Browser forensics: adblocker extensions.

Willem Rens (UvA MSc SNE student)

Supervisor: Johannes de Vries (Fox-IT)

February 12, 2017

Abstract

The purpose of this research is to identify artifacts that are left behind by adblocking extensions. First their mechanics are explored, which includes a source code study of the two most used adblocking extesions, AdBlock and Adblock Plus. Then, samples are gathered by browsing to the top 50 websites of the Netherlands, without and with an adblocker extension installed. The same is done for running the browser in private mode. Although all tested combinations leave artifacts, or even send data over the network, only Firefox together with Adblock Plus stored artifacts that can be used to determine browsing history.

Contents

1 Introduction

Ad-blocking extensions for web browsers have seen a large increase in use over the past years. The latest usage estimates vary widely, from 20% by analysis of network traffic of a major European ISP [1][2] in 2015, to 62% by undergraduate business students measured in 2011 [3]. A report by Adobe and Pagefair indicates a 41% increase of usage year by year [4]. This wide and increasing adoption makes it so that in case these extensions implicitly or explicitly store useful artifacts, browser forensics may benefit from this. Of course there are other ways to determine browsing activity but its methods are not always available, such as when browsing happened in private mode. If an ad-blocking extension runs during this mode and leaves usable artifacts, such as filter hits, they may provide forensic researcher with just enough information to complete their investigation. In short, the goal of this project is to determine what artifacts are stored by ad-blocking extensions and to determine to what extent they can be used to proof browsing history. The next section will briefly go over related work and describes two situations why more traditional methods to browser forensics do not always succeed.

2 Background information and previous work

A reason why conventional browser forensics may not succeed is when the user has deliberately cleared its browsing history, caches, cookies and other data stored by the browser. This does not always mean the data is not recoverable because when a file is deleted from a file system, most operating systems do not overwrite the blocks on the disk that the file is written on [9]. Specifically, Jeon et al. [10] has showed that deleted SQLite files can be recovered successfully, and browsers do store artifacts in this format. However, the success of recovery depends on whether the deleted data has been overwritten, which may happen anytime, so a forensic investigator can never completely rely on this method. Furthermore, the increase in use of solid state drives often halts such recovery techniques because unused blocks are overwritten [26].

Another reason is that all popular web browsers now include the feature of private browsing. In Mozilla Firefox this feature is known as 'Private Browsing', while Google Chrome calls it 'Incognito mode' and in Internet Explorer and Edge it is known as 'InPrivate'. When launching these browsers in private mode they all claim to maintain complete user privacy by not storing traces of web browsing sessions such as visited websites, search history, download history, web form history, cookies, or any temporary Internet files. Flowers et al. [5] studied the validity of this claim. According to them, IE11 used in 'Inprivate' mode still leaves artifacts on the disk that can lead to visited URL's and search terms used. For Firefox and Chrome the only on disk traces were found in the Windows file pagefile.sys which functions as virtual memory that temporarily stores RAM data for potential later use, which is far from optimal for forensics. Ad-blocker plug-ins, depending on the browsers settings, run during private

browsing sessions which gives ground for including this mode in the experiments.

3 Research questions

1. What artifacts are stored by the tested ad-blocking extensions during normal and private browsing?

   To support these research questions, the mechanics used by the tested ad-blockers are explored. It includes a short study of their source code.

2. To what extent can the artifacts be used to determine browsing history?

4 Approach

The latest versions of the browsers Internet Explorer, Microsoft Edge, Google Chrome and Mozilla Firefox were tested with their most popular ad-blocking extension. According to w3counter [8] in December 2016 these browsers have a market share of 8.7% (IE + Edge), 56.6% (Chrome) and 10.4% (Firefox). Safari also has a considerable market share with 14.1%, but this browser was not tested. The browsers will be tested both in normal browsing and private browsing mode. First control samples are gathered without the use of ad-blocker extensions. Next, samples are gathered with the ad-blocker extensions enabled. AdBlock is the most downloaded extension for Google Chrome and Microsoft Edge, but is not available on the other tested browsers [12] [11]. Adblock Plus is the most popular ad-blocker for Mozilla Firefox [13]. Adblock Plus also claims that they are the most popular ad-blocking extension for Internet Explorer. The tested browser and ad-blocking plugins combinations can be found in Table 1.

Browser	Extension
Google Chrome/55.0.2883.87	AdBlock 3.8.4
Mozilla Firefox 46.0	Adblock Plus 2.8.2
Internet Explorer 11	Adblock Plus 1.6
Microsoft Edge/14.14393	AdBlock 1.9.0.0

Table 1: Tested browsers and extensions.

For Chrome and Firefox each browser session is started with the creation of a fresh browser user data directory, after the browser session, this folder and its contents are stored for later analysis. The expectations were that it would be unlikely artifacts are stored outside this folder. Internet Explorer and Microsoft Edge do not have the concept of a data directory for each user profile. So for this reason, and to make sure no artifacts are stored somewhere on other places of the file system, the forensics tool OSForensics was used to determine system wide file changes. This tool is capable of creating snapshots of the file system so that a comparison of two stages provide insight into which files were created,

modified and deleted. The same tool was used by Flowers et al. [5] in their forensic investigation to browser artifacts in private and portable modes.

The browser sessions entailed the visitation of the top 50 sites of The Netherlands as per Alexa [6], see appendix section 6.1 for the complete list. This was automated using Python and Selenium, which allows for browser control. Automation of this process enables relative easy reproduction of this research project. All code is published on a repository here[1], including the user-data directories for Firefox and Chrome for each sample. Furthermore, code to analyze the samples and parse patterns.ini can be found here.

Research indicates 80% of software is used in its default setting, Wills et al. [7] confirms this for the use of Adblock Plus. This brings up the question whether it is sensible to include different settings of the extensions during the experiments. Even if specific settings have an effect on the storage of artifacts, the likelihood of finding these settings in a realistic environment may be unreasonably small to incorporate it in forensic procedures. That is why only default configurations were considered. And lastly, the experiments were carried out on a Windows 10 Home version 1607 (build 14393.693) 64-bit OS.

5 Ad-blocker mechanics

In 2004 an ad-blocker called AdBlock was released for Firefox, which was later renamed to AdBlock Plus. When Google Chrome came along in 2008, Adblock Plus initially did not port it to this new browser. This is where a new extensions called AdBlock filled the gap, which has no relation with the initial AdBlock for Firefox released in 2004. Over time, AdBlock and Adblock Plus added support for additional browsers[16]. AdBlock and Adblock Plus are thus two distinct ad-blocking extensions, that have different code bases and maintainers. Furthermore, because each browser has their own API for extensions, code bases differ slightly for each browser.

Regardless of subtle implementation specifics, its core functionality is the same for both extensions. At the heart is a mechanism to **block** or **hide** adverts based on **filters**, which follow a syntax that that can be found here [17]. It is also described in the next section. The syntax is the same for both tested ad-blockers. For more control regular expressions may also be used, but for performance reasons it is recommended against [18]. If one of these rules match an URL that is not whitelisted, the request of this url is blocked. This works trough a mechanism called content policies, this is code that gets called whenever the browser needs to load something. There are several of such built-in content policies, for example, when the browsers settings are changed so that some websites images should not be loaded, a content-policy is configured. Any browser extension can register such content-policies. So in essence, ad-blockers register a content-policy and then evaluate the to be requested domains against its loaded filter list and decide whether to block or not.

[1] https://github.com/berdt/AdBlocker-forensics

This tactic, however, does not always work because some web pages embed ads in such a way that without requesting them the page will never load, for example, by embedding them into the main HTML document. So a second tactic used is to hide HTML elements trough CSS modification, e.g. by specifying '**display: none !important;**' for elements that should not be visible. Extensions have the capability to add *user stylesheets* which can be set to have a higher priority than the websites stylesheets. These ads are however still transferred over the network but are just not visible. Elements to be hidden can be specified according to the earlier referenced filter rules, or, by any supported CSS selector for the particular browser.

5.1 Filter lists

Adblock Plus and AdBlock users can obtain lists of filter rules by subscribing to them, these are automatically updated. There are also lists available for other purposes than blocking advertisements, such as for privacy by blocking web trackers. The most popular filter list used for blocking advertisements is EasyList, it was originally developed for the 2004 version of AdBlock [19]. It currently has almost 60.000 filter rules. By default, in the used test set-up, AdBlock subscribed to three lists: EasyList[21], the exception rules list [22] that whitelists advertisements that are blocked by EasyList but comply with certain criteria as is described here [20]. And the AdBlocks custom filters list [23]. Adblock Plus used EasyList and exception rules list, which are exactly the same as the ones used by AdBlock.

5.2 Filter syntax

The simplest filter rule for blocking is a complete domainname, e.g. '*example.com*' will block:

- http://example.com
- https://example.com
- http://ads.example.com
- http://example.com/ad.js

One can also specify wildcard characters, e.g. '*example.com/ads/**', will block:

- http://example.com/ads/newad500
- https://example.com/ads/annoyingad/5

More options can be specified, such as to restrict from within which html tags the domain may or may not be loaded or restrict the filter to specific domain names. These options are specified after the '$' sign and are separated using a comma, e.g. ads.google.com$script,domain= myownsite.com will block:

- http://ads.google.com if it is between script tags or loaded as an image but not loaded from myownsite.com.

But it will not block:

- http://ads.google.com if it is between an img tag or requested from from myownsite.com.

Elements can be hidden by explicitly specifying them, such as by an id, class, table format or any combination of them. CSS selectors can also be used. Note that these filter do not need to have a domain component. For example, *'###advert'* will hide all elements that have the id 'advert', regardless on which domain. *'##.advert'* will hide elements that have the class 'advert'.

6 Code review

To get a better understanding of the mechanics used by AdBlock and Adblock Plus its source code was studied. For this analysis the versions for Firefox and Chrome were used. Note that each browser has its own API for extensions so there are subtle differences depending on the browser. Some code snippets that are interesting from a forensics point of view of are dissected below. By no means an exhaustive of all interesting code is shown here, they are merely a subset with the most notable findings.

6.1 Adblock Plus 2.8.2 source

For this short code review the Firefox version of Adblock Plus 2.8.2 was used. The .xpi file format used by Firefox to install extensions is merely a ZIP-archive with an installation script so unzipping is enough to get access to the source.

```
addUserCSS(subject, selectors.map(
  selector => selector + "{display: none !important;}"
).join("\n"));

if (!isPrivate(subject))
  port.emit("addHits", filters);
```

In the above snippet, when there is a filter hit for an element that needs to be hidden, the function AddUserCSS is called, which updates the user stylesheet css with the selector of the to be hidden element together with the property 'display: none !important'. Also interesting is the conditional branch depending on the value of isPrivate. This boolean is true if the request belongs to a private browsing session.

```
port.on("addHits", filters =>
{
  for (let text of filters)
    FilterStorage.increaseHitCount(Filter.fromText(text));
});
```

This is a listener for port.emit with statement 'addHits', increasing the hit count for a given filter by calling FilterStorage.increaseHitCount. This class manages filters from local storage, manages them in memory and writes them back. In several other filter hit cases this function is called, the next snippet shows this function.

```
increaseHitCount: function(filter)
{
  if (!Prefs.savestats || !(filter instanceof ActiveFilter))
    return;

  filter.hitCount++;
```

```
  filter.lastHit = Date.now();
}
```

This function has a conditional branch depending on the boolean !Prefs.savestats and another variable. If this branch is taken, the filter hitCount is increased and its lastHit variable is set by Date.now(), which returns the number of milliseconds elapsed since 1 January 1970 00:00:00 UTC [24]. The next snippet shows where Prefs.savestats is defined.

```
toggleSaveStats: function(window)
{
  if (Prefs.savestats)
  {
    if (!Utils.confirm(window, Utils.getString("clearStats_warning")))
      return;

    FilterStorage.resetHitCounts();
    Prefs.savestats = false;
  }
  else
    Prefs.savestats = true;
}
```

This snippet is responsible for managing the setting that decides whether filterhits are to be stored. When turned off (default: turned on) the function resetHitCounts() is called. This functions sets the hitCount and lastHit for each filter hit to 0.

6.2 AdBlock 3.8.4 source

For this short code review the Chrome version of Adblock 3.8.4 was used. The .crx file format used by Chrome to install extensions is also just a ZIP-archive with a special header. So again, unzipping was enough to get access to the source.

```
for (var i = 0; i < selectors.length; i += SELECTOR_GROUP_SIZE)
    {
      var selector = selectors.slice(i, i + SELECTOR_GROUP_SIZE).join(",
          ");
      style.sheet.addRule(selector, "display: none !important;");
    }
```

It can be seen in the above snippet, that for elements that are to be hidden, the css styling property display: none !important is added. This is the same mechanism Adblock Plus uses.

```
logging = function(enabled)
{
```

```
  if (enabled)
  {
    log = function()
    {
      if (VERBOSE_DEBUG || arguments[0] != '[DEBUG]') // comment out for
        // verbosity
        console.log.apply(console, arguments);
    };
    logGroup = function()
    {
      console.group.apply(console, arguments);
    };
    logGroupEnd = function()
    {
      console.groupEnd();
    };
  }
  else
  {
    log = logGroup = logGroupEnd = function()
    {
    };
  }
};
logging(false); // disabled by default
```

This is a logging function but it only logs to the console and is disabled by default.

```
var data = {
       u : user_ID,
       v : version,
       f : flavor,
       o : os,
       bv : browserVersion,
       ov : osVersion,
       ad: getSettings().show_advanced_options ? '1': '0',
       l : determineUserLanguage(),
       pc : total_pings,
       cb : getSettings().safari_content_blocking ? '1' : '0',
     };

var ajaxOptions = {
       type : 'POST',
       url : stats_url,
       data : data,
       success : handlePingResponse, // TODO: Remove when we no longer
           do a/b
                                     // tests
```

```
      error : function(e)
      {
        console.log("Ping returned error: ", e.status);
      },
    };
```

Above the definition of two variables is shown, that is data and ajaxOptions, they are part of a periodic 'ping', every 55 minutes, which send a POST request to https://ping.getadblock.com/stats/. The 'data' variable shows all the parameters that are send as payload.

```
function updateStats()
  {
    var statsPage = document.getElementById("stats-page");
    var blockedPage = getBlockedPerPage(currentPage).toLocaleString();
    i18n.setElementText(statsPage, "stats_label_page", [blockedPage]);

    var statsTotal = document.getElementById("stats-total");
    var blockedTotal = Prefs.blocked_total.toLocaleString();
    i18n.setElementText(statsTotal, "stats_label_total", [blockedTotal]);
  }
```

This function updates the stats counter that is visible when clicking on the AdBlocks extension logo. Those stats are twofold, one is the total amount of filter hits on the current page, the second one is the total amount of filter-hits since installation.

7 Results

This section presents the results of the investigation to artifacts left behind by ad-blockers. Browser extensions are limited to storing data by using the browsers provided API. As such, any code logic that explicitly stores information can not store files outside of pre-defined locations by the browser. Firefox and Chrome store all data for each profile in its own data directory. These were extracted and can be found in the github repository.

Because Internet Explorer and Microsoft edge store data on different places of the file system, and to account for any implicit left behind artifacts, OSforensics was used to monitor the whole file system during sample creation. Snapshots were created before and after, they were compared, resulting in a list of file changes. Comparing these lists of file changes from a control sample against an ad-blocking sample gave clear insight in system wide file activity that was specific for ad-blocking enabled browsing.

7.1 Google Chrome AdBlock 3.8.4 Artifacts

In the local extensions settings folder, local persistent storage for the keys as seen in Table 3 is managed. They are stored in LevelDB format, which is developed by Google. Its files are stored using a name scheme starting with 6 integers with .ldb or .log extension. The .log file seems to be related to levelDB, using it as sort of temporary storage, in this case it stored old values of blobked_total. This storage is not encrypted [25]. A way to read its contents is by browsing to the extensions background page, then having the extension 'Storage Explorer' installed, enter the development tools and open chrome.storage.local. Other attempts, such as by reading this file with several python libraries and a dedicated levelDB viewer did not work.

File(s)	**Content(s)**
Default/Local Extension Settings/gighmmpiobklfepjocnamgkkbiglidom/[0-9]{6}.log	Old blocked_total values
Default/Local Extension Settings/gighmmpiobklfepjocnamgkkbiglidom/[0-9]{6}.ldb	Settings

Table 2: AdBlocks data storage location, file names are in regular expression format.

Key	Content
blockage_stats	Epoch installation time
file:pattern.ini	Filter list + subscription
next_ping_time	Sends user data to https://ping.getadblock.com/stats/ on given epoch time
pref:blocked_total	Total amount of filter hits since installation
pref:currentVersion	Version number
pref:notificationdata	Information about the subscriptions such as when to check for updates.
pref:settings	Settings
pref:total_pings	Total amount of pings
userid	unique user ID

Table 3: Chrome together with AdBlock key value store contents

And lastly, the private browsing sample has similar results except for the .log file that is now filled with the content of the levelDB key: file:pattern.ini that contains the filters and information about filter list subscriptions. Outside of the Local Extension Settings folder no other notable artifacts were found.

7.2 Mozilla Firefox Adblock Plus 2.8.2 Artifacts

A file called patterns.ini stores the filters used, filter list subscriptions and filter hits. New filter hits get appended to the beginning of the file just after a version declaration. They contain a hitCount and lastHit variable for each filter that was ever activated. hitCount is the total amount of times this filter has been activated by the user. lastHit is the epoch timestamp of the last hit.

File	Content(s)
/adblockplus/patterns.ini	Filters used, filter list subscriptions and filter hits
/adblockplus/patterns-backup1.ini	Empty
/extensions/d10d0bf8-f5b5-c8b4-a8b2-2b9879e08c5d/*	Adblock Plus application files
prefs.js	User preferences, adblockplus settings that are different than default are added here

Table 4: Adblock Plus files, location relative to the users data directory.

To test the usability of the stored filter hits, a python script was made that parses patterns.ini to check which visits out of the 50 visited websites in the sample left traces. 16 visits including their time of visit could be proofed. To get a better measurement, it was tested on a larger sample of to browsing to 500 different websites. Out of these 143 site visits could be determined.

Adblock Plus for Firefox explicitly checks whether the browser is running in private mode, if that is the case, filter hits are *not* stored.

7.3 Internet Explorer Adblock Plus 2.0.1 Artifacts

Unlike Adblock Plus together with Firefox, Internet Explorer together with Adblock Plus 2.0.1 does not store filter hits in the patterns.ini file. See table 5 for what it does store. Furthermore, a registery key HKCU/Software/AdblockPlus was created which contains a REG_SZ called AppDataFolder that points to the folder where the filters and settings are stored.

File(s)	Content(s)
AppData/LocalLow/Adblock Plus for IE/patterns.ini	Filters used and filter list subscriptions information.
AppData/LocalLow/Adblock Plus for IE/prefs.json	"notificationdata"
AppData/LocalLow/Adblock Plus for IE/settings.ini	Settings that are different than default are added here .
C:/Program Files/Adblock Plus for IE/*	Application files

Table 5: Adblock Plus artifacts

7.4 Microsoft Edge AdBlock 1.9.0.0 Artifacts

The last tested combination stores data in several places, which is shown in Table 6. Just as with AdBlock together with Chrome, this combination also stores data in a key value store, its contents can be seen in 7.

File(s)	Content(s)
WindowsApps/BetaFish.AdBlock/*	Application files
ProgramData/Microsoft/Windows/AppRepository/BetaFish.AdBlock/	Manifesto file
ProgramData/Microsoft/Windows/AppRepository/Packages/BetaFish.AdBlock/*	package deployment files
AppData/Local/Packages/BetaFish.AdBlock_c1wakc4j0nefm/Settings/settings.dat	Key value store

Table 6: AdBlock artifacts

Key	Content
blockage_stats	Epoch time first filter hit and the total amount of filter hits since installation, split between 'total' and 'malware_total".
filter_lists	Pointing to filter lists location.
last_subscriptions_check	Epoch time last time filters were updated.
next_ping_time	Sends user data to https://ping.getadblock.com/stats/ on given epoch time.
settings	Settings.
total_pings	Total amount of pings.
userid	Unique user ID.

Table 7: Key value store

8 Conclusion

The Windows file system was systematically checked for artifacts left behind when using AdBlock or Adblock Plus, which are the most popular ad-blocking extensions depending on the tested browser type. This was done by creating control, ad-blocking and private browsing samples. These samples were then compared to determine what artifacts are stored during browsing with the tested adblockers. Three tested combinations at least stored a total filter hit-count, which is the total amount of filter hits since its installation. Its usability for browser forensic purposes seems limited. Furthermore it was found that AdBlock periodically sends data to their servers, which among others, contain information about OS type, browser type and settings used.

Most notable was the finding that Firefox in combination with Adblock Plus stores filter hits, they can be used to determine the time of the last visit of a website. In the tested sample[2] of 50 websites, 18 of them could be proven due to domain components in the filter hit. Furthermore, in a larger sample[3] of 500 websites, an estimated 30% of visits could be proven. Filter hits that have no domain component may still be useful to proof the visitation of a website but it will always be with a certain degree of uncertainty. E.g., if there is the suspicion a specific website has been visited, and there are exact matching filter hits, there is an 1/X change, depending on the amount of websites X that trigger the exact same filters, that this website has indeed been visited. Finding out X, however, is non trivial if not impossible. Lastly, the tested Adblock Plus version for Firefox explicitly checks whether the browser is running in private mode, if this is the case, the filter hits are not stored. So the developers made sure private browsing mode is respected.

Combination	**Artifacts useful for determining browsing history**
Mozilla Firefox and Adblock Plus 2.8.2	Yes
Google Chrome and AdBlock 3.8.4	No
Internet Explorer and Adblock Plus 2.0.1	No
Microsoft Edge and AdBlock 1.9.0.0	No

Table 8: Summarized results

[2] https://github.com/berdt/AdBlocker-forensics/tree/master/FireFox/AdBlock
[3] https://github.com/berdt/AdBlocker-forensics/tree/master/FireFox/AdBlock500

9 Future research

The PoC[4] created to reliably parse the patterns.ini file is only partly finished, correctly parsing the filter hits so that domains can be classified in definitely visited and maybe visited is not trivial. It is doable but due to strict time constraints no time was left to complete it. Future researchers or forensic investigators are strongly encouraged to take up this challenge, it might make a difference someday if this combination is found during a forensic investigation.

Lastly, by no means does this research claim to have tested the whole ad-blocking extension landscape. There are many more, such as uBlock Origin with millions of users, this seems much, but it is has an estimated 21 times smaller market share than AdBlock. Still, testing other ad-blocking extensions may help forensic researchers in the future when they do encounter them. A similar approach as taken in this research should be taken, however, the use of OSforensics is strongly discouraged for this task. Using the sysinternals tool Process Explorer made by Microsoft is more suitable.

[4] https://github.com/berdt/AdBlocker-forensics/blob/master/FireFox/parsePatterns.ini.py

References

[1] Pujol, Enric, Oliver Hohlfeld, and Anja Feldmann. "Annoyed Users: Ads and Ad-Block Usage in the Wild." Proceedings of the 2015 ACM Conference on Internet Measurement Conference. ACM, 2015.

[2] Metwalley, Hassan, et al. "The online tracking horde: a view from passive measurements." International Workshop on Traffic Monitoring and Analysis. Springer International Publishing, 2015.

[3] Sandvig, J. Christopher, Deepinder Bajwa, and Steven C. Ross. "USAGE AND PERCEPTIONS OF INTERNET AD BLOCKERS: AN EXPLORATORY STUDY."

[4] The cost of ad blocking PageFair and Adobe 2015 Ad Blocking Report, https://downloads.pagefair.com/wpcontent/uploads/2016/05/2015_report-the_cost_of_ad_blocking.pdf

[5] Flowers, Cassandra, Ali Mansour, and Haider M. Al-Khateeb. "Web browser artefacts in private and portable modes: a forensic investigation." International Journal of Electronic Security and Digital Forensics 8.2 (2016): 99-117.

[6] The top 500 sites on the web, https://www.alexa.com/topsites

[7] Wills, Craig E., and Doruk C. Uzunoglu. "What Ad Blockers Are (and Are Not) Doing." Hot Topics in Web Systems and Technologies (HotWeb), 2016 Fourth IEEE Workshop on. IEEE, 2016.

[8] Browser & Platform Market Share December 2016, https://www.w3counter.com/globalstats.php

[9] SIMSON, L. "Remembrance of data passed: A study of disk sanitization practices." (2003).

[10] Jeon, Sangjun, et al. "A recovery method of deleted record for SQLite database." Personal and Ubiquitous Computing 16.6 (2012): 707-715.

[11] AdBlock, https://getadblock.com/

[12] Chrome adblocker extensions, https://chrome.google.com/webstore/search/adblock

[13] Firefox adblocker extensions, https://addons.mozilla.org/nl/firefox/search/?q=adblock&sort=users

[14] Edge adblocker extensions, https://www.microsoft.com/en-us/search/result.aspx?q=adblock&form=apps&search=

[15] Adblocker modules and owners, https://adblockplus.org/en/modules

[16] AdBlock and AdBlock Plus differences, https://help.getadblock.com/support/solutions/articles/6000087894-what-s-the-difference-between-adblock-and-adblock-plus-abp-

[17] Filters, https://adblockplus.org/nl/filters

[18] Adblocker regular expression performance, https://adblockplus.org/en/filters#regexps

[19] Easylist, https://easylist.to/

[20] Acceptable ads, https://adblockplus.org/en/acceptable-ads

[21] Exceptionrules, https://easylist-downloads.adblockplus.org/exceptionrules.txt

[22] Easylist, https://easylist-downloads.adblockplus.org/easylist.txt

[23] Adblock custom list, https://cdn.adblockcdn.com/filters/adblock_custom.txt

[24] date.now() JS, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now

[25] Chrome storage API, https://developer.chrome.com/extensions/storage

[26] Nisbet, Alastair, Scott Lawrence, and Matthew Ruff. "A forensic analysis and comparison of solid state drive data retention with trim enabled file systems." (2013).

10 Appendix

10.1 Top 50 most popular sites in The Netherlands as per Alexa.com on 10-01-2017.

1. Google.nl
2. Youtube.com
3. Google.com
4. Facebook.com
5. Wikipedia.org
6. Vk.com
7. Yandex.ru
8. Yahoo.com
9. Live.com
10. Marktplaats.nl
11. Linkedin.com
12. Instagram.com

13. Reddit.com
14. Google.co.id
15. Twitter.com
16. Rutracker.org
17. Ing.nl
18. Livejasmin.com
19. Nu.nl
20. Pornhub.com
21. Aliexpress.com
22. Mail.ru
23. Booking.com
24. Ok.ru
25. Dumpert.nl
26. Telegraaf.nl
27. Google.ru
28. Imdb.com
29. Xhamster.com
30. Txxx.com
31. Bongacams.com
32. Wordpress.com
33. Imgur.com
34. Netflix.com
35. Vkmag.com
36. Tumblr.com
37. T.co
38. Bol.com
39. Rabobank.nl
40. Bing.com

41. Tweakers.net
42. Nos.nl
43. Abnamro.nl
44. Amazon.com
45. Thepiratebay.org
46. Msn.com
47. Vice.com
48. Stackoverflow.com
49. Whatsapp.com
50. Microsoft.com