows:

> "Access Time - The moment in time the user browsed the website."

Similarly, Boyd and Forster [3] states that the timestamps in the activity files

> "[...] represent the UTC date and time that the corresponding URL was last visited using Internet Explorer."

Other work continued to focus on the technical side of evidence collection. For example, for Firefox, Pereira [4] analyzed history artifacts of version 3 and also proposed a method to recover deleted entries. Mahaju and Atkison [5] instead, compared different tools for analyzing Firefox browsing artifacts, and Oh et al. [6, 7] developed a new methodology to collect and analyze evidence from different browsers and to integrate all evidence in a single timeline. Sonntag [8] also developed a tool which can extract and collect data from Internet Explorer, Chrome and Firefox. While they mention that the history of Internet Explorer may contain spurious elements which were not intentionally visited by a user and that Firefox and Chrome history entries have attributes which allow to distinguish intentional from unintentional visits, they do not investigate this systematically. Similar to the early work on browser forensics, the interpretation of the collected data is rather plain. For example, Sonntag [8] writes that

> "[the] actual sequence of visits is stored in moz_historyvists where for each visit [...] a timestamp of the visit [is listed.]"

and Pereira [4] states that

> "[since] moz_historyvisits table also records the date and time for each visit [visit_date field], it is possible to determine the complete history access of each URL."

More recent work either moved its focus to Google Chrome [9, 10, 11] or increased the amount of data extracted from the browser. For example, Horsman [11] showed that video streams played by Google Chrome are cached in a fragmented way on persistent storage, but can often be reassembled resulting in a complete video. Other work investigated artifacts left by HTML5 web applications within the WebStorage API, e.g., opened tabs and windows [12, 13]. Finally, several

publications [e.g. 14, 15, 16, 17, 18, 19] addressed the privacy implications of portable browsers and the provided private browsing modes. And Joseph et al. [20] showed how to extract user credentials, visited sites and search strings from volatile memory (with the possibility of false positives when searching for URLs).

B. Contributions

Overall, it appears that the main thrust of all the work we are aware of predominantly concentrates on data extraction and takes data interpretation for granted. After all, it seems obvious that if the browser history records the access of URL x with timestamp t then in fact the browser accessed website x at real time t. The only circumstances that have been formulated in the literature and that might question this interpretation is explicit tampering of browser data, a task that was shown to be hard [21].

In this paper, we study whether even under normal circumstances (standard software installation, no tampering) the browser history data is a reliable and accurate representation of a user's browsing behavior. More specifically, we ask the question to what extent the URL and the timestamp recorded in the history in fact match reality. Our contributions are as follows:

- We systematically created browser evidence by visiting over 50 popular websites multiple times over a long period of time with Chrome, Firefox, Internet Explorer 7, 8 and 9, resulting in a total amount of 1767 website visits.
- We evaluated the correlation of the generated data and the user's action in terms of temporal correlation, correlation of URLs and quantity of data entries.
- In doing so, we show that the timestamps extracted from the browser history actually often deviate from the time of the user's action, but that the deviation is usually small (in the range of seconds). On average 90% of timestamps are within a 10 second offset.
- We also show that in exceptional cases (only three websites, and mainly for Internet Explorer) the URL recorded in the browser history *does not* match with the URL of the actually visited website. By correlating the accessed URL to URLs found in the browser cache, we even show that 69% of cache entries in our dataset refer to a completely different domain than the originally visited URL contained.

Overall, our results show that the interpretation of browser evidence has to be done with care and that in critical cases statements regarding this evidence have to be backed by experiments.

C. Roadmap

This paper is structured as follows: In Section II we describe our setup for creating browser evidence with Chrome, Firefox and Internet Explorer. In Section III we analyze the generated data in history files of all browsers and how the data correlates with the user action. In Section IV we do the same for all generated cache data. We discuss how the results can be used to either support or attack testimony regarding browser history in legal proceedings in Section V and conclude the paper in Section VI.

II. Background and Methodology

A. Investigated Browsers and Their Relevance

To get a broad overview over the evidence generated by desktop browsers we chose to investigate the market leaders Google Chrome and Mozilla Firefox. With a mean market share of 68% for Chrome and 11% for Firefox they were the most used desktop browsers worldwide in 2018 [22].

Because of its former importance but mainly for historic reasons, we also used older versions of Microsoft Internet Explorer. In historic market shares, Internet Explorer version 7 had a mean share of 15%, Internet Explorer version 8 had a mean share of 27% in 2010 [23]. In 2012 Internet Explorer 9 gained a mean market share of 14% [24]. Table I summarizes the different system configurations used within our study.

TABLE I
Configurations of systems used for evidence generation

Name	OS Name	OS Version	Browser Version
Chrome	Windows 10 Enterprise	10.0.17763	75.0.3770.100 75.0.3770.142
Firefox	Windows 10 Enterprise	10.0.17763	67.0.4 68.0
IE 7	Windows Vista Enterprise	6.0.6002 SP2	7.0.6002.18005
IE 8	Windows 7 Enterprise	6.1.7601 SP1	8.0.7601.17514
IE 9	Windows 7 Enterprise	6.1.7601 SP1	9.0.8112.16421

More specifically, we used the versions of Internet Explorer 7 and 8 that were shipped together with the respective versions of Windows. In the case of Internet Explorer 9 we decided to update version 8 with a installer provided by Microsoft, because we had no Windows version available where Internet Explorer 9 is preinstalled.

B. Relevant Evidence

Files of particular interest in this work are the browser cache and history. We want to investigate which data gets created in these files when a user visits websites with the browsers. For all Internet Explorer versions we determined the path of the history and cache files by comparing the system image before and after the automation (which we explain below).

We identified the browsing data files which get modified or created during the Internet Explorer usage. They are enumerated in Table II. All these identified files are in the Internet Explorer History File format which was analyzed by Jones [1] who also developed the *pasco* tool which we used to parse the IE data content in this work. For our further investigation of the Internet Explorer history traces, only the main history files are of relevance, because the other history files sometimes get created but have no entries included in our automation runs. They are shown in Table II and marked with *. Sometimes entries are written in the *Low* history and sometimes in the

TABLE II
IDENTIFIED INTERNET EXPLORER BROWSER DATA FILES

Feeds Cache	Users\<User>\AppData \Local \Microsoft \Feeds Cache\index .dat
Cookies	Users\<User>\AppData\Roaming\Microsoft\Windows\Cookies\index.dat Users\<User>\AppData\Roaming\Microsoft\Windows\Cookies\Low\index.dat
Cache	Users\<User>\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\index.dat Users\<User>\AppData\Local\Microsoft\Windows\Temporary Internet Files\Low\Content.IE5\index.dat
History	Users\<User>\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist<dateinfo>\index.dat Users\<User>\AppData\Local\Microsoft\Windows\History\Low\History.IE5\MSHist<dateinfo>\index.dat * Users\<User>\AppData\Local\Microsoft\Windows\History\History.IE5\index.dat * Users\<User>\AppData\Local\Microsoft\Windows\History\Low\History.IE5\index.dat

normal history. We do not know why IE operates like that. Feeds Cache and Cookies files are out of scope in this work.

We identified the path of Firefox and Chrome browsing data by visiting `about:support` or `chrome://version` in the respective browser. In our test setup, Firefox stores the data in `Users\<User>\AppData\Roaming\Mozilla\Firefox\Profiles\<profile>.default` and Chrome in `Users\<User>\AppData\Local\Google\Chrome\User Data\Default`. Firefox uses an SQLite database as history file which is stored in `<profile>\places.sqlite` and the cache data in `<profile>\cache2` [25]. In this work we used FirefoxCache2 parser [26] to parse the content of the cache files.

Chrome stores the history also as an SQLite database in the path `<profile>\History` and the cache data in `<profile>\Cache` [27]. We used ChromagnonCache [28] to parse the Chrome cache files.

In the automation framework (which we explain below) for Firefox and Chrome we had the chance to define a path where the browsing data gets stored. To verify that the created artifacts are the same as if browsing manually, we tested it for three websites. Afterwards we compared the resulting artifacts. In the history files, all entries were identical in the test cases except the timestamps, obviously because the test were executed at different moments. Also, some history entries which get created in the automated process by Firefox are falsely typed as visited by link instead of visited by the address bar. The respective field in the entry will be discussed later in this paper.

We cannot systematically compare the cache entry-wise because the cache is dependent on the content displayed by a webpage which nowadays is highly dynamic and differs from visit to visit. Overall it seems that the cache behaves the same in manual and automated session and we have no reason to assume that the cache is treated different when the browser is used by the automation framework.

C. Experimental Setup

1) Chrome and Firefox: For Chrome and Firefox the setup differs from the Internet explorer setup. We use Open Nebula to define our systems for evidence generation and QEMU as underlying hypervisor and deployed a dedicated system for Chrome and Firefox. Such a system consists of Windows 10 which we installed on a QCOW image v3, a respective browser and the software for automation. The virtual machine is defined with the libvirt domain[1] format and has 2 CPUs, 4GB RAM and 50 GB disk storage. On the system we installed either Firefox or Chrome in the version shown in Table I. During our automation runs the browsers updated automatically to the versions listed in the table, too.

We chose to use a python module called Splinter [29] to automate Firefox and Chrome. This python module provides functions to visit sites or interact with elements on the sites, e.g. clicking a link. Splinter relies on Selenium [30] and special drivers as a bridge to interact with Chrome and Firefox. We used geckodriver v0.24.0 [31] for Firefox and chromedriver v74.0.3729.6 [32] for Chrome.

Since Splinter/Selenium deletes the complete temporary user profile (e.g. History, Cache etc.) after the automation by default, we had to alter this behavior. For Chrome we had the chance to define a custom directory for the user profile, which doesn't get deleted after the automation process. For Firefox we modified the Splinter code which handles the automation of Firefox. We removed the code which creates an instance of Selenium's `FirefoxProfile` class and added an *profile* argument to the geckodriver call instead. This modification allows to preserve the user profile after the automation process.

2) Internet Explorer: For each of the investigated Internet Explorer versions (see Table I) we created a disk image (a bundle of OS and browser) that we used as virtual machines to systematically generate browser evidence. We used VirtualBox as virtualization environment and stored OS/browser bundles as a pair of files: the system was stored in open virtualization format (OVF) [33] and the disk as virtual machine disk (VMDK) [34]. Additionally, the VirtualBox Guest Additions were installed on all systems to allow more control of the VM guest.

The setup of these systems was conducted with VirtualBox 5.2.8, where the Windows installation disks were mounted from images. After the setup, browser data like Temporary Internet files and website files, Cookies and website data,

[1]https://libvirt.org/formatdomain.html

History, Download History, Form data, Passwords and Tracking Protection data was cleared with the function *Browsing history→Delete...* in the option menu. In the case of Internet Explorer 9 system image, we installed the update, before clearing the history.

We generate our evidence by using the AutoIt framework [35] to implement a browser automation executable. The executable takes 2 arguments defining the site to open and the link which will be followed. The automation opens a site by pressing *<ctrl + o>* then typing the sites URL and pressing *<return>*. The link clicking action is implemented as searching for the link text in the UI object pool and performing a left mouse click on the found object.

D. Evidence Generation

1) Chrome and Firefox: The following list describes the steps performed for Chrome and Firefox to generate the browser data which we want to evaluate. This process is also shown in Figure 1. We repeated these steps 10 times in different points in time during June 27th and July 16th, 2019.

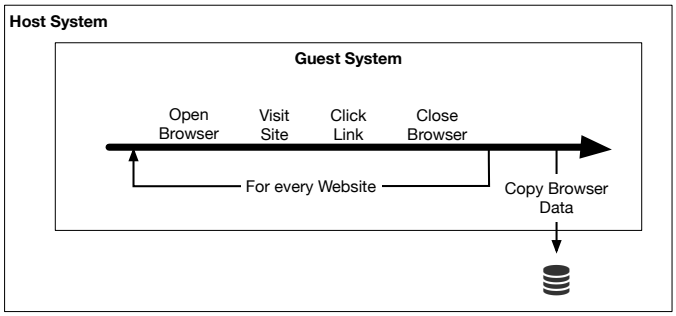

Fig. 1. Chrome/Firefox evidence generation process.

- Open browser via Splinter with custom user data path which has no data from previous sessions.
- Open website via `visit` function of Splinter and wait two minutes.
- Click link on the site which leads to another site on the same domain. The specific links get identified with the Splinter function `find_by_text` and the link element get clicked with the `click` function of Splinter. Wait four minutes after the action.
- After all 50 websites are processed, copy browser data from automation system to the evaluation system.

2) Internet Explorer: To study the evidence generated by executing a particular action within a browser, we set up the VM image under consideration within VirtualBox and repeatedly performed the following four steps, which are also shown in Figure 2:

- Boot system: The host system boots the guest system in headless mode meaning that no graphic output gets displayed. For debugging, a remote desktop connection can be used. The boot process is monitored by the VM host with the VirtualBox Python API. The trace generation continues only when the system booted completely.

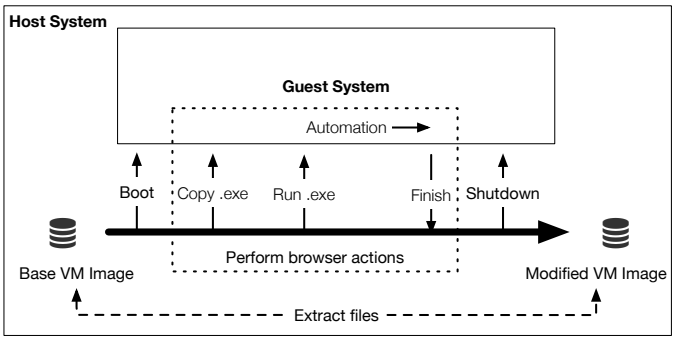

Fig. 2. Internet Explorer evidence generation process.

- Perform browser actions: We copied the appropriate AutoIT executable into the guest system with the Python API and the guest additions. The VM host starts the executable using the Python API and provides two parameters: The URL entered into the browser bar and the name of a link which gets clicked afterwards. Between starting the browser, opening a website, clicking a link and closing the browser the AutoIt program waits five minutes, to give the browser the chance to complete each step.
  The automation logs the point in time when the action (opening a site, clicking a link) took place as a ground truth. Also, a screenshot is produced after every action, that we can later check the success of a run. At the end the browser gets closed and the AutoIt program quits.
- Shutdown system: The VM host gets notified when the AutoIt process quits. Then a shutdown command gets sent to the VM. The system then performs a clean shutdown. The host monitors the system state of the guest system again with the Python API and the trace generation process continues only after the shutdown.
- Extract files: We used a modified version of fiwalk [36] to extract the history and cache files, screenshots and the ground truth from the VM image. Additionally, we created an *idifference*[2] log, which lists every file system transformation (adding, deleting, modifying files or metadata) compared to the original image.

During execution the automation program writes a ground truth, to log when the actions open website and click link happened in time. Also, a screenshot is created after each action to verify the success of the automation, later.

E. The Resulting Dataset

To generate the browsing data for our analysis we choose to visit the top 50 global sites of Alexa [38] on January 16th, 2018. With the Internet Explorer setups we encountered two issues. First, the old Internet Explorer versions where not able to render many of the modern websites. Second, our chosen automation solution for IE where not able to click links with non-ASCII characters. We choose to additionally visit the top

[2] *idifference* is a small tool included in The Sleuth Kit [37]. It outputs the difference between two filesystem images.

TABLE III
VISITED WEBSITES BY CHROME (C), FIREFOX (F), INTERNET EXPLORER 7 (7), INTERNET EXPLORER 8 (8) AND INTERNET EXPLORER 9 (9)

URL	Clicked Link	IE 7	IE 8	IE 9	Chrome	Firefox
http://google.com	Werbeprogramme (C,F), Unternehmensangebote (7,8,9)	8	7	7	10	10
http://youtube.com	Trending	-	-	-	10	10
http://facebook.com	Create a Page	-	-	-	8	10
http://baidu.com	新闻	-	-	-	8	10
http://wikipedia.org	English	13	-	-	10	10
http://reddit.com	Learn More	-	-	-	10	-
http://yahoo.com	OK - Stars (C,F), Nachrichten (7,8,9)	14	-	-	10	10
http://google.co.in	Werbeprogramme	-	-	-	10	10
http://qq.com	图片	-	-	-	10	8
http://amazon.com	Departments (C,F), Sell (8)	-	14	-	10	10
http://taobao.com	免费注册	-	-	-	10	10
http://tmall.com	请登录	-	-	-	10	10
http://twitter.com	About (C,F), Anmelden (9)	-	-	14	10	10
http://vk.com	Forgot your password?	-	-	-	10	10
http://instagram.com	Terms	-	-	-	10	10
http://live.com	Create free account (C,F), Weitere (7), Konto erstellen (8)	13	14	-	10	10
http://google.co.jp	Werbeprogramme	-	-	-	10	10
http://sohu.com	专题	-	-	-	10	10
http://sina.com.cn	English	-	-	-	10	9
http://jd.com	秒杀	-	-	-	10	10
http://weibo.com	媒体	-	-	-	10	10
http://360.cn	电脑软件	-	-	-	10	10
http://google.de	Werbeprogramme	-	-	-	10	10
http://google.co.uk	Werbeprogramme	-	-	-	10	10
http://google.com.br	Werbeprogramme	-	-	-	10	10
http://list.tmall.com	请登录	-	-	-	10	10
http://google.fr	Werbeprogramme	-	-	-	10	10
http://google.ru	Werbeprogramme	-	-	-	10	10
http://netflix.com	TRY 30 DAYS FREE (C,F), Einloggen (9)	-	-	14	10	10
http://yandex.ru	Картинки	-	-	-	10	10
http://google.it	Werbeprogramme	-	-	-	10	10
http://google.com.hk	Werbeprogramme	-	-	-	10	10
http://t.co	Learn more (C,F), API (8)	-	11	-	10	10
http://pornhub.com	Categories	-	-	-	10	10
http://twitch.tv	Browse	-	-	-	10	10
http://linkedin.com	Cookie Policy	-	-	-	10	10
http://google.es	Werbeprogramme	-	-	-	10	10
http://xvideos.com	Parents read this to protect your kids.	-	-	-	6	6
http://alipay.com	International Business	-	-	-	9	10
http://ebay.com	Electronics	-	-	-	10	10
http://yahoo.co.jp	トラベル	-	-	-	10	10
http://google.ca	Werbeprogramme	-	-	-	10	10
http://bing.com	Maps	-	-	-	10	10
http://google.com.mx	Werbeprogramme	-	-	-	10	10
http://imgur.com	I accept - Sign in	-	-	-	10	10
http://ok.ru	Registration	-	-	-	10	10
http://microsoft.com	Windows (7,9,C,F), Weitere (8)	14	13	14	10	10
http://imdb.com	Movies (C,F), Help (7,9)	6	-	14	10	10
http://wikia.com	Accept Advertising Cookies - Games (C,F), TV (7)	8	-	-	-	10
http://aliexpress.com	x - Flash Deals	-	-	-	10	10

TABLE IV
WEBSITES VISITED ONLY WITH INTERNET EXPLORER

URL	Clicked Link	IE 7	IE 8	IE 9
http://office.com	Anmelden (7), Support (8,9)	-	14	-
http://craigslist.org	AGBnew	-	-	14
http://diply.com	Humor (7,8), About (9)	14	14	14
http://espn.com	Sports	-	12	15
http://tumblr.com	Nutzungsbedingungen (7,8), Datenschutz (9)	-	-	13
http://cnn.com	Edge	14	-	-
http://chase.com	CDs (7,8), ATM (9)	10	14	9
http://pinterest.com	verwendet	11	10	14
http://nytimes.com	World	12	12	14
http://paypal.com	Konto	14	14	14
http://apple.com	Mac	14	14	14
http://yelp.com	Events	14	-	14
http://intuit.com	Products	14	14	14
http://stackoverflow.com	Questions	14	14	14
http://blogspot.com	Anmelden (7,9), Blog (8)	14	14	14
http://washingtonpost.com	Regional	14	14	12
http://walmart.com	Find	14	14	-
http://zillow.com	Rental	-	13	14
http://msn.com	Nachrichten (7), Anmelden (8,9)	14	14	14

50 sites of United States on February 9th, 2018 with the IE setups.

In total we based our analysis on 1767 successful visits whereas we define a visit successful when the initial open of a site and the link clicking action was performed correctly, what we checked with the created screenshots. For every combination of website and browser we tried to visit the site multiple times, to get more reliable data in the end. With Chrome we visited 50 different sites in 481 runs. With Firefox we also visited 50 different sites in 483 runs. With Internet Explorer 7 we visited 21 sites in 263 runs. With Internet Explorer 8 we visited 20 sites in 260 runs. And with Internet Explorer 9 we visited 21 sites in 280 runs.

Table III lists all websites of the global top 50 and the number of visits we performed per browser. Column *Clicked Link* states the link text which was clicked during the automation. The link can be different for some browsers. The abbreviation in brackets states which link was clicked by which browser. Additionally, Table IV lists all sites which we only visited with the IE setups to compensate the few successful visits of the global top 50.

III. ENTRIES IN BROWSER HISTORY

In this section we study which entries get written to a browser history when the user performs the actions browsing a site using the address bar (action *open*) and clicking a link on website (action *click*).

A. Firefox

For Firefox, all analyzed history entries are stored in the SQLite database `places.sqlite`. This database not only contains a history of all visited sites, but also bookmarks for example. We took the data for analyzing from the tables *moz_places* and *moz_historyvisits*. Thereby, we define a history entry of Firefox as the result of the inner join of both tables by the attributes *place_id* and *moz_places.id*. We choose to investigate the semantics of these attributes: *url*, *visit_count*, *hidden*, *from_visit*, *visit_date* and *visit_type*. For further analysis we interpreted *visit_count* as the amount of entries, e.g. with a *visit_count* of 2 we counted this entry as two entries.

For every run we counted how many entries get written per action. To distinguish entries created by the action *open* and *click*, we used the timestamp of an entry and the ground truth which logged the point in time when the action was performed. All entries with a *visit_date* before the execution time of *click* are assigned to the *open* action and to *click* vice versa. As shown in Table V, if we do not further consider the attributes of an entry, more than one entry is created per action in most cases. We calculated the values by counting the entries for every action and sorted the results. The percentiles state the entry count in different places in that sorted list. The 50th percentile is equal to the median value. 5th and 95th percentile gives an upper and lower bound. If only visible entries are considered, we get only one entry for one action for at least 95% of all runs in the dataset. When counting only entries which do not reference a previous visit (*from_visit* equals 0), we get no entry for many *click* actions.

When evaluating the *visit_type* fields of the generated entries, we found that only one entry per action was generated with a *visit_type* of 1 which means *TRANSITION_LINK* [39]. As exception, the link clicking action on twitter and the open action on Baidu generated two entries this type. In all our automated generated test data, the entries generated by the open action get a *visit_type* of 1 instead of 2 (*TRANSITION_TYPED*). We think this result from the automation framework. In our manual created verification dataset, the entries of an open action have the expected type of 2. In conclusion, the users browsing behavior is mapped best, when

TABLE V
NUMBER OF CREATED ENTRIES IN HISTORY FOR ACTION *open* AND *click* FOR DIFFERENT PERCENTILES.

Browser	Open					Click				
	5th	25th	50th	75th	95th	5th	25th	50th	75th	95th
Internet Explorer 7	0.1	1	1	1	2	0	1	1	1	2
Internet Explorer 8	1	1	1	1	2	0	1	1	1	2
Internet Explorer 9	1	1	1	1	2	0	1	1	1	2
Chrome	2	2	3	3	4	1	1	2	2	3.6
Chrome (*from_visit*==0)	1	1	1	1	2	0	0	1	1	1.6
Firefox	2	2	3	3	3.6	1	1	1	2	2.6
Firefox (*visible*==1)	1	1	1	1	1	1	1	1	1	1
Firefox (*from_visit*==0)	1	1	1	1	1	0	0	0	1	1

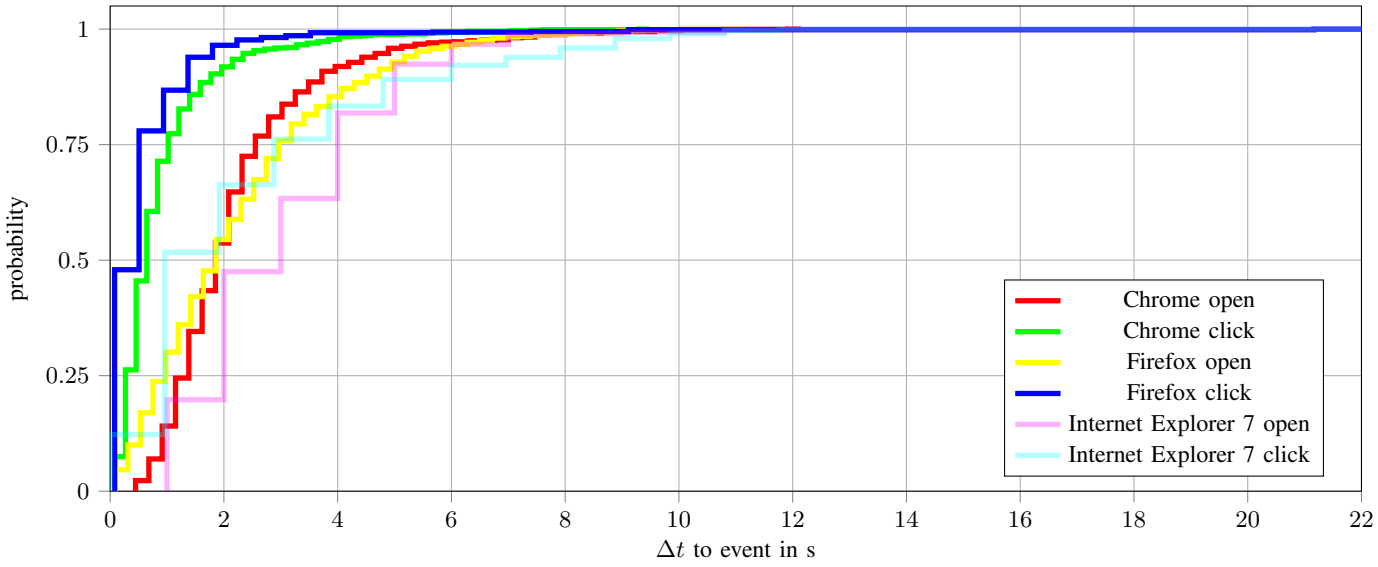

Fig. 3. Time delta from performed action to timestamp in history entry for Chrome and Firefox. (Internet Explorer 7 as reference)

only considering entries that are marked as visible and have either a *visit_type* of 1 or 2.

When we consider all history entries of the dataset regardless of the attributes *hidden* or *from_visit*, all entries' URL relate to the domain of the visited website. Only the dataset of wikia.com contains entries with the domain fandom.com because the Wikia site redirects the browser to this page.

The time difference between the timestamp of the entries compared to the logged time of the action are visualized in Figure 3. Overall, all entries related to *open* have a bigger delta than the *click* entries. It seems that the entries of the history are dated after the browser finished loading the website. For the *click* action the loading is faster because of cached content. 90% of entries created by an *open* action have a time delta smaller than 5 seconds. 90% of entries created by *click* have a time delta smaller than 1.5 seconds.

B. Chrome

For Chrome, all analyzed history entries are stored in the SQLite database `History`. This database not only contains a history of all visited sites, but also the history of downloaded items for example. The data we want to analyze is stored in the tables *urls* and *visits*. Thereby, we define a history entry of Chrome as the result of the inner join of both tables by the attributes *visits.url* and *urls.id*. We choose to investigate the semantics of the attributes *urls.url*, *visit_count*, *hidden*, *visit_time*, *from_visit* and *transition*. For further analysis we interpreted *visit_count* as the amount of entries, e.g. with a *visit_count* of 2 we counted this entry as two entries.

For every run we count how many entries get written per action, the same way as we explained for Firefox. As shown in Table V, if we do not further consider the attributes of an entry, more than one entry is created per action in most cases. When counting only entries which do not reference a previous visit (*from_visit* equals 0), we get only one entry for most *open* action, but no entry for many *click* actions.

The *transition* field encodes different values in one field which can be looked up in the chromium source code [40]. In our test dataset, every open action generated exactly one entry in the Chrome history with the attributes *PAGE_TRANSITION_TYPED* and *PAGE_TRANSITION_CHAIN_START*. For every click action

one entry with the attributes *PAGE_TRANSITION_LINK* (or *PAGE_TRANSITION_FORM_SUBMIT*) and *PAGE_TRANSITION_CHAIN_START*. The link click action on Facebook generated no entry with these attributes. Some other websites generated more than one entry with these attributes.

In conclusion, there is no ideal filter option based on attributes for entries, which will map the users browsing behavior best. When only looking for sites opened by the address bar, it is best to apply the filter *from_visit* equals 0 or filter by the *transition* attribute. To reconstruct link clicking actions the best filtering option will be the *transition* attribute, but in the case of Facebook we would not reconstruct the users click event.

When we consider all history entries of the dataset regardless of their attributes, all entries' URL are related to the domain of the visited website.

The time difference between the timestamp of the entries compared to the logged time of the action are also visualized in Figure 3. As with Firefox, all entries related to *open* have a bigger delta than the *click* entries. Again, it seems that the entries of the history are dated after the browser finished loading the website. For the *click* action the loading is faster because of cached content. 90% of entries created by an *open* action have a time delta smaller than 4 seconds. 90% of entries created by *click* have a time delta smaller than 1.8 seconds.

C. Internet Explorer

In all data from the IE runs we encountered different history files which get written to disk. But in all runs only the main histories contain entries which get listed by pasco. An entry in the MSIE history format can store up to seven attributes, but when used as main history, in our case only three attributes are used: type (always set to *URL*), url, modified time and access time. In the main history, modified and access time are always set to an identical value in all our data.

For evaluation, we split up all entries into two groups: Entries related to the *open* action and entries related to the *click* action. The decision is based on the set timestamp of the entry and our ground truth which was written during execution. If an entry has a timestamp after the *click* action it is assigned to the *click* group, to the *open* group otherwise.

For every run we counted how many entries get written per action. In most runs IE versions 7, 8 and 9 produces one entry in the history per action, as can be seen on the 25th, 50th and 75th percentiles in Table V. There exist outliers which can be seen in the different amount of entries in the 5th and 95th percentiles. In some runs (some websites) there are up to 2 entries per action and sometimes also no entry for the *click* action.

Of importance is also, how the url field of an entry relates to the opened or clicked URL. In most runs of our dataset, we observed that the history files contain only entries whose domain is highly correlated with the visited site. We encountered only three sites which additionally create entries regarding their domain, meaning that it's originating domain is not related to the visited site. For example, for IE versions 8 and 9, msn.com produced entries of akamaized.net in addition to msn.com entries. The website tumblr.com produced yahoo.com entries and wikia.com produced nocookie.net entries, additionally.

Last, we want to investigate how the timestamp of an entry correlates with the real time an action took place. We compare the timestamp of every entry in the main histories t_e with the timestamp of the action logged in our ground truth t_{gt} by calculating the delta $\Delta t = t_{gt} - t_e$. The deltas are shown as cumulative histogram in Figure 4. We see that all entries have an offset smaller than 110 seconds, but we also see, that the offsets are dependent on IE version and performed action. In general, we observed greater time deltas for IE 9, particularly for the *open* action. For these entries, only around 50% have a time delta smaller than 62 seconds.

IV. Entries in Browser Cache

In this section we evaluate, to what extent the browser's cache describes a user's browser usage. We show that a browser's cache did not map the browser usage as good as the history relating to the amount of entries, the URLs and the timestamps.

To receive the entries, we have to preprocess the cache data of Firefox and Chrome different to their histories. For all Internet Explorer versions we also use pasco to get the entries from the cache files and we use the same attributes as described in the IE history analysis.

We use the attributes *Fetch Count*, *Last Fetch*, *Last Modified* and *URL* which the Firefox cache parser outputs for every entry in the cache files. For the Chrome cache the attributes *Usage Counter*, *Reuse Counter*, *Creation Time* and *URL* get extracted with the designated parser.

In our evaluation we see, that the amount of created cache entries varies tremendously between different visited websites and also the time of visit becomes important. For example, in the evaluation of the Chrome cache data, the amount of cache entries varies between the different runs for 92% of all websites. We see the dependence of the cache from the visited website in another example: One visit of vk.com generates 3 entries and one visit of sohu.com generates 567 entries. These findings meet the expectations, that the cache of a browser is highly dependent on an websites content and applies also for the cache of the Internet Explorer and Firefox.

Which URLs are present in the cache entries after an action and if they correlate with the opened website is also of interest. For evaluation we set up four qualitative descending categories to rate how a listed URL correlates with the URL of the clicked link or the opened website:

- **Exact Match**: The URLs match exactly. An added trailing slash is allowed.
- **Indirect Match**: Added www in front of the host part is allowed. Also, a change in the protocol part is allowed (e.g. http to https) and an added trailing slash.
- **Domain Match**: The first and second level domain matches between the entered URL during the action and the URL reference in the cache.

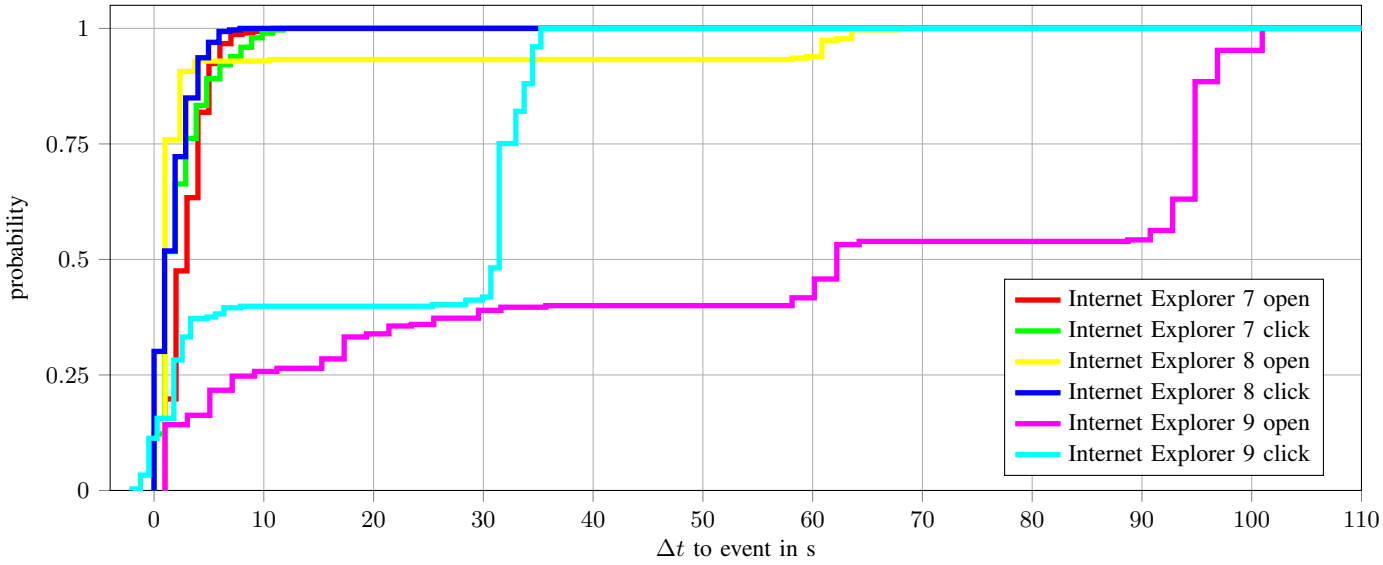

Fig. 4. Time delta from performed action to timestamp in history entry for Internet Explorer.

TABLE VI
AMOUNT OF MATCHING ITEMS IN CACHE ACCORDING TO OPENED LINK EXPRESSED BY DIFFERENT PERCENTILES.

Browser	Exact Match			Indirect Match			Domain Match			No Match		
	25th	50th	75th	25th	50th	75th	25th	50th	75th	25th	50th	75th
Internet Explorer 7	0%	0%	1%	0%	0%	0%	5%	12%	63%	35%	86%	94%
Internet Explorer 8	0%	0%	1%	0%	0%	0%	0%	10%	64%	31%	87%	97%
Internet Explorer 9	0%	1%	1%	0%	0%	0%	0%	5%	65%	30%	92%	97%
Chrome	0%	1%	3%	0%	0%	1%	9%	28%	35%	58%	69%	87%
Firefox	1%	2%	4%	0%	0%	2%	10%	31%	41%	50%	67%	87%

- **No Match**: All cache entries that match non previous category.

For every run in our dataset, we calculated the normalized amount of entries in each category. The result is shown in Table VI, where the amounts are listed for every browser and URL match category. The mean value (50th percentile) gives a good overview over the whole dataset. We also provide the 25th and 75th percentiles to show the spread of the amounts for different websites. Overall, exact and indirect match are rare and domain and no match dominate. If we look at the mean values of Chrome and Firefox, around two thirds of URLs do not correlate with the visited website. At least, one third of entry URLs correlate with their first and second level domain. Only around one tenth of entries are in the category domain match and around nine tenth do not match at all. When looking on all cache entries over the complete dataset, arround 69% of entries are in the "no match" category.

At last we evaluate how the timestamps present in cache entries correlate with the point in time when the user action was performed. As with the history evaluation we took the timestamp of a cache entry and calculated the delta to the action's timestamp.

Our method to assign entries to one of the user actions (*open* and *click*) relies on clustering entries by timestamps and assign all entries of one cluster to one action. For IE this method failed, because we get no 2 distinct clusters as expected. Therefore, we could not choose the right timestamp from the ground truth and get wrong results for the Internet Explorer versions.

The results for Chrome and Firefox are visualized in Figure 5 as a cumulative histogram. For Chrome, 90% of the *open* entries have a time delta smaller than 30 seconds and 90% of the *click* entries smaller than 10 seconds. For Firefox, 90% of the *open* entries have a time delta smaller than 42 seconds and 90% of the *click* entries smaller than 11 seconds. In general, cache entries which gets created during the initial loading of a page (action *open*) have a bigger time delta than entries produced during the second loading (action *click*). Compared to the time deltas of history entries, the time deltas of cache entries are bigger.

V. DISCUSSION

The above results can be used to scrutinize the reliability of browser evidence under normal circumstances, i.e., standard software installations, synchronized system time and no tampering [21]. Overall we experienced cases when visiting a site x at time t a history entry recorded site y (with a different domain than x) at time $t + \varepsilon$ (a different time than t). This

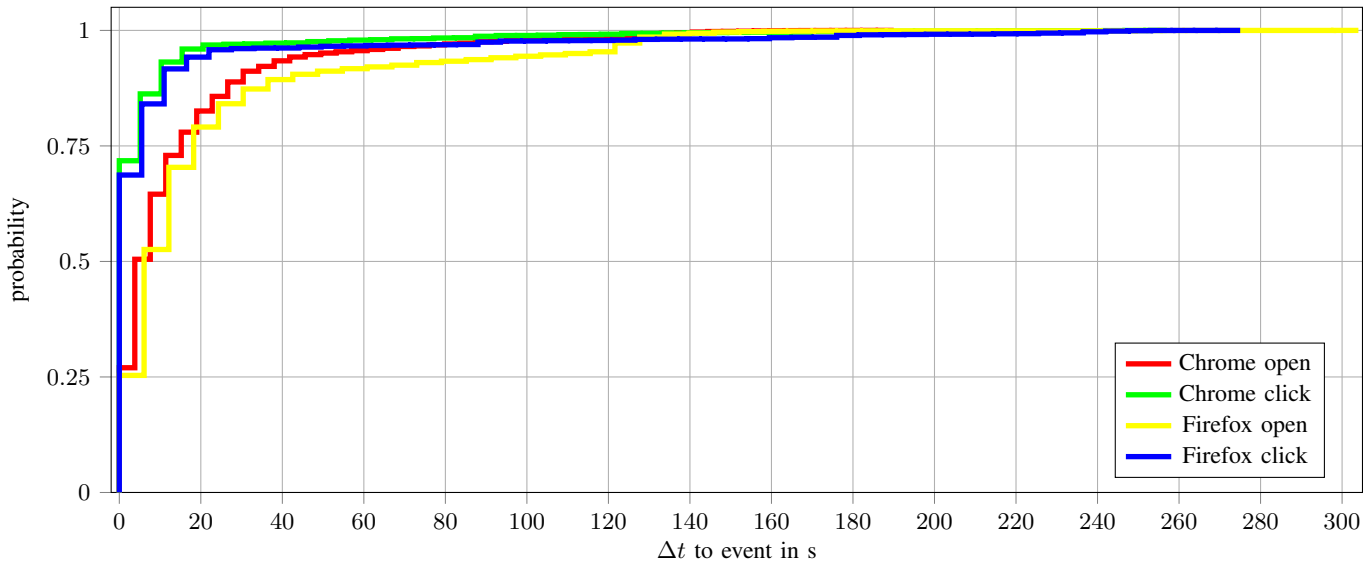

Fig. 5. Time delta from performed action to timestamp in cache entry for Chrome and Firefox.

may appear as if any claim by the investigator can be easily denied by the defendant. But this is not necessarily true. We now describe three cases in which our results can be used to support or attack testimony based on browser evidence.

A. Claim: "I did not visit site x at time t."

The first case concerns claims that, given a history entry indicating the visit of site x at time t, the user actually visited site x at time t. As our results show, this claim is almost never true, however, the probability that the user visited site x at a time shortly before t is very high. In fact, in only 3 out of 1767 cases we saw a URL value that was different to the URL that was entered or clicked on (and these three cases occurred when using the rather historic Internet Explorer). Moreover, more than 90% of the recorded timestamps were within a time window of 10 seconds from the true time at which the action was taken. So which it is not possible to attribute actions "to the second", it is hard to deny that the action was actually performed within a given time window of less than a minute.

In by far the most cases one entry per user action are created in the history data of the browsers. But for every investigated browser we also found additional entries in the history data in some cases. The Internet Explorer sometimes generated entries, referencing a favicon (`.ico` files) in addition to the entry of the called website, but we had not observed this behavior with favicons on every website. In rare cases, entries for other resources were also generated by a website visit. We could not confirm this behavior for a website visit of our test website which should provoke this behavior.

For Chrome we also experienced that an additional entry in history gets created if a redirect from http to https occur. In the user data of Firefox we also identified the creation of multiple entries which are not hidden when opening the sites Baidu and clicking the link *About* on Twitter.

B. Claim: "I definitely visited site x, oh, and I accidentally deleted my browser history."

While the timestamps and URLs present in the browser history are relatively reliable, evidence stored in the browser cache is more diverse and cannot be used to confirm the visit of a website at a particular time. Looking at our data, if we find a URL x in the cache it is overall less probable that x was directly visited than the opposite. This is a particularly tricky situation if a user has coincidentally deleted the browser history and traces cannot be found through carving.

However, we found that *if* URL x is visited then entries of x can definitely be found in the cache. Therefore, URLs that are *not* contained in the cache can be used as strong evidence that they were *not* visited before.

C. Claim: "This entry was caused by the website itself."

Within our study we did not test explicitly which data in history or cache gets generated by particular web technologies like WebSockets, AJAX, JavaScript or iframes. We believe that the most popular websites which we visited use most of the modern web technologies and we would have seen if particular technologies generate unexpected data in the history. In general, we did not experienced the generation of history entries for background communication of websites. This follows from the fact that for every user action we found in most cases only one entry in the history and in rare cases only a few (see Table V). If background communication adds entries to the history, we would have seen more entries in our dataset. Also, the distribution of time deltas to the user actions would look different if continuous background communication would have added entries in the history. The gradient of the graphs in figures 3 and 4 would be flatter in that case.

D. Claim: "I used IE 7, your conclusions do not hold."

During the execution of the Internet Explorer data generation, we experienced some problems. First of all, it was hard to create a setup where Internet Explorer 7 runs on a Windows machine, because it was already released in 2006 and was totally outdated in 2019. Also, web technologies which are used nowadays, where not available at the release of the old Internet Explorer versions 7, 8 and 9 and are not supported by it. Because of that, we were very restricted in the sites we visit with the IE browsers and we were not able to generate a more homogenous dataset. This points to a critical issue in interpreting browser evidence, in that any measurements studies like ours are always time dependent, i.e., visiting modern websites with very old browsers will come to different conclusions than visiting websites with old browsers back in the day when they were current.

VI. Conclusions

Overall, we conclude that the history of all investigated browsers in many cases corresponds to the user's browser usage but not in all. To reconstruct a user's browsing actions from the browser's cache is more complex but can help in cases where the browsing events from the history have to be checked for plausibility or in cases where the history was deleted from the user beforehand. As the problems with old versions of Internet Explorer show, our experiments can only give very vague hints on the behavior of the IE versions we tested. Studies like ours should therefore be repeated in regular intervals by researchers or forensic institutes to observe possible developments in data reliability over time.

Acknowledgments

We wish to thank Benedikt Lorch, Tilo Müller, Janine Schneider and our shepherd Michael Losavio for helpful comments on earlier versions of this paper. Work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/GRK2475/1-2019) and the German Federal Ministry of Education and Research (BMBF) as part of the FIDI project.

References

[1] K. J. Jones, "Forensic Analysis of Internet Explorer Activity Files," Tech. Rep., 2003. [Online]. Available: http://nys.fd.org/cja/forensics/ieactivity.pdf

[2] K. J. Jones and R. Belani, "Web Browser Forensics," 2005, visited on 28.06.2018. [Online]. Available: https://www.symantec.com/connect/articles/web-browser-forensics-part-1

[3] C. Boyd and P. Forster, "Time and date issues in forensic computing — a case study," *Digital Investigation*, vol. 1, no. 1, pp. 18–23, feb 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1742287604000076

[4] M. T. Pereira, "Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records," *Digital Investigation*, vol. 5, pp. 93–103, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2009.01.003

[5] S. Mahaju and T. Atkison, "Evaluation of Firefox Browser Forensics Tools," in *ACM SE '17 Proceedings of the SouthEast Conference*, 2017, pp. 5–12. [Online]. Available: http://dx.doi.org/10.1145/3077286.3077310

[6] J. Oh, S. Lee, and S. Lee, "Advanced evidence collection and analysis of web browser activity," *Digital Investigation*, vol. 8, pp. 62–70, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2011.05.008

[7] J. Oh, N. Son, S. Lee, and K. Lee, "A Study for Classification of Web Browser Log and Timeline Visualization," *Information Security Applications*, pp. 192–207, 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-35416-8_14

[8] M. Sonntag, "Automating Web History Analysis," in *IDIMT Interdisciplinary Information Management Talks*, 2012, pp. 313–324.

[9] D. Rathod, "Web Browser Forensics: Google Chrome," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 7, pp. 896–899, 2017.

[10] N. Shafqat, "Forensic Investigation of User's Web Activity on Google Chrome using Open-source Forensic Tools," *International Journal of Computer Science and Information Security*, vol. 16, no. 9, pp. 123–132, 2016. [Online]. Available: http://paper.ijcsns.org/07_book/201609/20160919.pdf

[11] G. Horsman, "Reconstructing streamed video content: A case study on YouTube and Facebook Live stream content in the Chrome web browser cache," in *Digital Forensic Research Workshop*, 2018.

[12] S. Matsumoto and K. Sakurai, "Acquisition of Evidence of Web Storage in HTML5 Web Browsers from Memory Image," in *2014 Ninth Asia Joint Conference on Information Security*. IEEE, sep 2014, pp. 148–155. [Online]. Available: http://ieeexplore.ieee.org/document/7023253/

[13] S. Matsumoto, Y. Onitsuka, J. Kawamoto, and K. Sakurai, "Reconstructing and Visualizing Evidence of Artifact from Firefox SessionStorage," in *Information security applications: 15th International workshop, WISA 2014*, 2014, pp. 83–94.

[14] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh, "An Analysis of Private Browsing Modes in Modern Browsers." in *USENIX Security'10 Proceedings of the 19th USENIX conference on Security*, 2010, pp. 1–8. [Online]. Available: http://www.collinjackson.com/research/private-browsing.pdf

[15] H. Said, N. Al Mutawa, I. Al Awadhi, and M. Guimaraes, "Forensic analysis of private browsing artifacts," in *2011 International Conference on Innovations in Information Technology*. IEEE, apr 2011, pp. 197–202. [Online]. Available: http://ieeexplore.ieee.org/document/5893816/

[16] A. Marrington, I. Baggili, T. A. Ismail, and A. A. Kaf, "Portable web browser forensics: A forensic examination of the privacy benefits of portable web browsers," in

2012 International Conference on Computer Systems and Industrial Informatics. IEEE, dec 2012, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6454516/

[17] D. J. Ohana and N. Shashidhar, "Do private and portable web browsers leave incriminating evidence?: a forensic analysis of residual artifacts from private and portable web browsing sessions," *EURASIP Journal on Information Security*, pp. 1–13, 2013.

[18] A. Ghafarian and S. A. H. Seno, "Analysis of Privacy of Private Browsing Mode through Memory Forensics," *International Journal of Computer Applications*, vol. 132, no. 16, pp. 27–34, 2015.

[19] A. Ghafarian, "Forensics Analysis of Privacy of Portable Web Browsers," in *ADFSL Conference on Digital Forensics, Security and Law*, 2016, pp. 183–194.

[20] N. Joseph, S. Sunny, S. Dija, and K. L. Thomas, "Volatile Internet Evidence Extraction from Windows Systems," in *2014 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2014.

[21] F. C. Freiling and L. Hösch, "Controlled experiments in digital evidence tampering," *Digital Investigation*, vol. 24, pp. S83–S92, 2018. [Online]. Available: https://doi.org/10.1016/j.diin.2018.01.011

[22] Desktop Browser Market Share Worldwide Jan - Dec 2018. Last visited: 2019-12-10. [Online]. Available: https://gs.statcounter.com/browser-market-share/desktop/worldwide/2018

[23] Browser Version Market Share Worldwide Jan - Dec 2010. Last visited: 2019-12-10. [Online]. Available: https://gs.statcounter.com/browser-version-market-share/all/worldwide/2010

[24] Browser Version Market Share Worldwide Jan - Dec 2012. Last visited: 2019-12-10. [Online]. Available: https://gs.statcounter.com/browser-version-market-share/all/worldwide/2012

[25] Forensicswiki: Mozilla firefox. Last visited: 2020-01-29. [Online]. Available: https://forensicswiki.xyz/wiki/index.php?title=Mozilla_Firefox

[26] J. Habben. Firefoxcache2. Last visited: 2020-01-08. [Online]. Available: https://github.com/JamesHabben/FirefoxCache2

[27] Forensicswiki: Google chrome. Last visited: 2020-01-29. [Online]. Available: https://forensicswiki.xyz/wiki/index.php?title=Google_Chrome

[28] J.-R. Bancel and L. Cimon. (2018, Mar.) Chromagnon. Last visited: 2019-08-30. [Online]. Available: https://github.com/JRBANCEL/Chromagnon

[29] (2018) Splinter. Last visited: 2020-01-07. [Online]. Available: https://splinter.readthedocs.io/en/latest/

[30] Selenium. Last visited: 2020-01-07. [Online]. Available: https://www.seleniumhq.org/

[31] Geckodriver. Last visited: 2020-01-07. [Online]. Available: https://github.com/mozilla/geckodriver

[32] ChromeDriver. Last visited: 2020-01-07. [Online]. Available: https://sites.google.com/a/chromium.org/chromedriver/

[33] "Open Virtualization Format," visited on 28.06.2018. [Online]. Available: https://www.dmtf.org/standards/ovf

[34] "Virtual Disk Format 5.0," visited on 28.06.2018. [Online]. Available: https://www.vmware.com/support/developer/vddk/vmdk_50_technote.pdf

[35] AutoIt Scripting Language. Last visited: 2020-01-07. [Online]. Available: https://www.autoitscript.com/site/autoit/

[36] S. L. Garfinkel, "Automating disk forensic processing with sleuthkit, xml and python," in *2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*. IEEE, 2009, pp. 73–84.

[37] B. Carrier. The Sleuth Kit. Last visited: 2019-10-04. [Online]. Available: https://www.sleuthkit.org/sleuthkit/

[38] Alexa - top sites. Last visited: 2020-01-10. [Online]. Available: https://www.alexa.com/topsites

[39] Mozilla firefox 3 history file format. Last visited: 2020-01-28. [Online]. Available: https://forensicswiki.xyz/wiki/index.php?title=Mozilla_Firefox_3_History_File_Format

[40] Page transition types source code. Last visited: 2020-01-28. [Online]. Available: https://chromium.googlesource.com/chromium/+/trunk/content/public/common/page_transition_types.h