The ad blocking paradox: On the privacy risks of identifying installed filter lists

Anonymous Author(s)*

ABSTRACT

Ad blockers have become a widely popular tool to protect privacy and ensure a pleasant user experience on the web. Some browsers include their own built-in ad blocker while others encourage users to install the ad blocker of their choice. The most popular ad blockers all rely on filter lists in order to block malicious network requests or hide unwanted elements from a web page. These filter lists are almost always open source and are constantly updated by a vast community. However, even though these tools are specifically designed to improve and protect users' privacy on the web, they can be exploited to complement well-known tracking techniques.

In this paper, we show that different rules in filter lists can be abused to precisely detect which lists and which versions are installed in the user's environment. This information can be used to enhance tracking, as users are likely to have a combination of different lists under different versions, but they can also reveal very specific user behaviors when a list is tailored to a specific website with very few domains in it. To understand the impact of our technique, we analyze the evolution of 460 filter lists over a period of one month to understand what makes them unique and detectable. Moreover, in a more realistic setting, users have different combinations of filter lists in which separate rules from different filter lists may end up blocking the exact same element. Thus, to assess the accuracy of our approach, we simulate realistic environments loaded with different filter lists and show that we are able to correctly identify 89% of the installed lists. Finally, we discuss the limitations as well as the impact on privacy of our attack and propose defenses.

CCS CONCEPTS

• **Security and privacy → Privacy protections**; *Social aspects of security and privacy.*

KEYWORDS

Ad blocker, filter list, tracking

ACM Reference Format:
Anonymous Author(s). 2021. The ad blocking paradox: On the privacy risks of identifying installed filter lists. In *Proceedings of The Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The web is taking an ever growing place in our everyday lives with phones, laptop, watches and even TVs being constantly connected to the internet and allowing users to access the web. As this user base keeps growing, more and more privacy and security issues arise from constantly more capable web applications. Advertisement providers make use of complex and varied techniques in order to track and extract data without the user's knowledge, building a very lucrative database of users information which can then be used for malicious purposes. Such techniques involve the use of cookies that can then be exploited through a complex set of techniques. Various tracking methods have been shaped by cookie tracking and the majority of these methods make extensive use of third party injection.

As users are becoming more aware of the privacy risks induced by their behavior on the web and look for accessible and simple ways to protect their information, ad blockers pose as a relatively straightforward defense against the unwanted spies. Browsers, such as Brave [1] and the Tor browser [2], are being conceived with the main focus on privacy and provide the users with in-the-box tools to protect them from tracking and data leaks. Major browser vendors are joining the trend with integrated protection mechanisms such as Firefox's enhanced tracking protection [3] and Safari's intelligent tracking prevention [4], proving that the public is more demanding on privacy issues. Major browser vendors also provide their users with a marketplace of extensions which contains multiple supported ad blockers such as Adblock Plus [5] and uBlock Origin [6]. Additionally, all the browsers have been providing protections against malicious popup for many years.

However, ad blockers remain the most used form of tracking protection with more than 236 millions active desktop users [6]. Adblock Plus remains one of the most used blocking extension, along with uBlock Origin and AdGuard [7]. Although the ad blocking ecosystem is vast and diverse, most adopted extensions and browser's integrated ad blockers rely on hard coded rule based filtering. These *filter lists* define domains and HTML elements that are known to provide advertising content. However, the main drawback of these lists lies in the fact that advertisers can easily evade them, by renaming their elements or moving to another domain. This issue causes the advertisers and the list maintainers to play an ever lasting game of hide and seek. Popular filter lists are therefore frequently updated, often adding new rules to counter the newly introduced advertising elements, or removing obsolete rules, which results in multiple versions of the same lists in a short time period.

[1]https://brave.com/
[2]https://www.torproject.org/
[3]https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop
[4]https://webkit.org/blog/8311/intelligent-tracking-prevention-2-0/
[5]https://adblockplus.org
[6]https://github.com/gorhill/uBlock
[7]https://adguard.com/

As these lists are supposedly updated frequently, users are likely to have different combinations of lists under different versions. This results in the user having an unique set of rules, which can help identify them.

1.1 Research questions

As users might have different ad blockers using different filter lists, privacy issues may arise from such diversity. We collected 460 lists for the duration of one month with the aim of answering the following questions:

(1) How do filter lists evolve over time ?
(2) How can filter lists be effectively exploited in order to detect which list is used along with its corresponding version ?
(3) What information can filter lists leak about the user ?
(4) What measures can be taken by ad blockers to avoid such privacy issues ?

1.2 Paper's contributions

To evaluate the extent to which filter lists may be exploited, we introduce the following contributions:

(1) **We perform a longitudinal study of filter lists which spans over the duration of one month**: we analyse the composition, frequency of update and lifespan of 460 filter lists collected every 10 minutes for a month.
(2) **We show that filter lists can be effectively detected**: we introduce a novel technique to precisely determine which filter list is used and discuss the possibility of learning user specific information like their location or browsing habits from the detected lists.
(3) **We show that it is possible to determine the version of most of the used lists**: we introduce a technique to determine which version of a specific filter list is used and discuss how these findings may contribute to a browser fingerprint.

2 BACKGROUND

Tracking on the web has long been used by advertisers to very finely target the users. Trackers rely on gathering personal information of a user such as their gender, sexual orientation or their location in order to build a profile and display very precisely targeted advertisement. Such practices have been around since the generalization of cookie usage, which came around quickly after their introduction, due to their simplicity and efficiency. In 1999, the New York Times was already describing cookies a *"comprehensive privacy invaders"* which shows that the general public was already aware of such issues more than twenty years ago [8]. Old methods have evolved into a complex and vast set of techniques that allows the users to be tracked in a very efficient way, all while trying to slip under the radar.

2.1 Tracking and advertising

As third-party elements on a website have steadily increased over the years, the usage of tracking methods has exploded. Third parties resort to complex techniques such as cookies syncing [1, 30] and cookies forwarding [5]. An alternative called browser fingerprinting has been introduced more recently, and exploits the vast diversity of the browser ecosystem to forge an unique signature using the user's device and configuration. This method was first discovered by Eckersley [9] and was followed by several studies looking at its adoption online [1, 2, 13, 27] to its tracking effectiveness [17, 24].

Most of these techniques, while being different, present a common approach and abuse the browser on the client side to track users. Current networking protocols make it relatively easy for third party providers to store and use cookies, which makes it the most common form of online tracking to day. When crawling the Alexa Top 1 Million websites, Englehardt et al. [13] found that more than 81,000 third parties were present on at least 2 pages with Google Analytics present in more than 60% of the crawled pages. Additionally, a study led by Ghostery and Cliqz showed that 77 percent of 144 million page loads contained at least one tracker [16]. These numbers show the pervasiveness of trackers on the web and provides a glimpse of the massive ecosystem that collects all this data.

As the public is getting constantly more aware and wary of their privacy, many users resort to multiple methods to make their browsing experience less obtrusive. Consequently, multiple approaches have emerged to allow users to either reduce their tracking footprint or anonymize themselves online. VPNs and privacy preserving proxies (such as Privoxy[9]) provide an efficient way to hide the true identity of the user, even though recent work have shown that it is still possible to workaround the anonymity of these tools and extract valuable information [15]. Moreover, these tools require to put your trust in an additional third party and can demand some technical knowledge to set them up correctly. On the other end of the spectrum, ad blocking can be found. By simply installing a browser extension or using a built-in protection, users can hide ads and block trackers when they go online without any specific interaction. Because of their efficiency and ease of use, ad blockers have seen a massive adoption increase in the past decade from 26 million users in 2010 to 236 million on desktop and 527 million on mobile in 2020 [6].

2.2 Ad blockers and filter lists

The first version of Adblock has been introduced on Firefox in 2002 and was aiming to remove image ads by identifying specific URLs [28]. Since then, the ad blocking ecosystem has flourished but the main mechanism behind all blockers has always stayed the same: identify resources known for tracking and block them. As described in Section 7, new ways of detecting these resources are being researched on but we focus here on the filter lists which rely on user-defined rules.

Anatomy of a filter list. Listing 1 shows an extract from the popular Easylist filter list. Lines 1 to 13 represents some metadata with comments and lines 15 to 24 some examples of rules that can be found in it. Each new line in a filter list represents a new rule and while some lists may have less than 10 rules, the biggest ones like EasyList have more than 70K rules.

[8]https://www.nytimes.com/1999/04/06/opinion/privacy-isn-t-dead-yet.html

[9]https://privoxy.org

Listing 1: Extract from the Easylist filter list

```
1  [Adblock Plus 2.0]
2  ! Version: 202010160649
3  ! Title: EasyList
4  ! Last modified: 16 Oct 2020 06:49 UTC
5  ! Expires: 4 days (update frequency)
6  ! Homepage: https://easylist.to/
7  ! Licence: https://easylist.to/pages/licence.html
8  !
9  ! Please report any unblocked adverts or problems
10 ! in the forums (https://forums.lanik.us/)
11 ! or via e-mail (easylist@protonmail.com).
12 ! GitHub issues: https://github.com/easylist/easylist/
       issues
13 ! GitHub pull requests: https://github.com/easylist/
       easylist/pulls
14
15 ! Network rules
16 ||doubleclick.net^
17
18 ! Element rules
19 ###doubleClickAds_top_banner
20 ##.doubleClickAd
21 ##a[href^="https://ad.doubleclick.net/"]
22
23 ! Exception rules
24 @@||g.doubleclick.net/ads/preferences/$popup
```

Type of rules. Ad blockers work by examining networks requests and the DOM composition of a webpage to try and match the resulting against the set of rules defined by the filter lists. Filter lists can be separated into three main categories as described in the AdBlock Plus filter's specification [10]:

- **Network rules (also called Blocking filters)**, which allows the ad blocker to intercept and blocks networks requests from third party domains defined in the list.
- **Element rules (also called Content filters)**, which hides HTML content on a web page. The targeted page can either be a specific domain, or the rule can apply regardless of the domain.
- **Exception rules**, which allow specific URLs defined as a network rule to be authorized by the ad blocker.

Examples. Rules in filter lists all follow the same syntax [31] and Listing 1 present several examples of them. For the only network rule on Line 16, all network requests to *doubleclick.net* and its subdomains denoted by "||" are blocked. For the three element rules present on lines 19 to 21 denoted by the **"##"** anchor, different types of elements are hidden. The first with a **"#"** ID identifier hides all elements with the ID *doubleClickAds_top_banner*. The second with a **"."** class identifier hides those with the *doubleClickAd* class while the last one hides all **"a"** link elements that directs to the *ad.doubleclick.net* website. Finally, line 24 with the **"@@"** anchor illustrates why ad blockers need to support exception. Visiting *g.doubleclick.net/ads/preferences/* lets a user set her ad preferences for *doubleclick* ads (even if the user has an ad blocker, she may wish to whitelist a site to show ads and support them financially). However, because of the rule on line 16, all requests to the *doubleclick.net* are blocked and she will not be able to set her preferences unless it is added as an exception. The uBlock Origin Unbreak list[11] is full of examples of such exceptions to counteract the effects of very generic rules that block too many elements.

[10] https://help.eyeo.com/adblockplus/how-to-write-filters#creating-filters

Sources and updates. Each filter list has its own way of being maintained: some are updated by either a small number of privacy activists while others are crowdsourced by a large number of users. Some companies like Ghostery [12] or Disconnect [13] even perform their very own crawls to maintain their own lists. Looking at Listing 1, we can see that users can report problems or unblocked ads on a forum, by email or on GitHub where the list is hosted. Regarding updates, each list has its own frequency of releases and here, the *Version* and *Expires* lines provide information on when the list was downloaded and when it will expire.

3 ANALYSIS OF FILTER LISTS

In this section, we look at the composition and evolution of a large dataset of filter lists to understand how they can be abused to enhance user tracking. More specifically, we analyze their uniqueness (Section 3.2), their update frequency (Section 3.3), and their expected lifespan (Section 3.4).

3.1 Dataset Description

In order to provide a global overview of the current state of filter lists, we used the website *filterlists.com* that references 2,100 different filter lists along with their description, their download links and their syntax. To limit the efforts of supporting a wide range of different syntaxes and creating parsers for each of them, we limited ourselves to two specific ones: 1) the *Adblock Plus filter* syntax which is the standard one adopted by all the major ad blockers; 2) the *uBlock Origin static filter* syntax supported by uBlock Origin [14]. We ended up with a selection 526 lists that we reduced to 460 after we removed the ones that were a simple aggregation of other lists or supported multiple syntaxes, along with those whose links were broken.

As we will see in Section 4, we are also interested in studying the evolution of filter lists over time. We collected the contents of these 460 filters lists every 10 minutes for a duration of 1 month starting from `18/09/2020` until `18/10/2020`. This led us to collect in total `9,713` versions of filter lists during the entire period of our data collection.

3.2 Uniqueness of rules

Rule Type	# Total Rules	% of unique rules
Element	744,504	20.6%
Network	2,139,823	66.7%
Exception	87,328	23.45%

Table 1: Number of rules by rule type

[11] https://raw.githubusercontent.com/uBlockOrigin/uAssets/master/filters/unbreak.txt

[12] https://github.com/ghostery/ghostery-extension/blob/master/databases/README.md

[13] https://disconnect.me/trackerprotection

[14] https://github.com/gorhill/uBlock/wiki/Static-filter-syntax

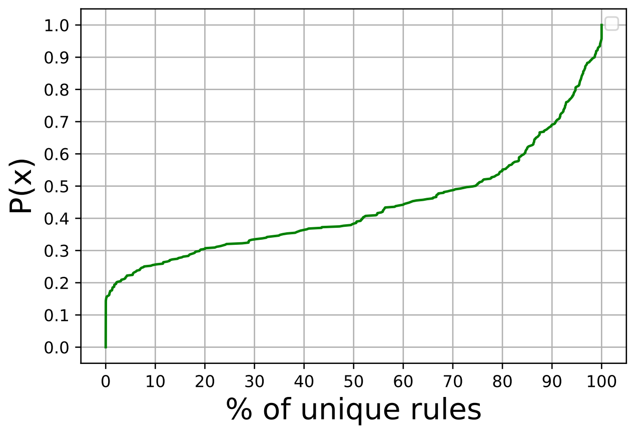

Figure 1: CDF showing % of unique rules on X-axis and fraction of lists on Y-axis

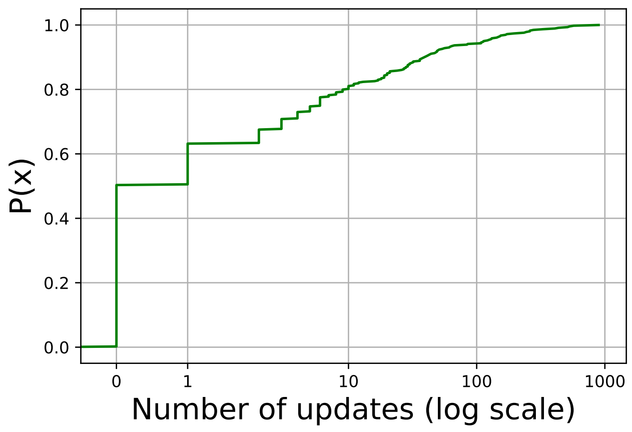

Figure 2: CDF showing number of updates to filter lists during the collection phase of 1 month.

The incredible diversity of filter lists, ranging from generic lists such as Easylist or Easyprivacy, to more specific lists such as the *YouTube Annoyances* or the *Twitch: pure viewing experience* lists, leads to an interesting study of the composition of filter lists. From our dataset of 9,713 versions of lists collected over a month, we pick the latest version of each of the 460 lists and compute the number of rules in each of them. We observed on average 6,462 rules.

As described in Section 2.2, filter lists contain 3 types of rules, namely, network rules, element rules and exceptions. Table 1 shows the total number of rules of each type in our dataset of 460 lists. We also compare each filter list with every other list in our dataset to get the rules which are unique to a given list. Hence, the table also presents the percentage of rules that are unique for a given rule type. We can see from the table that network rules have a significantly higher percentage of rules that are unique as compared to element rules. This can be explained by the fact that element rules are not domain specific and hide all ads with a similar structure so they have a higher chance of being found in multiple lists than the rules targeting a single domain.

Figure 1 shows a CDF with % of unique rules on X-axis and fraction of lists on Y-axis. We can see from the graph that ~60% of filter lists in our dataset have at least of 50% rules that are unique to a given list. This means that majority of filter lists are at risk of being detected as explained in Section 4.

3.3 Update Frequency

These filter lists are mainly maintained by individual contributors and given the ever lasting cat-and-mouse game between the advertisers and ad blockers, their effectiveness depends on how often they are updated. Thus, we study the update frequency of 460 filter lists in our dataset and present our findings in this subsection. During our collection phase, we were able to collect 9,713 versions across 460 filter lists, thus showing 21.11 updates on average for each list.

Figure 2 shows a CDF of the update frequency of the filter lists in our dataset with the number of updates on the x-axis and the fraction of lists on the y-axis. It is evident that nearly half of the lists in our dataset never received any update, while the other half displays a skewed update count. Furthermore, about 80% of all filter lists received less than 10 updates over the entire duration of our data collection, while only around 6% of lists had more than 100 updates.

The fact that the majority of the lists are relatively static can be explained by multiple factors:

- The list may no longer be maintained, hence receives no updates. As many of these filter lists are maintained by a small community, and sometimes by a single person, their development may be slowed or halted for various reasons. It is the case for both the *Andromeda uBlock list* and the *whtsky's filter list* which received no update for the duration of our study and have only one known contributor on their associated GitHub repositories.
- The list may have reached maturity. This mainly applies to the lists that are very specific and are targeted to a small set of websites. As advertisers constantly adapt to new entries in filter lists, it is unlikely that major lists reach maturity, but smaller list maintainers may consider that their list no longer needs any update if their rules are sufficiently dynamic to counter any new advertisement on the set of websites that it covers.
- The update frequency of the list may be more than a month *i.e.* longer than the duration of our data collection. It is possible that some filter lists receive updates at a very slow rate and none of their updates fall during our data collection phase of 1 month. For example, the *0131 Block-list* has not updated during our data collection. Furthermore, the associated Github repository shows an update of frequency of two months approximately.

Types of Changes. When a filter list gets updated, the changes made to it can either be an *addition* of a new rule or a *deletion* of an existing rule. In our dataset of 9,713 versions of 460 filter lists, we observed 1,052,717 additions while there were only 852,346 deletions over a period of 1 month. This slight difference between the

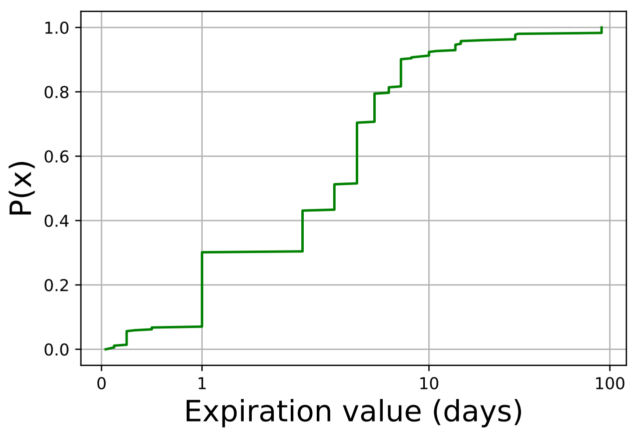

Figure 3: CDF showing the lifespan of filter lists in days

number of additions and deletions reflects that filter list maintainers are constantly removing rules that are not useful anymore, as well as constantly adding rules to keep up with new trackers and advertisements. This comes in direct opposition to observations made by Snyder *et al.* where they found that filter lists are rarely purged of their outdated rules [33].

3.4 Lifespan

As described in Section 2.2, AdBlock Plus syntax allows filter lists to set a duration for which they should be considered fresh by the ad blocker. This duration is called the lifespan or the expiry duration of the list. Ad blockers use the expiration value present in the list headers to decide whether to perform an update or not. This value is respected in AdBlock Plus but other ad blockers, like uBlock Origin, chose to set the maximum expiration value to 5 days, rather than 14 days, with the aim of reducing the number of outdated list on the client's side. Since the lifespan of a list is directly responsible for how frequently an ad blocker checks for updates, this makes for an interesting analysis on the expiration value of filter lists.

In our dataset of 460 filter lists, 356 of them had a specifically set life span while the rest did not set any expiry duration. Figure 3 depicts a CDF of expiry duration of these 356 filter lists. We can observe that the majority (~70%) of these 356 filter lists have a lifespan of more than 1 day. Furthermore, 14 of them have a lifespan of at-least 30 days, even though AdBlock Plus specification limits the expiration value to 14 days. High expiration values are commonly found in lists that have a lower update frequency to reduce their server load. The median lifespan among all the lists that display an expiration value is of 3 days. Figure 3 shows that ~30% of the lists have a life span of less than one day, with most of these becoming obsolete after 6 hours. However, even though these lists had a short life span, none of them were updated as frequently as their expiration time specified. The *Japanese AdBlock Plus* list for instance, was updated only once while specifying an expiration value of one hour. We however note that lists that had a higher expiration rate, such as EasyList, updated more frequently than its expiration value would have led us to expect.

4 EXPLOITING THE FILTER LISTS

This section presents the pipeline we built to generate test pages that can detect filter lists and their versions.

4.1 Threat model

The attack presented in this paper employs a threat model wherein the attacker performs multiple network requests or injects Document Object Model (DOM) elements that are known to be blocked by specific lists only. The malicious script can then infer which lists are used along with the ad blocking configuration. This allows for extraction of certain specific information about the user. This may include her location based on the detected linguistic filters, her browsing habits based on the usage of website-specific filter lists, or her general profile, such as the user's focus on her privacy by analyzing the detected filter lists. Furthermore, the attacker can extract the specific version of the filter list used by the user by studying the history of additions and deletions for that list.

This extracted information can then be combined with various browser fingerprinting techniques to identify an user. As *Gomez-Boix et al.* [17] showed, a high percentage of non-unique fingerprints are considered "fragile" and can be made unique by only adding or changing a limited number of characteristics. An attacker can thus make a non-unique fingerprint unique by appending the filter list composition to the user's browser fingerprint. The uniqueness can further be increased if the user has filter lists that are not used by default by the ad blockers. This unique fingerprint can then be used to track more users which were not previously tracked due to their non unique fingerprint and allows for greater efficiency in serving targeted ads.

4.2 Attack Overview

Summary. Websites can learn the browsing patterns, or user specific information by determining the filter lists used by the user. These information can then be used to track and served targeted ads to the user.

How does it work? Figure 4 provides an overview of the complete process. An attacker keeps track of different filter lists and their updates on the server-side. This results in the generation of a large database spanning over multiple days, and comprising all the versions of the different filter lists. For each retrieved filter list, the attacker loops through each of their current version and compare its entries to the rules of the other filter lists in the database. This operation retains only the non-overlapping entries of the list and leaves us at the end with an unique set of rules for each version of every filter list. The resulting set of rules are then parsed and corresponding HTML elements are crafted that can be included in the website visited by the target users. The element can then be tested to determine if it has been blocked, by using *jQuery* to evaluate its visibility. In essence, our attack can be broken down into 3 steps:

(1) Identifying detectable rules
(2) Generating HTML elements from detectable filter rules
(3) Injecting the generated HTML elements on a target page and testing the visibility of the injected HTML elements corresponding to detectable rules.

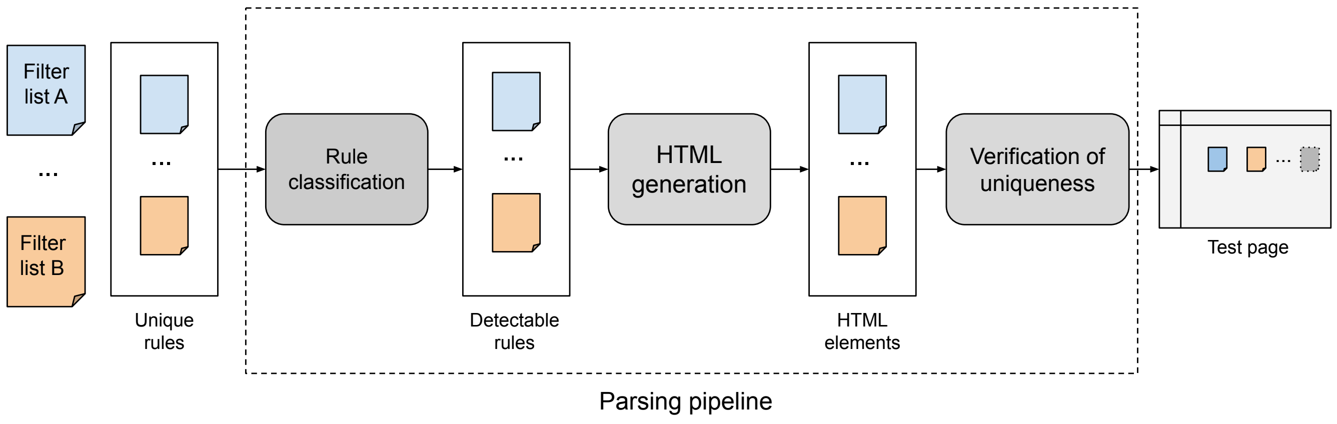

Figure 4: Attack Overview: From filter list parsing to the testing of the generated elements

4.3 Identifying Detectable rules

As described in Section 2.2, filter lists are composed of several rules in the form of regular expressions. We rely on the *re* [15] Python package to parse the different rules of a filter list and extract their values so that we can proceed to a first batch of filtering. In filter lists, some specific rules are designed to block or hide elements on a specific domain. For example, *xyz.com##.ads* indicates that any HTML element hosted on *xyz.com* and whose class is *ads* must be hidden. This means that an attacker who runs her testing page on her own domain will not be able to leverage this specific rule to identify a list because it will not correspond with the domain present in the rule. Additionally, a rule can be complemented with diverse options. The *$domain* option explicitly ties a rule to a specific domain akin to the example we just detailed while another option like *$popup* restricts the blocking to popup windows. Since some options can be complicated to parse and can simply generate no corresponding HTML elements, we limited ourselves to the *image*, *third-party* and *domain* options. By identifying these special cases early, we can filter them out so that we focus on rules that can lead to detection later on. This filtering step is characterized by the *rule classification* block in Figure 4.

4.4 Generating HTML elements from rules

Once we parse the rules and identify the detectable ones, we generate the corresponding HTML elements for network and element rules and ignore exceptions. We describe this process, depicted as the *HTML generation* in Figure 4, for both these rule types:

Element rules. Based on the target of a rule we perform the generation in three parts. Each part involves breaking down and extraction of necessary components needed to generate the HTML element it is designed to hide.

- Element rules may target a specific HTML tag and corresponding attributes. For instance, *##div[width=100][height=100]* will result in the following generated HTML element: *<div width=100 height=100></div>*.
- Some other element rules may only target a specific ID or class. Such rules result in a div element whose ID or class corresponds to the value mentioned in the rule.
- Element rules may also be a combination of both previous cases, resulting in nested HTML elements. For example, *##.ads > a[href≙'google.com']* will result in a *div* element of class *ads* containing *<a href='google.com' />*.

Network rules. These rules specify the network requests which should be intercepted and blocked by the ad blocker. These rules may specify blocking requests to a cross-origin resource when the user visits our test page, hence we only create <img> elements because they are allowed to be included in a page in a cross-origin context under the Cross Origin Resource Sharing (CORS) policy[16]. Consequently, for each network rule, we generate an associated *<img>* tag whose *src* attribute's value points to the one specified by the rule. Network rules may also be defined by URL path, and thus, for these kind of network rules, we generate a random domain and prepend it to the rule's value if no domain option is specified.
For instance, the rule */specific/ads* will end up generating the HTML element: *<img src='https://example.com/specific/ads' />*.

Verifying uniqueness. It is important to note that not all detectable rules can be ultimately used to detect filter lists as some rules might be present in multiple lists and it would be impossible to determine from which list the rule was triggered. Thus, we compare all the generated HTML elements and keep only the ones that are unique to a given list. The reason we compare the generated HTML elements and not the rules themselves is that some different rules can lead to the exact same generated HTML element. For example, the two following rules *xyz.com$third-party* and *xyz.com* are different but they correspond to the same *<img src='xyz.com'/>* element in our test page. In that case when the HTML is identical, we discard both rules from our database.

[15]https://docs.python.org/3/library/re.html

[16]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

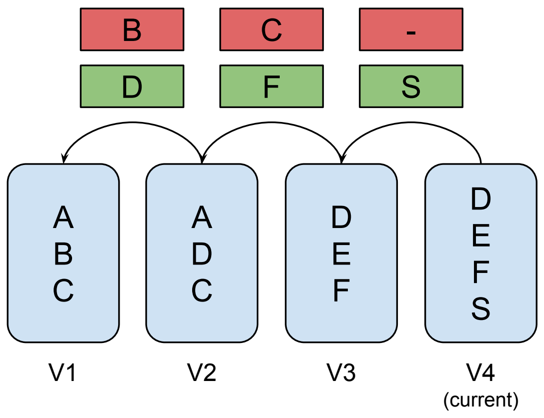

Figure 5: Evolution of rules in successive versions of a filter list

4.5 Detecting filter lists and their versions

Detecting lists Once HTML elements corresponding to every unique and detectable rules are crafted, these elements are then injected in a web-page that the targeted user is expected to visit. When the user visits it, our injected HTML elements are rendered by the client and we use *jQuery's :visible* selector to detect if some elements are visible or not to infer the presence of specific filter lists. This last step is represented by the *test page* block in Figure 4. For instance, if a filter list has a unique rule which blocks every HTML element whose *id* contains the string *ads*, we inject an element with this *id* on the web-page and check its visibility. If the element is not visible, we infer that the user is not using that filter list.

Detecting versions The technique described in the previous subsection allows the attacker to determine the used filter lists, but not the version. Once a list is detected with the above technique, we utilize the differences between versions of this given list in order to recognize rules that are unique to specific versions. We achieve this by comparing successive versions of the list to identify *additions* and *deletions* made in each version.

Example Figure 5 shows a simplified representation of how different versions of filter lists can be detected. To detect the versions from this example we test which of the rules from *A,B,C,D,E,F* and *S* are triggered by crafting HTML elements corresponding to them and checking for their visibility. Based on the visibility of these elements which reflects the rules that were triggered, we make the following inferences:

- V4: If rule S is triggered.
- V3: If rule S is not triggered and rule F is triggered
- V2: If rule A is triggered and rule B is not triggered.
- V1: If rule A is triggered and rule B is triggered.

This technique works particularly well for the lists with an average life span, *i.e* lists that are updated regularly while leaving some time between the updates, as these lists are more likely to have detectable rules that can be tested in the different versions.

5 EVALUATION OF THE ATTACK

This section presents an empirical evaluation of our attack methodology on 10 popular filter lists. First, we present the description of each of these filter lists including the number of rules in them and the criteria of their selection. Then, we craft two experiments which simulates a realistic scenario where these lists are used in combination by users. Finally, we present the results of these experiments.

5.1 Dataset Description

In order to perform the evaluation of our attack, we selected 10 lists from our dataset of 460 filter lists. *Table 2.* shows these lists along with the number of rules, and the number of times they were updated. These lists were selected based on the following criteria:

- **Adoption of lists by popular ad blockers**: we focus on the most adopted and available in default settings in the popular ad blockers as they are more likely to be used and represent a more common use-case. This choice is also guided by the fact that other less popular lists are often derived from these popular filter lists.
- **Number of rules in the list**: we try to include lists with a significant number of rules, however we do not ignore popular lists with fewer rules.
- **Update Frequency**: we include frequently updated lists as they tend to be more useful and represent a higher user base.

EasyList is one of the lists we selected for our evaluation as it is included in the vast majority of the ad blockers by default, including browser's integrated ad blockers. AdBlock Plus also enables linguistic lists by default, depending on the browser configuration, which led to the decision of including at least one linguistic list in the study. We note that ad blockers are not restricted to the default lists and the majority of them allow users to introduce their own lists, or even build their own rules. This particularity in the ad blockers motivated our decision to include lists that may not be known by a large public, such as the Fanboy's annoyances filter list. We also retrieve more independent and more limited lists, along with lists constructed for a very specific target, such as the the *Peter Lowe's list* which is mainly targeted towards domains performing user tracking.

Detectable rules Identifiability of a list depends on the presence of unique and detectable rules. If a list does not have any unique rules, it would be impossible for us to detect it based on our attack methodology. Thus, we follow the procedure described in Section 4.3 and identify detectable rules in each of these 10 lists. Table 2 shows the number of unique rules along with the amount of detectable rules. One surprising result that we observed is that despite most lists having more than 90% of unique, some of them have very few detectable ones. This is due to the way our pipeline operates. When we classify rules, we discard the ones that we cannot generate HTML elements for. For example, uBlock Annoyances reports more than 99.3% of unique rules but only 1.2% can be properly detected. After careful examination, the majority of them present unsupported options with 60% of them using the *js* option [17]. In

[17] https://github.com/gorhill/uBlock/wiki/Static-filter-syntax#scriptlet-injection

List	# total rules	# unique rules	# detectable rules	# versions	# Network/Element/Exceptions
EasyList	60,493	59,946 [99.0%]	48,605 [80.3%]	543	30,801/26,976/2,655
EasyPrivacy	17,855	17,275 [96.8%]	15,698 [87.9%]	190	17,054/0/778
EasyList French	19,330	18,150 [93.9%]	13,552 [70.1%]	29	13,856/5,080/343
uBlock	13,697	13,498 [98.5%]	464 [3.4%]	132	2,252/9,053/2,389
uBlock Annoyances	2,100	2,082 [99.3%]	25 [1.2%]	25	85/1,892/123
uBlock Privacy	79	75 [94.9%]	21 [26.6%]	2	66/10/3
AdGuard Social Media	7,910	6,967 [88.0%]	650 [8.2%]	51	431/7,221/247
AdGuard Annoyances	16,584	14,116 [85.1%]	1,273 [7.7%]	240	1707/13,597/1,278
FanBoy's Annoyances	46,159	43,606 [94.4%]	31,965 [69.2%]	140	6,129/38,149/1,881
Peter Lowe's List	3,546	3,404 [96.0%]	3,364 [94.9%]	23	3,546/0/0

Table 2: Lists selected for our experiments

the end, depending on the nature of a list and the way it is built, the number of rules detectable by our pipeline can drastically drop.

5.2 Experiment 1: Filter list Detection

To evaluate the efficiency of our attack, we create a test page and inject HTML elements corresponding to unique and detectable rules for each of the 10 filter lists. We use *Puppeteer* instrumented with *uBlock Origin*[18] to simulate a user who visits this test page while using an ad blocker. With the aim to reproduce the closest real world scenario, we generated every possible combination of the lists presented in Table 2. The motivation of introducing different combination of lists arises due to the fact that some rules deemed to be unique in a given list can in reality have a collision with another rule in a different list since one rule can be written in multiple ways. It is easy to understand this issue using an example, assume a list contains a rule that blocks any HTML element whose ID contains the word "ad", while another list contains a rule that blocks any HTML elements whose ID is exactly "specific_ad". Even though these rules are different, both these rules will be triggered in the case of HTML element generated by the second rule. These issues can be avoided by performing a more complex rule parsing in order to identify such collisions but such an optimization is out of scope of our paper as they are computationally expensive and very time consuming. Thus, we choose to detect filter lists in presence of other lists in order to build a more realistic scenario in which the user may have different filter lists, that may present some conflicts. Furthermore, we only select combinations made of at least two lists as users are unlikely to have only one activated filter list at a time. This led to us performing 1013 tests. Moreover, to avoid any caching issues we visit the test page using a new instance of browser for each combination of filter list.

F1-score. Since the detection of filter list is a classification problem we use the *F1-score* as a metric to measure the efficiency of our attack. The F1-score is represented by a weighted average of both precision and recall. It is given by the following general formula by setting β to 1:

$$F_\beta = \frac{(1+\beta^2)\cdot \text{true positive}}{(1+\beta^2)\cdot \text{true positive} + \beta^2 \cdot \text{false negative} + \text{false positive}}$$

[18]v1.30.0

List	F1-score
EasyList	82.75%
EasyPrivacy	88.67%
EasyList French	90.60%
uBlock	93.21%
uBlock Annoyances	89.27%
uBlock Privacy	88.88%
AdGuard Social Media	90.50%
AdGuard Annoyances	87.46%
FanBoy's Annoyances	96.99%
Peter Lowe's List	84.42%

Table 3: List detection's F1-score

The F1-score is particularly suitable for our problem as it takes into account both false positives and false negatives and thus gives a good overview of the performance of our detection.

For each of the 10 selected lists, we compare whether our attack managed to effectively detect the list. In the opposite case, we note whether the algorithm triggered a *false positive* or a *false negative*. These measurements resulted in a **89% F1-score** for the complete set of lists. We also measure the F1-score for each list individually.

Results. Table 3 presents the results of our experiment. The minimum detected score is 82.75% for EasyList while we achieved a maximum score of 96.99% for Fanboy's Annoyances list. We notice that lists such as FanBoy's and uBlock achieve significantly higher score of detection as compared to EasyList and Peter Lowe's lists. Even though the number of detectectable unique rules is significantly higher for these two lists, we believe the reason behind such a result is the same as described earlier in the section, with multiple rules colliding. This results in a consequently smaller number of truly unique rules than stated in Table 2. Thus, the lower score for Peter Lowe's and EasyList may be explained by the presence of many colliding rules that could not be detected by our method.

5.3 Experiment 2: Version Detection

We follow the same procedure for both the crawler and the combination of list as previously described for list detection. However, one notable difference with the previous experiment is that instead of using the latest version of each list in the ad blocker, we instead load

List	Accuracy
uBlock	100%
EasyList French	87.30%
FanBoy's Annoyances	84.13%
AdGuard Annoyances	68.25%
EasyList	44.44%
EasyPrivacy	39.68%
Peter Lowe's List	17.46%

Table 4: Version detection's accuracy

the ad blocker with a version chosen at random. We only include those versions in our combinations which have detectable rules as described in Section 4.3. As such, versions that present changes which can not be used to generate testable HTML elements are discarded from our experiment.

Unlike the previous experiment where we evaluated our method using the F1-score, we only measure the accuracy for this experiment. Lists for which the right version is detected at each occurrence will have an accuracy of 100%, while lists for which the version is never correctly detected will have an accuracy of 0%. Thus, our accuracy is measured by the following simple formula:

$$Accuracy = \frac{Correct\ results}{Correct\ results + Incorrect\ results}$$

Results. The results are presented in Table 4 and are very disparate between the filter lists. It should be noted that three lists are not present since they only had a single detectable version. We manage to achieve a very high accuracy for lists such as uBlock, Fanboy's and EasyList French, while our technique performs badly for other lists. Lists such as EasyPrivacy and Peter Lowe's have the lowest accuracy among the filter lists we tested, with Peter Lowe's list going as low as 17% of detection rate. The lowest scores are justified by the fact that the elements of the tested combinations are not unique to one list and thus are triggered by the wrong list, leading to the detection of the wrong version.

6 DISCUSSION

This section presents the privacy risks of detecting ad blockers and how it applies to real world applications along with the limitations of our method.

6.1 Privacy risks

For most users, detecting the exact list of installed filter lists presents a low privacy risk but it can become much greater in very specific instances.

Tracking users. Knowing the list of installed filters in the browser plays the same role as any other attribute collected in a browser fingerprint. It can bring a very limited quantity of information or it can be the one defining feature that makes the user unique and trackable on the web. For example, a lot of users may install ad blockers and just forget about it without ever going in the preferences. In this case, only the default lists are activated and this combination of installed lists will be shared among all the other users who did the same. On the other end of the spectrum, power users may want to install very specific lists to fit their needs. Here, if the user ends up with either a unique or a highly unusual combination of filter lists, she will make herself more visible on the Internet than if she stuck with the default ones.

Impact of versions. Different versions of the same filter list can provide a little bit of diversity when trying to distinguish users for tracking as the lists between users are not updated at the same time. However, the time frame in which these can be exploited is very short since their longevity is not longer than a few days. An attacker must account for quick changes if she wants to rely on version information to identify users.

Bypassing the ad blocking protection. Another privacy risk of identifying a filter list and its version is the ability of an attacker or a website to bypass it. By knowing which rules block a specific element on a page, an attacker can dynamically change the way this element is injected so that it is not blocked. This technique has been a constant cat-and-mouse game with popular websites like Facebook [29] where ad blockers always react rapidly to the new ways ads are presented. With our attack presented here, this could further intensify this exchange as websites would have an additional tool to identify with precision which lists are running on the client's side.

Beyond the filter lists. While lists like EasyList or EasyPrivacy are very generic and cover an incredible amount of websites, others are a lot more specific and contain very few rules. This can provide some insight into the user's preferences and what she is used to visit. For example, if the user installs lists blocking elements only on Twitch [12], Dailymotion [10] or Crunchyroll [34], this implies that she may be a heavy user of these services. Some other lists are based on topics and while some appears benign like one removing Rickroll links [42] or another stopping autoplay on videos [11], some are more personal blocking gambling [32] or religious [7] websites.

This type of information may not seem like much but it can be the first stepping stone towards learning more of the user's habits. Detecting a list can be combined with other techniques that can provide a lot of information on the user when combined. Karami et al. showed that it was possible to learn about the user's health condition, political leanings or religious beliefs by simply detecting installed extensions [21]. Gulyás et al. showed that abusing URL login redirection can expose if a user is logged into a specific website or not [18]. Mishra et al. demonstrated that collecting regular IP addresses from a client can provide insight into her routine [26]. In the end, even if sometimes collecting little bits of information may not represent much, it has been shown that it can still represent a real privacy risk by revealing some behaviors of the user.

6.2 Future of ad blocking

Looking more towards the future, it is unclear how long ad blockers based on filter lists will sustain or how specific changes to web browsers will impact the technique described in the paper. Google announced in October 2018 that they will introduce Manifest V3 to increase the security of browser extensions [41]. The most controversial change that has been discussed a lot over the past two years on the Internet is the plan to replace the *webRequest* API by the *declarativeNetRequest* API [4, 8, 19, 40]. The main complaint about this new API is that it will greatly hamper the capabilities of blocking extensions by setting restrictive caps on the number of accepted rules [35]. Looking at our dataset, the current cap of `30,000` rules would directly impact 13 different filter lists with popular ones like EasyList and AdGuard Base included in them. Regarding our technique, we believe such a change would not impact it as we base our detection on the latest additions or deletions of specific lists. It also remains to be seen which browser vendors will follow such changes as Microsoft is already adopting it [36] but Mozilla decides not to implement Manifest V3 fully [22].

Finally, new approaches are being developed to block ads called perceptual ad blocking [37]. By looking at what is actually rendered by the browser, a machine learning model runs to decide if an element should be blocked or not. As a proof of concept, Abi Din et al. have implemented a prototype inside Chromium and Brave [38]. They note that the model must be retrained and updated often to defend against adversarial attacks. Such updates would be similar to how filter lists are updated today and we believe our approach could be transposed to detect which ML model and which version of the model is running on the client's side.

6.3 Defenses

The attack presented in this paper exploits the diversity among filter lists thus, we propose the following two defenses which attempt to mitigate the effectiveness our approach by reducing the natural diversity of filter lists:

(1) As the risk of detection increases when multiple lists are used, a *mega list* containing all the different rules from individual filter lists may decrease the diversity and homogenize the attack surface of our technique. Such a list would thus render our detection technique useless as none of the filter lists will have a unique set of rules. An example of such a list already exists: BlockConvert [19] consolidates many lists from a wide variety of sources and it currently has more than `788,000` rules. However, this method has its own drawbacks as it directly increases the performance cost of loading a page. On less powerful mobile devices, it would be most taxing.
(2) To further reduce their diversity, ad blockers may enforce each user to have the same version of a list at a given point of time. Such a policy would greatly reduce the diversity of filter lists among users as every user will always have the same version. Ad blockers could implement this by checking for updates very frequently but one important remaining issue is to figure out the best pace for updates. Waiting too long could render some fast changing rules obsolete while updating too often would increase the server costs for the list providers, which might be not desirable or feasible for small volunteer based filter lists community.

6.4 Limitations

Parsing pipeline. We showed in Section 4 that relatively high results were obtained for the lists detection and even though we should have been able to detect the right lists 100% of the time in theory, the score has been dragged down by multiple issues. One of these issues is explained by the limitations we encountered when parsing the rules and generating DOM elements to use in the attack. We were able to parse the vast majority of rules and generate valid resulting HTML elements. Some rules could however not be parsed: these includes rules in which we encountered a chaining of options, including options that had the key/value format. These were too complex to be parsed and were useless for our attack most of the time, as they were triggered on very specific elements, such as popups, or on a websocket connection.

Rules uniqueness. Another explanation for the obtained score can be encountered in the unique rules that were determined. Lists may contains overlapping rules that are still defined relatively differently than each other. These rules generate different HTML elements, but are still blocked by each others. For instance, one list may block every element whose ID contains the word "*ads*", while the other may block elements whose ID contains a combination of the previous word and another word. The obvious consequence of such similar rules is that the first list inevitably blocks the element which adhere by the second rule, triggering a false positive. As the rules can be defined in a more and more flexible way, thanks to options and a more complete rule syntax, lists may contain a growing number of overlapping rules that are defined in multiple ways but still block the same elements.

Another limitation of our technique involves the choice of lists to be tested. Extracting unique rules for every lists involves comparing the lists between each others. If the chosen lists are all built by using another list present in the set in a basis, this latter list will have little to no unique rules to be tested. Users which use overlapping lists may be less prone to our attack, however, overlapping lists brings the disadvantage of reduced ad blocker performances as the ad blocker has twice as much entries to parse and compare.

Version's differences. Multiple limitations arise with our technique regarding the version's detection. One of these limitations involve the number of added rules and their type. We showed in Section 4.3 that many of the additions for each new versions regarded rules of types that can not be used in our attack. This limits our technique as additions in a new version may regard these type of rules only: this result in versions that introduce changes compared to the previous version, but are not detectable using our technique.

New versions of filter lists may also introduce that were present in a given older version and then removed. These type of rules may trigger a false positive when testing for the oldest version. This issue can be bypassed by combining the rules to be tested for each lists but this is not possible for versions for which the redundant rule is the only testable addition.

Finally, we do not check for the uniqueness of the additions among all the lists as this would further restrain the number of testable rules per version. This choice leads to the risk of a rule being triggered by other lists, resulting with a false positive. However, the uniqueness of the rule is not as necessary when checking for the list's version, as added rules for each list have a higher chance of being specific to the list at the moment of the addition. This chance grows smaller as time goes as filter lists may use rules introduced by other lists. We however showed in Section 3.4 that a consequent number of filter lists are updated frequently and thus the user may end up constantly having the latest version of the list if she allows the ad blocker to constantly perform update checks.

[19]https://github.com/mkb2091/blockconvert

7 RELATED WORK

In this section, we provide an overview of used tracking techniques and defenses against them. We first present cookies based tracking methods before introducing more recent browser fingerprinting techniques used for tracking users. We finally present the latest state-of-the art blocking techniques.

7.1 Tracking Techniques

Multiple studies have emphasised the use of cookies for online tracking. Engelhardt et al. [14] showed that cookie syncing is widely used to learn the user's browsing history, and an adversary can reconstruct up to 73% of it. They later showed by crawling the Alexa Top million websites that more than 81,000 third parties were present on at least 2 pages, with Google Analytics being present in more than 60% of the crawled pages [13]. Acar et al. [1] found that hundreds of domains of the Alexa Top-3K passed unique identifiers to each other using cookies while crawling websites. Bashir et al. [3] proposed a technique to identify cookie matching in a bid to limit the flow of information between different tracking domain. Li et al. [25] developed the first machine learning based tool, called TrackAdvisor, to identify the HTTP requests carrying sensitive information to third-party trackers, achieving a very high accuracy.

In direct opposition to cookies, browser fingerprinting has been introduced more recently, and exploits the vast diversity of the browser ecosystem to forge a unique signature from a device and its configuration. Eckersley [9] first discovered this method, and was followed by several studies looking at its adoption [1, 2, 13, 27]. Laperdrix et al. [24], then Gomez et al. [17] showed that browser fingerprinting is widely used for tracking. Laperdrix then introduced FPRandom [23], a framework that helps mitigate the risks of browser fingerprinting by introducing randomness. However, Vastel et al. [39] introduced FP-Stalker, a technique that enables to track users over time, even as their browser fingerprint evolve over time.

7.2 Evolution of ad blockers

With rising awareness of users towards their privacy, ad blockers have steadily gained popularity with 236 millions desktop users and 527 million on mobile [6] in 2020. The majority of popular ad blockers use simple regex based filter lists to block advertisements and trackers. However, little work has been done on studying filter lists and their privacy implications. Snyder et al. [33] performed an extensive analysis of the *EasyList* filter list and proposed an optimization to current filter lists based ad blockers. Snyder stated that lists were rarely purged of rules that were no more usable, pointing to the fact that *EasyList* presented consequently more additions than deletions. However, our analysis led us to a different conclusion as we observed only a slight difference between the number of additions and deletions.

Several other machine learning and heuristics based filer lists as well as blocking techniques have previously been proposed in order to improve the web experience. Din et al. [38] introduced a deep-learning based approach that intercepts images generated by pages and classifies them to detect advertisements. Their novel method showed similar efficiency as filter lists, and even managed to outperform them for specific platforms such as Facebook. Iqbal et al. [20] used machine learning utilizing features such as HTML elements, HTML requests, and JavaScript content to determine if a request is malicious. Privacy Badger [20] uses learning techniques in order to build a personalized base of requests to be blocked based on the user's behavior on the web. Fouad et al. [5] analyzed the usage of invisible pixels by third-parties and introduced a classification method to identify invisible pixels belonging to third-party trackers.

8 CONCLUSION

In this paper, we present a novel attack to detect ad blockers and filter lists used by a user. By looking at the evolution of 460 different filter lists over a period of one month, we show that the diversity in filter lists can be abused to detect them as well as their specific versions in most cases. Depending on the number of updates and their composition, we find that lists like EasyPrivacy are a lot more prone to detection because they have a high number of identifiable rules with frequent changes that are unique. Others like uBlock Annoyances which are mainly composed of rules with untestable options are a lot less identifiable by an external attacker. We also design two experiments to simulate a realistic scenario where people might browse the web with a combination of multiple filter lists. We find that we can identify lists with a F-1 score of 89% while the detection accuracy of specific versions range from 100% for uBlock to 17.4% for Peter Lowe's list because of rule collisions between versions.

In the end, this study shows the true paradoxical nature of ad blockers. On one hand, they enhance online privacy for millions of users but on the other, they can represent a real privacy risk by rendering users a lot more visible to trackers. Knowing the exact set of lists installed on the client side can be used along with other information to enhance user tracking or it can be abused by advertisers to bypass protection and dynamically inject a specific ad in the DOM.

REFERENCES

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 674–689.

[2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: Dusting the Web for Fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer amp; Communications Security (CCS '13)*. Association for Computing Machinery, New York, NY, USA, 1129–1140. https://doi.org/10.1145/2508859.2516674

[3] Muhammad Ahmad Bashir, Sajjad Arshad, William Robertson, and Christo Wilson. 2016. Tracing information flows between ad exchanges using retargeted ads. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 481–496.

[4] Karan Bhatia. 2018. *Issue 896897: Extensions: Implement Manifest V3 – Chromium bug tracker.* https://bugs.chromium.org/p/chromium/issues/detail?id=896897

[5] Nataliia Bielova, Arnaud Legout, Natasa Sarafijanovic-Djukic, et al. 2020. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. *Proceedings on Privacy Enhancing Technologies* 2020, 2 (2020), 499–518.

[6] Blockthrough. 2020. *Growth of the Blocked Web – 2020 PageFair Adblock Report.* https://s3.amazonaws.com/media.mediapost.com/uploads/2020-PageFair_Blockthrough-Adblock-Report.pdf

[7] Chris Buijs. 2020. *Religious domains – Filter list.* https://raw.githubusercontent.com/cbuijs/shallalist/master/religion/domains

[8] Devlin Cronin. 2019. *Improving Security and Privacy for Extensions Users – Chromium blog.* https://security.googleblog.com/2019/06/improving-security-and-privacy-for.html

[20]https://privacybadger.org/

[9] Peter Eckersley. 2010. How Unique Is Your Web Browser?. In *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings (Lecture Notes in Computer Science)*, Mikhail J. Atallah and Nicholas J. Hopper (Eds.), Vol. 6205. Springer, 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[10] Imre Kristoffer Eilertsen. 2020. *DailyMotion Simplicity – Filter list.* https://raw.githubusercontent.com/DandelionSprout/adfilt/master/DailyMotionSimplicity.txt

[11] Imre Kristoffer Eilertsen. 2020. *Stop Autoplay On Video Sites – Filter list.* https://raw.githubusercontent.com/DandelionSprout/adfilt/master/StopAutoplayOnYouTube.txt

[12] Imre Kristoffer Eilertsen. 2020. *Twitch: Pure Viewing Experience – Filter list.* https://raw.githubusercontent.com/DandelionSprout/adfilt/master/TwitchPureViewingExperience.txt

[13] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security.* 1388–1401.

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

[15] Alexandros Fakis, Georgios Karopoulos, and Georgios Kambourakis. 2020. Neither Denied nor Exposed: Fixing WebRTC Privacy Leaks. *Future Internet* 12, 5 (2020), 92.

[16] Ghostery and Cliqz. 2017. *Tracking the Trackers: Analyzing the global tracking landscape with GhostRank.* https://cliqz.com/en/magazine/ghostery-study

[17] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference.* 309–318.

[18] Gábor György Gulyás, Dolière Francis Somé, Nataliia Bielova, and Claude Castelluccia. 2018. To Extend or not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, and Aaron Johnson (Eds.). ACM, 14–27. https://doi.org/10.1145/3267323.3268959

[19] Avatar Gregory Huczynski. 2018. *Chrome extension manifest v3 proposal – uBlock-issues GitHub repository.* https://github.com/uBlockOrigin/uBlock-issues/issues/338

[20] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. 2020. Adgraph: A graph-based approach to ad and tracker blocking. In *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, 763–776.

[21] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020.* The Internet Society. https://www.ndss-symposium.org/ndss-paper/carnus-exploring-the-privacy-threats-of-browser-extension-fingerprinting/

[22] Philipp Kewisch. 2019. *Mozilla's Manifest v3 FAQ – Mozilla Add-ons Blog.* https://blog.mozilla.org/addons/2019/09/03/mozillas-manifest-v3-faq/

[23] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *International Symposium on Engineering Secure Software and Systems.* Springer, 97–114.

[24] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP).* IEEE, 878–894.

[25] Tai-Ching Li, Huy Hang, Michalis Faloutsos, and Petros Efstathopoulos. 2015. Trackadvisor: Taking back browsing privacy from third-party trackers. In *International Conference on Passive and Active Network Measurement.* Springer, 277–289.

[26] Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Martin Lopatka. 2020. Don't Count Me Out: On the Relevance of IP Address in the Tracking Ecosystem. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 808–815. https://doi.org/10.1145/3366423.3380161

[27] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. 2013. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In *2013 IEEE Symposium on Security and Privacy.* 541–555.

[28] Lara O'Reilly. 2015. *The inventor of Adblock tells us he wrote the code as a 'procrastination project' at university — and he's never made money from it.* https://www.businessinsider.com/interview-with-the-inventor-of-the-ad-blocker-henrik-aasted-srensen-2015-7

[29] Lara O'Reilly. 2016. *Facebook is locked into a seemingly neverending game of cat and mouse with ad blockers.* https://www.businessinsider.com/facebook-versus-ad-blockers-is-a-neverending-ping-pong-game-2016-8

[30] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos Markatos. 2019. Cookie Synchronization: Everything You Always Wanted to Know But Were Afraid to Ask. In *The World Wide Web Conference (WWW '19).* Association for Computing Machinery, New York, NY, USA, 1432–1442. https://doi.org/10.1145/3308558.3313542

[31] Adblock Plus. 2020. *Adblock Plus filters explained.* https://adblockplus.org/filter-cheatsheet

[32] Sinfonietta. 2020. *Gambling hosts – Filter list.* https://raw.githubusercontent.com/Sinfonietta/hostfiles/master/gambling-hosts

[33] Peter Snyder, Antoine Vastel, and Ben Livshits. 2020. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. In *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '20).* Association for Computing Machinery, New York, NY, USA, 75–76. https://doi.org/10.1145/3393691.3394228

[34] Cardinal System. 2020. *Crunchyroll Anti-Circumvention filters – Filter list.* https://raw.githubusercontent.com/TheCardinalSystem/Cruncyroll-Filter-List/master/english.txt

[35] Chrome team. 2019. *Properties of chrome.declarativeNetRequest – Google Chrome API documentation.*

[36] Microsoft Edge Team. 2020. *Manifest V3 changes are now available to test in Microsoft Edge – Microsft Edge blog.* https://blogs.windows.com/msedgedev/2020/10/14/extension-manifest-chromium-edge/

[37] Florian Tramèr, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. 2019. AdVersarial: Perceptual Ad Blocking meets Adversarial Machine Learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2005–2021. https://doi.org/10.1145/3319535.3354222

[38] Zain ul Abi Din, Panagiotis Tigas, Samuel T. King, and Benjamin Livshits. 2020. PERCIVAL: Making In-Browser Perceptual Ad Blocking Practical with Deep Learning. In *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, Ada Gavrilovska and Erez Zadok (Eds.). USENIX Association, 387–400.

[39] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP).* IEEE, 728–741.

[40] Simeon Vincent. 2019. *Blog posts on the motivations for webRequest changes – Chromium.org Google group.* https://groups.google.com/a/chromium.org/forum/#!msg/chromium-extensions/qFNF3KqNd2E/8R9PWdCbBgAJ

[41] James Wagner. 2018. *Trustworthy Chrome Extensions, by default – Chromium blog.* https://blog.chromium.org/2018/10/trustworthy-chrome-extensions-by-default.html

[42] Jamie Wilkinson. 2020. *Rickroll Blacklist – Filter list.* https://rickrolldb.com/ricklist.txt