Errors, Misunderstandings, and Attacks: Analyzing the Crowdsourcing Process of Ad-blocking Systems

Mshabab Alrizah
The Pennsylvania State University
maa25@psu.edu

Sencun Zhu
The Pennsylvania State University
sxz16@psu.edu

Xinyu Xing
The Pennsylvania State University
xxing@ist.psu.edu

Gang Wang
University of Illinois at Urbana-Champaign
gangw@illinois.edu

ABSTRACT

Ad-blocking systems such as Adblock Plus rely on crowdsourcing to build and maintain filter lists, which are the basis for determining which ads to block on web pages. In this work, we seek to advance our understanding of the ad-blocking community as well as the errors and pitfalls of the crowdsourcing process. To do so, we collected and analyzed a longitudinal dataset that covered the dynamic changes of popular filter-list EasyList for nine years and the error reports submitted by the crowd in the same period.

Our study yielded a number of significant findings regarding the characteristics of FP and FN errors and their causes. For instances, we found that false positive errors (*i.e.*, incorrectly blocking legitimate content) still took a long time before they could be discovered (50% of them took more than a month) despite the community effort. Both EasyList editors and website owners were to blame for the false positives. In addition, we found that a great number of false negative errors (*i.e.*, failing to block real advertisements) were either incorrectly reported or simply ignored by the editors. Furthermore, we analyzed evasion attacks from ad publishers against ad-blockers. In total, our analysis covers 15 types of attack methods including 8 methods that have not been studied by the research community. We show how ad publishers have utilized them to circumvent ad-blockers and empirically measure the reactions of ad blockers. Through in-depth analysis, our findings are expected to help shed light on any future work to evolve ad blocking and optimize crowdsourcing mechanisms.

ACM Reference Format:
Mshabab Alrizah, Sencun Zhu, Xinyu Xing, and Gang Wang. 2019. Errors, Misunderstandings, and Attacks: Analyzing the Crowdsourcing Process of Ad-blocking Systems. In *Internet Measurement Conference (IMC '19), October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3355369.3355588

IMC '19, October 21–23, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6948-0/19/10…$15.00
https://doi.org/10.1145/3355369.3355588

Ad-block Extension	Default Filter List	Firefox	Chrome
Adblock Plus	EasyList	11,054,048	> 100 million
AdBlock	EasyList	1,329,314	> 40 million
uBlock Origin	EasyList	4,861,128	> 10 million
Adguard AdBlocker	AdGuard	337,776	5,641,143
AdBlocker Ultimate	AdGuard	413,605	731,041
μBlock	EasyList	98,894	884,482

Table 1: Number of active devices using Firefox and Chrome extension that have EasyList as a default filter-list in the first week of January 2019. Note that AdGuard is based on EasyList [2].

1 INTRODUCTION

Ad-blocking systems are widely used by Internet users to remove advertisements from web pages and protect user privacy from third-party tracking. Today, over 600 million devices are using ad-blocking systems around the globe [19].

Crowdsourcing is a crucial mechanism adopted by ad-blocking systems to introduce new filter rules and mitigate filter errors. For example, many popular ad blockers depend on a crowdsourcing project called EasyList [21]. EasyList maintains a list of filter rules that determine which ads to block. Among all the available filter lists, EasyList has the largest user base [11, 47]. Table 1 shows some popular Firefox and Chrome ad-blocking extensions that depend on EasyList. As of the first week of January 2019, more than 173 million active devices used EasyList for ad blocking [17, 49, 86]. Furthermore, Google uses ad-related URL patterns based on EasyList to block ads on sites that fail the Better Ads Standards (in Chrome for both desktop and Android version) [14].

To facilitate crowdsourcing, EasyList has established a large community through which users can provide feedback and report errors to the EasyList editors, who then update the ad-blocking filters manually. These editors are a small group of EasyList authors and ad-blocking experts. The editors collect feedback and reports from the crowd using two different channels: public forums and plug-in applications [21].

The evolution of ad-blocking systems has an influence on the practices of both website owners and web users in multiple ways. Therefore, a substantial amount of research has addressed ad-blocking systems from a range of perspectives including those involving the relationships among Internet users, ad publishers, and ad blockers [48, 53, 71, 89]. Other research has focused on the economic ramifications of the ad-blocking systems [19, 78], potential complementary solutions [26, 83], and specific cases of ad blocking (*e.g.*, tracker blocking, anti-adblocking) [25, 30, 46, 82]. However, there

remains a lack of deep understanding about how crowdsourcing functions improve the filter-lists of ad-blocking systems over time, and the potential pitfalls and vulnerabilities in this process.

In this paper, we focus on EasyList and present a measurement study on the dynamic changes of the filter-list and the crowdsourcing error reports over a period of nine years. We explored the answers to a series of key questions. First, how prevalent are the errors of blocking legitimate content, *i.e.*, false positive (FP) errors, and how prevalent are the errors of missing real advertisements, *i.e.*, false negative (FN) errors? Second, what are the primary sources of these errors? Third, how effective is *crowdsourcing* in detecting and mitigating false positive and false negative errors? Fourth, how *robust* is the filter-list against adversaries?

To answer these questions, we collected two large datasets that covered the update history of EasyList and the behavior of the ad-blocking community over the nine-year period from November 2009 to December 2018. The resulting dataset contains a total of 55,607 versions of the filter lists. Over the nine years, there were 534,020 filters added and 448,479 filters removed by 27 editors to correct both FP and FN errors. Additionally, we also crawled the feedback reports submitted by users in the community. We collected 23,240 reports of FP and FN errors. Moreover, we analyzed 0.5 million records of traffic history of 6,000 ad servers.

To effectively connect the first two datasets, we proposed and utilized some tools and simulation methods to create and extract real instances of FP and FN errors. Then, we performed an in-depth analysis of the occurrence of both types of errors to understand their causes and to evaluate the effectiveness of crowdsourcing for error detection and mitigation. Based on the result, we analyzed 15 different type of attacks which aim to circumvent ad blockers, including 8 new attacks that have not been systematically studied by existing literature.

Our study yielded a number of significant findings regarding the characteristics of FP and FN errors and their causes. *First*, a non-trivial portion of the community effort was spent on addressing FP errors (about 30% of the reports). *Second*, despite the community effort, there was still a long delay of FP error discovery. More than half of the errors persisted for over a month before they were reported. Once reported, it took, on average, 2.09 days to push the updates. *Third*, about 65% of the FP errors were caused by the "bad signatures" added by the EasyList editors. However, the website owners/designers made it worse by using *already-blacklisted* elements for their web content design (accounting for 35% of the errors). *Fourth*, FN errors were more likely to be rejected or ignored by EasyList editors. Some of the reported FNs are not real errors. For example, certain reporting users thought their ad blocker was failing to block certain ads, but the true situation was that their computers were infected by adware that was overwriting the ad blocker and injecting ads.

Our study also provided insights into understanding the practical evasion efforts by ad publishers and website developers to circumvent the ad-blocking systems and the countermeasures from ad-blockers. We find 15 types of attack methods used in our dataset. Among them, we analyzed 8 new attacks that have not yet been systematically studied in previous works. Among the many things, we find that only 60% of ad servers were influenced when their domains were blocked by the EasyList. Ad networks aggressively change their domains and ad elements and use new strategies to achieve evasion. Our analysis shows that the countermeasures from EasyList are often delayed or ineffective.

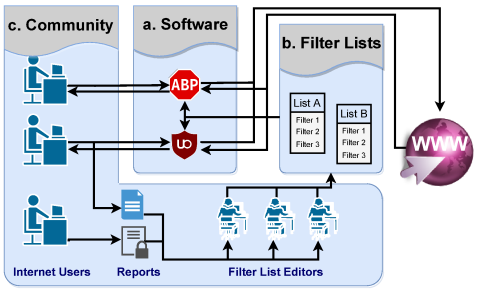

Figure 1: Overview of ad-blocking system.

The key contributions of our work are as follows:

(1) We collected two large datasets from the ad-blocking community to analyze the crowdsourcing process for ad-blocking error detection and mitigation. We plan to share our datasets with the research community.
(2) We presented an in-depth analysis of the accuracy of the ad-blocking system and the updating behavior of the filter list over 9 years. This analysis provides new insights into the causes of false positive and false negative errors and their impacts on websites.
(3) We presented a comprehensive analysis of the vulnerabilities of the ad blocking system by illustrating 15 different evasion methods and their empirical usage.

2 AD-BLOCKING DATASETS

Popular ad-blocking systems commonly have three main components: (a) ad-blocking software, (b) a filter list, and (c) a community of users who provide feedback to refine the filter list. As shown in Figure 1, the ad-blocking software utilizes the filter lists to block ads on web pages. The community, which consists of Internet users and ad-blocking filter-list editors, has two leading roles: contributing new filter rules and correcting errors caused by the filters. The Internet users, who use ad-blocking systems, contribute to the filter lists by reporting errors, while the editors respond by interpreting and acting on that feedback in order to control the list. Very few systems use hard-coded or fixed filter lists, which are typically less popular and less effective for ad blockers. Therefore, we do not consider them in this paper.

To perform the measurement study, we collected two datasets from different sources. The datasets include 1) the nine-year history of all versions of the EasyList filter list, and 2) users' FP and FN error reports for the same period. In the ad-blocking system, an FP error is an instance when the filter incorrectly blocked a non-advertisement element on a website, and an FN error is an instance when the filter fails to block a real ad on a website. For our analysis, we implemented multiple tools to collect and clean the datasets.

2.1 D1: EasyList Dataset

From November 30, 2009, to December 7, 2018, there were 117,683 versions of EasyList. The ad-blocking system updates EasyList when

the editors create a new version and push it to the remote repository. There are three reasons for updating EasyList: correcting FP errors, correcting FN errors, or modifying the order/structure of existing filter list. We will focus primarily on error correction related updates.

We crawled the Mercurial repository that tracks the changes to EasyList [45], which has the histories of updates back to November 2009. First, we tabulated the shortlog (commits) over the nine years to build an index of changes. Second, we used the index to seed an extractor of the EasyList changes. The extractor tracks the differences between the old and new EasyList versions. This allowed us to rebuild all the EasyList versions. Third, we focused on each filter in the list and tracked the changes by building the image of its lifetime.

Basic Data Cleaning. The biggest challenge for our analysis is that we could not merely compare the differences between two consecutive versions to detect the added or removed filters over time. Below, we discuss the reasons and how we resolved the issues.

First, *data synchronization problem.* EasyList editors may not always work in a synchronized way. After certain FP/FN errors are reported, different editors may try to address the same error at different times, leading to duplicated filters. We considered these changes as the noise and removed them.

Second, *temporally duplicated filters.* Duplicated filters may temporally exist in certain versions due to the operation of editors. For example, EasyList once contained two filters: (1) `missoulian.com###PG_fb` and (2) `missoulian.com,theolympian.com###PG_fb`. Actually, the editor was trying to modify the first filter by adding the website domain "theolympian.com" in the optional field to specify the filter scope. Instead of directly modifying the existing filter, the editor added the second filter in a new version and then removed the first filter later. This led to duplicated filters temporally existing in the list. For our analysis, we found these cases and merged the versions. More specifically, we created a record of each filter that indicates the time of the filter creation, the time of the filter removal, and the time period between the filter creation and removal (called filter lifetime). We leveraged the commit messages of EasyList creation to identify the reason for removing the filters. If the commit message indicated that the reason was to remove duplication, then we skip the version of the temporally duplicated filters.

Third, *structure maintenance.* Editors may reorder the filter list and rebuild some filters' syntax to create new versions (*e.g.*, the merge of the EasyList list and Fanboy list in 2011). Since these changes are not related to FP/FN errors, we did not consider these versions in our analysis.

With these complications in mind, we extracted the changes in EasyList by applying a greedy algorithm. The idea was to do a look-ahead-search to match with versions related to error corrections.

Dataset Statistics. After building and cleaning the dataset, we obtained a total of 55,607 versions as our dataset **D1**. The number of filters in each version increased almost linearly from 3,250 in November 2009 to more than 73,000 in December 2018. Over the nine-year period, there were 534,020 filters added and 448,479 filters removed in order to correct FP and FN errors. Some filters were added and removed more than once at different times. In total, there were 27 editors who maintained the list according to the crowdsourced reports.

2.2 D2: Crowdsourced Report Dataset

There are two channels for users to report errors to the EasyList editors. First, a user can report an error by submitting a public report on the EasyList forum [24]. Second, a user can submit the report via a browser plug-in. A key difference between the browser plug-in reports and the forum is that browser plug-in report is a one-way communication— users submit the reports but never get replies or feedback. Intuitively, for website owners/developers, they are more likely to submit to the forum to interact with EasyList editors and follow up on the error correction. In March of 2018, a few browser extension developers started to use emails to communicate with the reporters. Considering that such plug-in channels only started very recently (and its data is not public), our data collection was focused on the public forum that matched the period covered in D1. A key benefit of forum reports is that EasyList editors usually reply with the "new EasyList version" created to address the reported issues, which is the key to answer our research questions.

In the forum, FN reports are entered under "*Report unblocked content*", and FP reports are entered under "*Report incorrectly removed content*". Usually, errors related to the same website are grouped in one *topic* titled by the domain name of that website. Each topic might have a discussion thread with back-and-forth replies.

We built a crawler that collected all the available posts on the forum. In total, we have 23,240 topics with at least one report; 17,968 topics are about FN errors, and 5,272 topics are about FP errors. We refer to this dataset as **D2**.

3 METHODOLOGY

To analyze errors in EasyList, we first need to extract real FP and FN errors from the noisy datasets. In the following, we present our methodologies for information extraction.

3.1 Linking Reports with EasyList

To understand how crowdsourcing reports impact EasyList updates, we first needed to link the two datasets. More specifically, we sought to accurately identify which EasyList version was created as a result of a given report. Our linking method was based on a few observations. Under a crowdsourcing report, EasyList editors often refer to the *ChangeSet* [44] links in the Mercurial repository. This is a confirmation of the correction of an FN or FP error in the refereed EasyList version. Similarly, users (reporters) also use the ChangeSet links to refer to the filters or EasyList versions that caused the errors, or the ones where the errors were solved.

Based on these observations, we scanned the collected forums' topics and their threads/replies. We considered the post details if they met the following requirements: (1) The reply had a ChangeSet link; (2) The reply was created by EasyList editors; (3) The timestamp of the EasyList version in the ChangeSet link was after the timestamp of the report; and (4) The time gap between the timestamp of the EasyList version in the ChangeSet link and the timestamp of the reply was within a day.

It is worth mentioning that there is a probability that an editor corrected an error but did not reply to the reporter and confirm the

correction. In this case, there is no way for us to link the report to the EasyList version. Moreover, the editor might return an old ChangeSet link, which had been created more than a day before the editor's response or was created by another editor. In this case, we do not have strong evidence that an EasyList version was created in response to that, or to a similar report. The editors occasionally returned an old ChangeSet link to indicate that an error has been resolved. Therefore, the inclusion criteria listed above were designed to be conservative, and the successfully linked pairs selected by our approach are trustworthy (manually confirmed).

After this step, we had 5,284 reports in total. There were 3,700 reports about FN errors submitted by 758 users and 1,584 reports about FP errors from 801 users. The covered time span was from November 2009, to December 2018. We call this sub-dataset as **D2A**.

3.2 Reproducing FPs

To deeply analyze FP errors, we simulated the errors using old versions of EasyList and the web pages that were impacted. There were challenges to simulate the errors and pinpoint the influenced elements of the reported web pages.

Challenges of Reproducing FPs. Each website has a different structure that varies over time. Consequently, many FP errors occurred in the past do not exist in subsequent structures. This was the first challenge we faced, and we used the Internet Archive: Wayback Machine service [7] (Wayback Machine) to extract old versions of the websites.

The second challenge was how to recognize elements impacted by FP errors. Usually, EasyList simultaneously blocks and hides more than one element on the same page due to multiple matched filters. Sometimes it even filters out dozens of elements, the majority of which are ads, making it an arduous task to distinguish between true positives (TP) results and FP errors.

We identified the web page elements impacted by an FP error by utilizing the difference between two successive EasyList versions: the version that corrected the error and the version that was created before it. We designed our method to identify the legitimate elements that were falsely blocked by using a reverse method of ad blocking. The essence of the idea is that each EasyList version has its scope that depends on the scopes of its filters. When a new FN error is encountered, a new version is created to expand the range of EasyList to cover the ads in the page that has the error. For an FP error, the scope of EasyList is shrunk by adding new exception rule(s) or/and removing filter(s).

Identifying the Affected Elements by FP Errors. Next, we introduce the detailed methodology to extract web page elements impacted by an FP error (*i.e.*, the EasyList filter). Let Υ be the set of elements that are affected by the bad filters (FP-error-causing filters). In other words, Υ represents legitimate elements that are misidentified as ads. Let Z_k be EasyList version k created to fix an FP error. Let Z_{k-1} be the EasyList version immediately before EasyList version k, which contains the false-positive-causing filters. Here the goal is to identify Υ by matching Z_k and Z_{k-1} against the affected web page.

First, we consider EasyList version k. Let N be the set of elements in the web page and $F_i(N)$ be a set of ad elements matched by filter i. Typically, an EasyList filter may also match legitimate elements and thus the filters often come with *exception rules*. Here, $E_j(N)$ which represents a set of non-ad elements matched by exception rule j and overwrites the output of $F_i(N)$. Using the EasyList version k, the blocked ad elements in the web page are:

$$S = \bigcup_{i \in Z_k} F_i(N) - \bigcup_{j \in Z_k} E_j(N)$$

Next, we backtrace to version Z_{k-1} where there are FP-error-causing filters. To identify Υ (the set of legitimate elements that were incorrectly blocked by filters in version Z_{k-1}), we preform *set subtraction* between the set of elements blocked by Z_{k-1} and the set of elements blocked Z_k. As a result, legitimate elements blocked by Z_{k-1} are:

$$\Upsilon = \left(\bigcup_{i \in Z_{k-1}} F_i(N) - \bigcup_{j \in Z_{k-1}} E_j(N) \right) - \left(\bigcup_{i \in Z_k} F_i(N) - \bigcup_{j \in Z_k} E_j(N) \right)$$

To better explain the process above, we use a toy example. Suppose Z_1 is EasyList version 1, and Z_2 EasyList version 2 created to correct the FP error in version 1. A web page has elements N = $\{n_1, n_2, n_3, n_4, n_5, n_6\}$. The ad elements are $\{n_1, n_2, n_3\}$. Z_1 has filters $\{f_1, f_2\}$ and an exception rule $\{e3\}$ that prevents n_6 from being blocked because it is not an ad element. As a result, suppose f_1 blocks $\{n_1, n_2, n_5\}$ and f_2 blocks $\{n3\}$. e_3 excepts $\{n_6\}$ from being blocked. Since n_5 is not an ad, f_1 causes an FP error.

In our toy example, Z_2 is then created to correct the above FP error by adding a new exception to the list: $\{f_1, f_2, e_3, e_4\}$ where the new rule e_4 excepts $\{n_5\}$. To find n_5, we apply the above equation $\Upsilon = (\{n_1, n_2, n_3, n_5\} - \{n_6\}) - (\{n_1, n_2, n_3, n_5\} - (\{n_5, n_6\}) = \{n5\}$. As illustrated above, $\{n5\}$ is exactly the element affected by the FP error.

In this way, we built a list of CSS selectors to identify the Document Object Model (DOM) of Υ that was used when we scanned the Wayback Machine. We scanned 2,203 web pages from different websites that were reported with FP errors in **D2A**.

Limitations of using Internet Archive. We are aware of the potential limitations of Internet Archive[40, 41, 88]. Here, we want to briefly discuss how such limitations affect our analysis. First, the frozen ads problem. Because the archived webpages are no longer "live", the ads are frozen (*i.e.*, not loading from the live source). However, this does not impact our analysis since we focus on FP, *i.e.*, legitimate web elements blocked by EasyList filters, not the ad elements. Second, the nearest-neighbor timestamp problem. It is a known limitation that archived snapshots are incomplete. The consecutive snapshots might have a big time gap in between. For our analysis, we utilized the Wayback CDX Server API [66] to check the nearest snapshot to our requested date to make sure the dates are close by (309 web pages are eliminated due to the lack of snapshots). Third, the absence of archived web page problem. Certain web pages were not achieved due to the Robots Exclusion Protocol [1] or the websites were low-ranked. We found 76 websites in our analysis were not archived. Additionally, errors that could only be triggered by logging-in were not simulated. Other issues such as "archive-escapes destination" and "same-origin escapes" do not apply to our context.

Error Duration. The lifetime or duration of the FP error on each website started when the error-causing filter matched the non-ad element(s) on the website and lasted until the error was corrected. Thus, we should be able to find the duration of the impacted element on the website and the duration of error-causing filter on the filter list. The overlap between these intervals until error correction is the duration of an FP error.

To find the duration of each impacted element, we built a Java application to scan all the snapshots of impacted websites. We utilized the Wayback CDX Server API. The API enables complex queries that return the list of web page snapshots captured by Wayback. We used it to specify our queries, e.g., excluding permanent URL redirections [18]. We removed 82 web pages that had permanent URL redirection. We utilized the HtmlUnit, which is a "GUI-Less browser for Java programs"[29]. It is hard to use Java applications to emulate browser behavior, such as executing JavaScript. Therefore, we used HtmlUnit to emulate parts of browser behavior, including the lower-level aspects of HTTP and supporting JavaScript.

To find the duration of an error-causing filter, we used dataset D1 to identify the time when the filter was added to EasyList. Still, we could not directly identify the filter in each case. The predicament was that there were 869 cases of EasyList editors creating exception filters to correct the errors. That did not help the process of recognizing which filter caused the problem. The exception filter overrode the error-causing filter in the scope of the impacted filter. Thus, we built a Google Chrome extension to reproduce the ad-blocking effect in various circumstances. The Chrome extension is similar to the Adblock Plus extension, but it is fed with a specific list. For each case, we fed the extension with the EasyList version that was created directly before the version that was created to correct the error. Furthermore, we modified some filters to fit with the *archive.org* website. In each snapshot of any website, archive.org is the first party. For example, the URL of the snapshot of *cnn.com* in May 2013, is

```
https://web.archive.org/web/20130531-
230746/http://www.cnn.com/
```

Filters like: `/hads-$domain=cnn.com` will not block anything in this snapshot since the first party of snapshot is `archive.org`, while `http://www.cnn.com/` is the sub-domain. We changed the domain of the filters that required indicating the first party in each case to `archive.org`. Moreover, we used *Webrecorder* [84] to replay web archives and activate dynamic scripts.

FP Instances. We ended up with 570 instances (out of 2,203) that correctly emulated the FP errors. We analyzed each instance to extract the element types and feature and indicate the filter cause the FP error. This sub-dataset was designated as **D2FP**

3.3 Extracting FN Errors

EasyList depends on crowdsourcing to detect FN errors, which happens when ads are not blocked as expected. Three main parts need to be investigated in order to provide a pragmatic analysis: the crowd contribution, the EasyList reaction, and the countermeasures of ad publishers. Therefore, from dataset D2, we extracted such information from FN error reports as the profiles of users who reported the errors, the domains and explanatory words from titles, the timestamps of posting, the outgoing links from the replies, and the contents of replies. There were 17.9K FN reports submitted by 4,552 users who reported 12,866 websites.

We attempted to extract the rejected or incorrect FN reports. We classified a report as a rejected report if one or more of the following conditions were met: (1) The editors replied explicitly to reject the report; (2) The editors indicated that the report was not applicable because it was about FP errors or other issues such as adware; (3) The report was insufficient and locked (if it was not replied to or did not include the websites where the error occurred); (4) The editors referred to a ChangeSet link that belonged to another filter list, or responded with an old ChangeSet link and locked the thread; (5) The topic was locked after the editor asked a question, but there was no response from the reporter.

Conditions (4) and (5) were applied to posts with small threads, because we cannot manually validate posts with many replies. We found 3,750 reports that met at least one of these conditions. This sub-dataset was designated as **D2B**.

We studied the explanatory words from the titles and the ChangeSet links to indicate the error types generally. The forum mainly targets ad elements. However, we noticed in a previous stage that some reports were about adware and software issues. These reports were considered incorrect reports. In later sections, we will analyze errors that might have been reported due to privacy issues, anti-adblocks, and social media content that were not considered incorrect reports by EasyList editors.

3.4 Websites Involved in the Reports

Our dataset revealed that the EasyList community used the open forums to report over 12,266 websites. Some of them were mentioned in more than one report. In our analysis, we studied only the websites that were indicated in dataset D2A (reports with ground-truth). Dataset D2A includes 4,212 websites. FP reports mentioned 1,266 websites, whereas FN reports referred to 2,946 websites. Some of the websites were included in both FN and FP reports.

4 ANALYSIS

In this section, we deeply investigate the main factors related to ad-blocking systems' accuracy: the users, the websites, the types of errors, and the time. We study the association between these factors and the errors to draw a substantial picture of potential consequences.

4.1 FP vs. FN Errors

From dataset D2A, Table 2 shows that there are 1,584 FP-related reports and 3,700 FN-related reports. Even though we took conservative linking methods, FPs still took 30% of all the errors, which is a non-trivial portion. The delay of correcting FP errors, in general, was shorter than that of correcting FN errors, which also varied among different types of reporters. More specifically, the reporters were classified into seven categories according to their experience and tasks, as shown below:

- Editor: EasyList authors or maintainers
- Anonymous: guest users that are not registered in the forums
- New Member: registered users that have less than ten posts whether they are reports or not

	# Reports		Avg. of days		SD.	
Title	FP	FN	FP	FN	FP	FN
Anonymous	530	853	2.37	1.80	6.88	7.38
New Member	371	307	3.94	9.31	8.77	21.09
Senior Member	160	749	2.31	6.42	5.35	17.48
Developer	83	99	1.80	16.30	5.52	31.08
Other Lists Editor	105	603	1.65	2.65	3.86	11.02
Veteran	255	751	1.95	5.34	5.17	14.31
Editor	80	338	0.58	0.52	1.49	2.98
Total	**1,584.00**	**3,700.00**	2.09	6.05	5.29	15.05

Table 2: FP and FN error reports submitted by different categories of users: "# Reports" means the number of reports, "Avg". is the average of days between submitting the report and correcting the error, "SD" is the simple stander deviation of that periods.

	Avg. Report\User		SD.	
Title	FN	FP	FN	FP
New Member	1.97	1.16	4.61	0.50
Senior Member	6.24	2.32	18.47	2.02
Developer	8.25	13.83	17.40	11.69
Other Lists Editor	24.12	11.67	96.39	12.52
Veteran	25.90	17.00	42.68	21.28
Editor	37.56	16.00	57.99	17.22
Avg.	17.34	39.59	10.33	10.87

Table 3: Average number of FP and FN error reports submitted by each user in different categories of users: "Avg". is the average number of reports submitted by individual user, "SD" is the simple stander deviation of the number of reports.

- Senior Member: registered long-term users who have more than ten posts in the forums
- Veteran: users determined by the forums administrators as having made significant contributions to reporting errors. or correcting bugs
- Developer: ad-block software developers
- Other List Editor: authors of other filter lists such as non-English lists or privacy lists

We found that `anonymous` users have the highest contributions in reporting errors. FP errors were particularly skewed towards `anonymous` and `new Members`. FN errors were more evenly divided among four of the user categories.

EasyList editors were more concerned about FP errors. This was reflected by the fast response and standard deviation of solving the errors. However, the Editors' responses were faster to the anonymous FN reports than to the anonymous FP reports. The high portion of contribution done by anonymous and the quick response by the editors raise the concern about the credibility of the reports.

The individual contribution on reporting FP and FN errors varies. From dataset D2A, we computed the number of reports submitted by an individual user. Table 3 shows the average number and the standard deviation of FP and FN error reports submitted by each user in different user categories. We excluded the Anonymous category because the anonymous users do not have "permanent" profiles. Among all the groups, we found that the individual users in the Veteran class have the highest activities on correcting FP and FN errors. However, unlike other categories, the contribution among individual users in the Developer class tends to correct more FP errors than FN errors.

4.2 Websites Affected by FP and FN Errors

We studied the prevalence of FP and FN errors based on the set of the websites referenced in our dataset. We focused on the popularity of the websites to analyze two aspects: the estimated user population impacted by the errors, and the crowdsourcing ability to detect the errors in the websites of different ranks.

Website Popularity Ranking. For this analysis, we need to determine the popularity of a given website. To do so, we reference web ranking services that rank website based on traffic volume (*i.e.*, number of visitors). Web ranking services such as Alexa [3], Cisco Umbrella [28], Majestic [33], and Quantcast[63] serve the same purpose, and yet, their rankings are not always consistent with each other [58, 65]. In this paper, we chose to use Alexa's ranking for its coverage and the ability to access historical data. More specifically, Alexa list covers more than one million websites and provides access to the statistics for the past four years.

There are additional steps we took to improve the reliability of our analysis. First, Alexa list might be impacted by weekly patterns or daily changes [65]. As such, we utilized the premium API [5] to obtain the 3-month average rank instead of the daily and weekly average. Second, the ranking might be changing over time. Although we can obtain the past 4 years of data, our analysis covers the past 10 years. To this end, instead of looking at the specific ranking, we focused on the higher level "ranking range". We grouped the websites ranks into more coarse-grained ranges so that the ranking fluctuation would have a smaller impact on the overall conclusion. Finally, to validate our conclusion, we used another ranking service Umbrella [28] to generate another set of results for comparison. The analysis is limited to Top 1 million domains, which is the limit of Umbrella.

We first ranked the websites according to Alexa rank. Using that rank, we classified the websites into seven classes. The first class was the 500 top-ranked websites. The second class was called *No Rank* (NR) class. It contains small websites that were not ranked in Alexa because there was not enough traffic data to be analyzed by Alexa. Between these two classes, we used two high classes, one class was from 500 to 5K, and the other was from the 5K to 100K top rank. We called them class *5K* and class *100K*, respectively. We clustered the rest of the websites into three lower rank classes.

A glance at Figure 2 shows that the majority of the websites indicated in FN error reports have high ranks. Over 53% of the websites are ranked within the top 100K. A similar scenario happens in FP cases when there are more than 57.8% of the websites ranked in the top 100K. As a reference, if we used the Umbrella list, 30% of FN error reports and 44% of FP error reports of the websites would be ranked within the top 100K. However, this result does not mean that the FN errors do not exist in less popular websites. Indeed, we observed that class-NR websites (websites with no rank) also have many FN and FP errors.

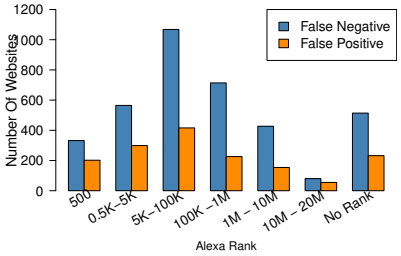

Figure 2: The number of websites that were indicated in FN and FP error reports.

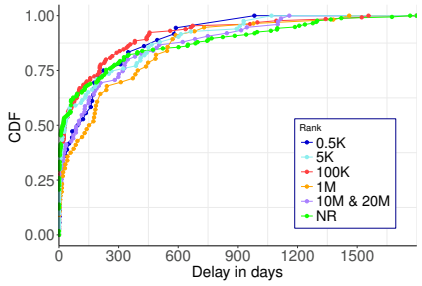

Figure 3: Distribution of delay of reporting FP errors classified by the website groups.

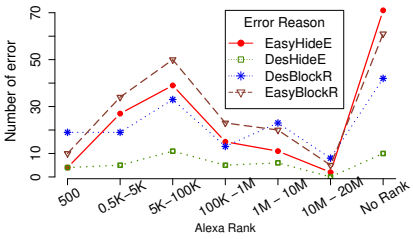

Figure 4: The frequency of four different actions that caused FP errors in websites.

	Anonymous	New Member	Senior Member	Developer	Other Lists Editor	Veteran	Editor
Likelihood Ratio	147.03	35.49	40.562	14.357	9.2607	27.911	11.097
X-squared (Pearson)	121.6966	33.30396	43.27946	13.92682	10.02684	26.67788	12.03555
P-value of X-squared	**7.17E-24**	**9.16E-06**	**1.03E-07**	0.030464	0.1235264	**0.000166**	0.0611805
Pearson correlation	**-0.05275**	**-0.158028**	0.09673657	0.061129	—	0.1041299	—

Table 4: Statistical results of Pearson's Chi-squared test and likelihood-ratio chi-squared test of independence and Pearson's correlation coefficient test of the linear trend between type of error variable and website class variable among different type of reporters.

User Categories. We also investigated the association between the error types and the website classes for *each user category*. We performed two tests. First, we tested the independence between the two variables. We used Pearson's chi-squared test and likelihood-ratio chi-squared test. Second, if the statistical tests show that there was a dependency, we measured the linear trend or correlation using Pearson's correlation coefficient. The error type variable was categorical (discrete), and the website class was ordinal. We labeled website classes scores from 1 to 7, and we gave an FP error score of 1 and an FN error score of 2 to perform Pearson's correlation test.

Table 4 illustrates statistical results of our tests. Our null hypothesis says that the types of error and website ranks are independent. We see that there is no evidence against the null hypothesis for `Editor` and `Other List Editor` users. The `Developer` class also provided weak evidence if we consider 0.01 as a significance level instead of 0.05.

On the other hand, statistical evidence shows that among `Anonymous`, `New Member`, `Senior Member`, and `Veteran` classes, the error type and website classes are not independent. The interesting point is that Pearson's correlation showed negative correlation in `Anonymous` and `New Member` cases. This indicates that these two types of users helped correct more FP errors than FN errors for lower-rank websites. Expert users such as `Senior Members` and `Veterans` showed the opposite.

4.3 Influence Time of FP Errors

Since this is the first work conducting a long-scale study on FP and FN errors of ad-blocking systems, we aim to dispatch a message to the industrial and research societies about how far these errors in systems that are used by hundreds of millions of users go. In this section, we try to answer the question of how long the websites have suffered from FP errors. We do not have the data to measure the delay of detecting FN errors and will focus on FP only.

In Section 3.2, we calculated the interval (i.e., *error duration*) between the time when the element is blocked and the errors are reported. Figure 3 shows the cumulative distribution function (CDF) of the delay in reporting the errors by the crowd. As we see, 25% of the websites had low error durations. Looking deeply into the dataset, almost 15% of all the website classes had relatively small error durations of less than one day, and less than 25% of the websites had error durations of less than four days. However, more than 50% of the websites had error durations of more than one month. All the website classes had the same trend over time. The same trend was observed when we generated the result using the Umbrella rank (omitted for brevity). The 100K class had the highest percentage in the first 600 days whereas the NR class had the lowest percentage after that interval. Finally, the NR group had the most extended error duration.

4.4 Causes of FP Errors

Who is responsible for blocking legitimate contents of web pages that are not ads? Is there an association between element types and FP errors? What (hidden) factors are there that may cause FP errors? The answers to these questions are the foundation of future work that aims to improve the accuracy of ad-blocking systems or prevent potential threats.

There are two possible reasons for FP errors. First, the errors occur because EasyList adds bad filters (unintentionally) that cause the blocking of non-ad content. Second, some website designers may create non-ad elements that are already in the scope of EasyList filters. Either way, ad-blocking software performs one of two actions: blocking the HTTP GET request or hiding the page element.

According to these observations, we break down the responsibilities of FP errors in Table 5 using the dataset **D2FP**. Depending on who introduced the elements first, the other party is the responsible party. For example, if the website designers first used an element

EasyList Action	Element Type	Designer's Failure					Ad-blocker's Failure				
		5K	100K	20M	NR	Total	5K	100K	20M	NR	Total
Hide Element	Style Element	1	3		1	0.9%	2	2	4	10	3.2%
	Embedded Content		1	2		0.5%	1	8	3	13	4.4%
	Form	1	2			0.5%	2	3	1		1.1%
	Interactive Element		1	1		0.4%	1	2	1	7	1.9%
	Link Element	1				0.2%	2	2	2	23	5.1%
	Section	6	4	5	9	4.2%	**23**	**22**	17	25	**15.3%**
	% of Total	1.58%	1.93%	1.40%	1.75%	**6.7%**	5.44%	6.84%	4.91%	13.68%	**30.9%**
Block Request	Document Metadata	1	2	4	1	1.4%	5	16	10	6	6.5%
	Embedded Content	10	4	12	4	5.3%	5	6	9	15	6.1%
	Links	9	4	**18**	**32**	**11.1%**	14	14	**24**	**29**	**14.2%**
	Scripting	**18**	**23**	6	5	**9.1%**	20	14	5	11	8.8%
	% of Total	6.67%	5.79%	7.02%	7.37%	**26.8%**	7.72%	8.77%	8.42%	10.70%	**35.6%**
	% of Total FP Error					**33.5%**					**66.49%**

Table 5: Count of FP errors, which happened in web pages, classified according to the type of impacted elements and the responsibility for the error among the classes of websites.

Filter Srource	%
FN error in home pages	31.16%
FN error in (internal) content pages	23.24%
Modifying existing filter	14.08%
From merging with other filter-lists	10.74%
Making filter more generic	4.23%
No information available (before 2009)	16.55%

Table 6: Sources of FP-error-causing filters in EasyList.

name for legitimate content, then the ad blocker who introduced the filter that caused the FP error is the responsible party. We analyzed the element types that were impacted by FP errors. We used the W3C [81] standards to cluster and name the elements. Specifically, we used HTML 5.2 standard [79], published on 14 December 2017. We clustered the elements according to the EasyList action and element type, as Table 5 illustrates.

Table 5 shows that ad-blockers had more responsibility (the source of the error) than the website designers since ad-blockers caused 65% of the errors. A large proportion was on small websites (`NR`). Looking at dataset D1, the majority of ad-blocking failures happened because they used generic filters with broad scopes. The "Sections" element types such as article, body, and header, had the highest number. The ad-block software hid this element using the element's attributes. The second type is the element type "Links" that refers to an element that has an `src` attribute. Ad-blockers block an HTTP request made to its source. The filter used is a generic string that is utilized as a signature to match the URL.

On the other hand, a large percentage of the designers' failures occurred because of using strings in domains and URLs that were already matched by filters, especially when they used CSS pop-ups. Moreover, it merits to mention that designers of the top 500 websites—used by millions—had more responsibility for FP errors than the ad blockers. However, about 34% of the errors impacted "Embedded content" such as the iFrame element. The reason for that was the embedded content used an src attribute that referred to a source provided by a third party. Hence, the designers of the third party bore the responsibility for that error.

Figure 4 shows the sources of errors, clustered into four groups. The first group is `EasyBlockR`, which means the error happened because ad blockers blocked web requests, and `EasyHideE` represents the reason that ad blockers hide elements incorrectly. The group `DesBlockR` is due to the designers using incorrect strings in the URL, and the group `DesHideE` results from the designers using incorrect attributes for elements that were matched by EasyList filters. The `DesHideE` had a lower number of errors among all the classes of websites. The `DesBlockR` had high impacts on the top 500 websites, but `EasyBlockR` took a high responsibility in the rest of the website classes.

The ad blockers and website designers are both cause FP errors in different place and by different methods.

4.5 Analysis of Filters causing FP Errors

Next, we take a closer look at the filters that caused the errors. A filter is added to EasyList by creating a new filter, modifying an existing filter, or transferring from another ad-block list. We extracted the ground truth of the filters from dataset **D2FP**.

Table 6 shows the percentage of the sources of error-causing filters. More than 54% of error-causing filters came from adding new filters directly (31% on home pages, and 23% on internal pages). Then 18% of the sources were from modifying existing filters. Filter lists such as Fanboy [15] were merged with EasyList, and about one in ten error-causing filters was from these filter lists. However, 17% of the error-causing filters were added to the first EasyList version in our dataset. We do not know where the filters added by the Editors to the first version in 2009 came from. Furthermore, 4% errors were caused by maximizing the existing filter scope (*i.e.*, making the filter generic instead of specific). Even though this percentage seems small, generic filters that caused errors on one website may cause errors on many other websites.

Each filter from EasyList uses a signature to match ad elements. We analyzed the signatures used by error-causing filters to match the elements. The signatures depend on HTML attributes to hide the elements or URLs that use src attributes to block HTTP GET requests. We studied each filter to extract the HTML attributes used to build the signatures. We grouped these attributes and named them according to the HTML 5.2 standard [79] of the W3C standard (Section 4). As Table 7 shows, the majority of error-causing filters used global attributes such as ID and Class name. A tiny number of

Attribute Category	%
Global Attribute	52.4%
Tree Order	18.1%
Tree Order + Specific Link	14.8%
Global Attribute + Tree Order	7.1%
Specific Attribute + Specific Link	3.9%
Global Attribute + Specific Link	2.9%
Specific Attribute	0.9%

Table 7: The attributes used by error-causing filters to match HTML elements.

Type of reason	%
Insufficient detail	62.8%
Not ad or the error solved before	15.4%
Report in wrong forum section	8.3%
Adware	8.37%
Incomplete report	2.9%
Software Issue	2.3%

Table 8: The reasons of rejecting FN error reports.

Website	# Reports	Alexa Rk.
youtube.com	201	2
Yahoo.com	98	6
facebook.com	73	3
google.com	45	1
Twitch.tv	41	43
CBS.com	31	1716
thevideo.me	23	656

Table 9: The most website indicated in FN error reports (Alexa Rk means Alexa's ranking).

error-causing filters used specific attributes. For example, for `src` attribute, it may use either a domain name or part of the address. Finally, we investigated the links that were used to block HTTP GET requests. We found that the majority of the error-causing filters (68%) used string (i.e., part of the address) as a signature. Indeed, this part of the address makes a filter more generic, which leads to more errors.

4.6 FN Errors Reported by the Crowd

EasyList depends entirely on crowdsourcing to detect FN errors in the wild. Between November 2009 and December 2018, about 17,968 reports were submitted. Each report was represented as a post with its thread or replies. Figure 5 shows that 29% of the posts did not have a response from EasyList editors. To understand if EasyList editors missed those reports or solved the errors without a public response, we extracted the incomplete reports from dataset D2B. We manually checked the reports. We found that the reports missed crucial details such as the names of the websites that had FN errors, and the editors closed the report since they are not complete. From all the FN reports, 20.6% of FN error reports were correct, and 20.8% of FN error reports were only about incorrect (or rejected) reports. The rest of the reports did not show strong evidences to be classified in either group. Even though the reports may have sufficient details such as the name/URL of the website and the type of errors, the editors did not respond or did not confirm whether the reports were correct or incorrect, or simply because the reports were closed. Certain reports were indeed discussed by the forum members but were closed without editors' approval. In general, such limitations of the public forums make it difficult to track the stats of some FN reports.

We analyzed the incorrect reports and listed the results in Table 8. More than 62% of them were ignored because the reports did not provide sufficient details about the ads and websites, or the reporters did not respond to the editors' questions. Another important observation is that about 8% of the reports were rejected because of ad-ware. Many users thought their ad-blocker failed to block specific ads, but the truth is that their computers were infected by adware that was overwriting the ad-blocker to inject ads. About 97.6% of these users were anonymous or new members.

Our analysis above indicates that lots of crowd efforts were wasted due to incorrect/incomplete reports. The forum web interface should be better designed to ensure all important information be collected.

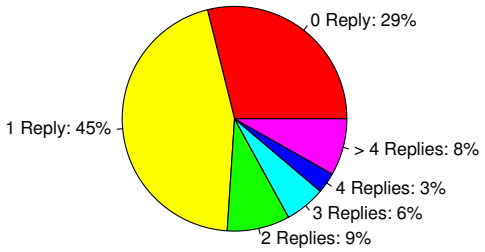

Figure 5: # of Replies to the topics that represent the reports.

4.7 Websites with FN Errors

From dataset D2, 12,866 websites were mentioned to have FN errors. Some of them were indicated in multiple reports. Table 9 illustrates the most mentioned websites in the reports. The common factor of these websites is that they are high-ranked websites. Intuitively, many users visit these websites, which increases the opportunity to report FN errors. However, we need to understand if these websites used countermeasures against ad-block systems, hence the increase in the number of FN errors. Therefore, we extracted the correct reports that mentioned these websites from dataset D2A . Then we inspected the filters that were added to correct these errors. We found that the majority of the filters were added to block third-party requests from reputable ad networks. The implication is that the FN errors in high-rank websites also occurred because of third-party ad networks that were dealt with. We conclude that the most of FN errors happened in High ranked websites are because of the third parity (ad publishers).

5 ADVERSARIAL EVASION ATTACKS

Existing works have discussed some vulnerabilities of EasyList [51, 82, 83] that allow ad publishers or website developers to bypass the ad-blockers to deliver an advertisement to users. While the arms race between ad publishers and the EasyList community is carrying

on, no study has looked into the problem from an empirical and longitudinal prescriptive. In the following, we present a series of vulnerabilities that are exploited (or can be exploited) to execute *evasion attacks*.

Specifically, we analyze 15 attacks and group them in three categories. The first category, called **More-Studied Attacks**, includes four types of attacks that have been investigated by previous research in one way or another. The second category, called **Less-Studied Attacks**, contains three attacks that are not well understood even though they are known. In the third category, We introduce eight **Nonstudied Attacks**, which have not been reported by the research community. Different from the previous research, our vulnerability measurement is based on analysis of the historical behavior of EasyList, and a rich set of data sources including the syntax change of the filters, the change of signatures used to match the ad elements in error reports, the behaviour of ad servers, and the unsolvable reports.

5.1 More-Studied Attacks

1. WebSockets. The WebSocket protocol gives web developers the ability to use client-side JavaScript to establish a connection to an ad server. This connection allows the server to push messages to the client without a request from the client. It is known that ad networks have been using webSockets to circumvent ad blockers [12]. By analyzing our dataset, we show that EasyList had blocked 291 websites and 137 ad servers for using WebSockets since 2016. The ad network Uponit [76] (which is listed in EasyList blacklist) has used WebSocket.

2. Anti-ad Blocker. A large body of existing work has studied anti-ad blockers [30, 51, 89, 90]. Their research goals vary from measuring the prevalence of anti-ad blockers, to designing methods to bypass anti ad-blockers, to studying the reactions of the websites that utilized anti-ad blockers against ad blockers. In particular, Zhu et al. [89] listed three reactions: showing warning messages, switching ads, or reporting ad-block statistics. Our analysis of the dataset D2 reveals a broader range of reactions, including restricting content on the sites using paywalls, blocking the websites, redirecting the users to different websites or content, and blocking legitimate content on the websites.

3. Randomization of Ad Attributes and URLs. A sophisticated method of circumventing ad blocker was proposed by [83]. The idea is that the server can randomize the DOM of the web page construction to hide the ad signatures. In addition, the whole URLs are encrypted using public-key cryptography to randomize the URL strings. To counter this evasion, EasyList introduced two new filter syntaxes: `-abp-has` to filter out an ad element according to its content subtree, and `-abp-contains()` to filter out an ad element according to a specified string that the element contains. EasyList as well as other filter lists take advantage of the new CSS4 *pseudo-class: ':has()'* [80] and jQuery API *:contains() Selector* [34]. For example, the filter `facebook.com##.outer:-abp-has(a:abp-contains(ad))` selects a child node that contains string `ad` of a node that has class `outer` on facebook.com. We analyzed the filters that used this syntax and found 15 websites that performed such randomization. Facebook appeared most frequently in the filter list. Among all other websites, 57% of the filters used this syntax function to block ads on Facebook. We did not find any case that encrypted the URLs to circumvent ad blockers.

Ad Network	Server Domains		Since
	Added	Removed	
PopAds	8,541	38	Sep-2016
Propellerads	910	2	Apr-2016
Yavli	338	6	Oct-2014
Uponit	207	1	Feb-2018
Hilltopads	155	3	Jan-2018
Tag adservers	47	2	Feb-2017
Admiral	47	4	Jan-2017

Table 10: Number of Ad-server domains that are added or removed to/from the blacklist of EasyList. The date indicates the first time when the first domain was added to EasyList.

4. Factoring Acceptable Ads List Sitekeys. Walls at el. [82] present a way to circumvent EasyList and exhibit the ads by cracking the whitelist sitekeys. Sitekeys are filters that have base64 representation of RSA public keys used to identify accepted ads by ad-block software. However, we did not find any report complaining about this issue.

5.2 Less-Studied Attacks

1. Changing Ad-Server Domains. The ad servers are used to store, maintain and serve advertisements to website visitors when the web pages are loaded. Iqbal at el. [30] and Vastel at el. [77] state that ad networks often change ad server domains to avoid being blocked. However, they did not provide large-scale statistical analysis (or evidences). As such, we empirically examine how these ad networks exploited EasyList using our dataset.

Our dataset shows that the number of ad server domains and IPs has increased from 505 in 2009 to 15,500 in January 2019. The average number of domains added to EasyList per month during the last 9 years was 146, while the average number of domains removed was 72. About 20% of the ad server domains listed in the EasyList (as of January 15, 2019) belonged to 7 ad networks. Table 10 shows the number of ad-server domains that were added or removed from EasyList. We observe that the number of added domains is significantly larger than the number of removed domains. After further investigation, we conclude that EasyList did not properly handle the obsolete and redundant domains. We found 104 domains listed in EasyList in May 2017 were duplicated with other domains, and they belonged to Propellerads [59]. The problem was discovered and the filters of such domains were removed in March 2018.

We also looked beyond our dataset to demonstrate the existence of this attack. More specifically, we measured the number of Internet users who visited the ad servers before and after they were added to EasyList. For this analysis, we obtained the historical traffic information of the ad-severs from Alexa Web Information Service (AWIS) [5]. AWIS offers traffic ranks of the websites, based on a combined measure of unique visitors and page views on a daily basis. Note that the AWIS API only provides the traffic history of a given server for the last 4 years, which is sufficient for our analysis.

Using the 4-years of AWIS dataset, We analyzed 567,293 records of 6,903 ad-server domains. We found that 52% of the ad servers' traffic activities disappeared *three days* after these ad-server domains were added to EasyList. This was because EasyList was set to be expired (updated) every four days. Lately, the versions of EasyList used by Adblock Plus (and some other lists such as EasyPrivacy Tracking Protection List) has changed the update frequency to be one day [54, 56, 57]. We further observed that 84% of these 52% ad servers were blocked shortly after they were created. Shortly after their traffic activities started, their domains were added to the EasyList in the same day. The reason was that certain ad networks randomly generated ad server domains. The EasyList community (including editors and Adblock Plus developers) ran code to monitor the changes of certain websites to detect the change of ad servers [20, 22]. Those randomly generated domains thus were quickly detected and added to EasyList for blocking.

The next question is whether the EasyList's blocking significantly reduces the ad-network domains' traffic in a longer term. We applied t-test to find if blocking by EasyList had a significant impact on the ad servers. To hold the normality assumption from both groups, we used the traffic activities of ad servers for 30 records before and 30 records after the blocking. The significance level $\alpha = 0.05$. We tested the differences in the records' average means of 1,400 ad-server domains and found only 61% of the ad servers were significantly influenced by the blocking.

2. Changing Ad-Element Attributes. Website developers can circumvent ad blockers by changing the ad element attributes [30, 71]. We studied the reaction of EasyList against this evasion using our dataset. We found that EasyList did not have the capability to automatically trace the change of ad elements. The changed ad elements were given new signatures and were treated as new ones. These new elements were blocked only if they were reported by crowdsourcing. Occasionally, EasyList could recognize the change of an ad attribute when a popular ad network changes it for all its websites. In this case, EasyList editors would replace an old filter with a new one.

To analyze this behavior, we extracted the corresponding filters that have common signatures from dataset D1. More specifically, we look for cases where the old signature(s) were removed and new signature(s) were added to the same EasyList version. If a new signature was applied to the same website, we conclude that EasyList was chasing after the website to detect the modification of an ad-element attribute. We only found 553 instances that EasyList changed the filters in response to this type of evasions. The average delay of changing the attributes was 10.3 days. From these instances, we found 311 websites changing their attributes. About 88% of them were the clients of the ad network Yavli (`yavli.com`).

3. Changing the Path of Ad Source. EasyList filters match a request using the domain name or the URLs path. Therefore, the ad publishers may change the "path" of the URLs to circumvent ad blockers. Like the previous analysis, we identified the signatures added and removed form each EasyList version that were applied on the *same websites*. In total, we found 644 websites changed their the ad URL's paths to circumvent blocking. About 47.6% of them were clients of the ad network Yavli.

5.3 Nonstudied Attacks

From our dataset, we have identified 8 types of evasion attacks that are not yet studied by the research community. We further investigated the error reports of EasyList, the public reports of other active filter lists such as uAssets filter list [73], and the technical issue reports of Adblock Plus [55] and uBlockOrigin [74]. We conclude that although some evasion attacks are known by certain ad blockers, they are not known by all ad blockers. The following are our discoveries on these nonstudied attacks.

1. Exploiting Obsolete Whitelist Filters. EasyList did not track the domains in the whitelist filters. The whitelist includes the exception filters indicating the elements and URLs on which ad blockers do not take effect. EasyList editors manually maintained and removed the obsolete filters over time (every $2-3$ months). Over the past 9 years, EasyList editors handled dead filters 82 times and removed 353 domains from the whitelist. Some of the whitelisted domains have been abused to deliver ads. For instance, `shackvideo.com` is a domain name added to whitelist filters in November 2010. Using Hosterstats [27], we found that the domain was deleted in May 2016. After that, a website exploited it to deliver ads. EasyList discovered this issue in January 2017 and deleted the filter.

2. Using Generic Exception Rules. When a user visits a website with anti-ad blocker(s), the anti-ad blocker script tests the states of ad elements to detect the presence of ad blockers. To counter this practice, EasyList would block the anti-ad blocker script or exclude the corresponding ad elements by putting them in the whitelist. To bypass ad blocking, some ad networks and developers introduced elements that could trigger anti-ad blocking and distributed them across many websites. With this, it was arduous for EasyList to discover all the websites that were using these elements; therefore, EasyList would add a generic exception rule for those elements. We have observed many websites abused generic exception filters to bypass EasyList. For instance, `jpost.com` exploited the exception filters,

```
@@||redtube.com*/adframe.js
```

to bypass EasyList and deliver ads using the following URL:

```
http://redtube.com.umamdmo.com/vp?&i=110
...=true&cb=OYmZJ&dr=true&q=/adframe.js
```

The exception rule was created in January 2014 and the abuse was discovered in November 2016.

3. Exploiting False Positive Errors. Some websites applied *self-defacement* to create difficulties for users that use ad blockers. We observed that some websites associated and linked the ad elements with their legitimate elements. When ads were blocked, the non-ad elements were blocked too, causing false positives for the ad-blocker. They linked the ad elements with legitimate elements using the signatures to match the EasyList filters with their ad elements.

4. First-Party Content and Inline Script. Despite the security risk of using the inline scripts [32, 36] , websites have been using them to detect ad blockers in order to perform anti-ad blocker

tasks [89]. Accordingly, EasyList made exception rules for ad elements that were used as inline script triggers. However, this behavior has tempted the web developers to increase the utilization of inline scripts and the first-party content. The web developers have utilized inline scripts to create network requests to deliver the ads directly or using the first-party content to host the ad URLs. EasyList started to take advantage of the Content Security Policies (CSP) [50] to block inline scripts. The first version of EasyList containing a CSP-based filter appeared in May 2018. Within six months, EasyList added 142 websites to the blacklist, 78% of which were in the top 100K Alexa rank.

5. ISP Injecting Ads. To deliver ads to users, Internet Service Providers (ISPs) may inject HTML, CSS, or/and Javascript into HTTP responses. This strategy was limited to the websites that use HTTP. Existing works have discussed this method of delivering ads [10, 52, 72]. However, this method may also be exploited to circumvent filter lists. The traditional method of blocking the ad server domain is no longer effective in this case, because there is no HTTP request to be blocked and the ads are appended to the first-party content. Moreover, an ISP is able to append an ad to a different URL each time. Among all the FN reports, we found only one report that indicated an FN error due to ISP ads injection, performed by the ISP named Optimum [6]. A countermeasure adopted by EasyList was to block the IP addresses that served the ads, which is not a fundamental solution.

6. Background Redirection. Ad publishers may circumvent browser's pop-up ad blockers by using a technique known as *Tab-unders*. Specifically, when users click on a link in a web page, the link is opened in a new tab and the background of the old tab is redirected to an advertisement page. Based on our analysis of the crowdsourcing reports, we found that this problem was reported in May 2015. However, until today, EasyList is still unable to address this issue. Recently, Chrome (version 68.0 and above) has blocked Tab-under navigation [75], but other browsers such as Firefox have not yet taken actions.

7. Exploiting WebRTC. WebRTC, using Real-Time Communications (RTC), is an open-source project providing APIs to enable browsers and mobile applications to communicate without requiring an intermediary. Security concerns of using WebRTC [85] have been reported recently [61, 62]. However, ad networks have utilized WebRTC to establish real-time communications between servers and browsers and used RTCPeerConnection APIs to send ads. In May 2018, EasyList and Adblock Pulse started to block ads that use WebRTC by wrapping the RTCPeerConnection. EasyList added to its blacklist 220 website domains and 139 ad-server domains that used WebRTC. Uponit ad servers used 136 domains based on WebRTC.

8. CSS Background Image Hack. Some websites used CSS background image hacks to exhibit the ads. EasyList created special filters used by Adblock Plus and uBlock Origin with a new syntax to countermeasure the attack. The syntax is `[-abp-properties='data:']`, which allows ad-blocking software to recognize the ad elements according to their properties. We observed that EasyList applied the new syntax on 40 websites in 2016 and 2017.

6 DISCUSSIONS

Key Implications. Our analysis shows that the number of FP errors is non-trivial. The website owners who used unsuitable element attributes and EasyList that created bad signatures were both responsible for the FP errors. However, it seems to be impossible to establish the communication between the two parties to avoid and mitigate such errors because of the "enmity" between them. This increases the responsibility of the researchers to build more effective systems to detect and resolve such errors. In this work, we aim to build a foundation for future research by analyzing the sources of the errors from different aspects.

Ad block Systems in General. Ad block systems are considered as a security-enhancing measure. With the quick expanding in employing these systems, many security tasks were assigned to them by introducing new filter lists to provide protections against malvertising, phishing, spamming, crypto-mining, clickbaiting, and others. We depended heavily on EasyList in our analysis for two reasons. First, as we have indicated, it is the most used filter list. That created a lot of adversaries against it, and expand the scope of our research to find all the possible attacks and vulnerabilities. Second, many ad-blocking software use it as a default filter list, and many filter lists directly inherit from it or follow its way to filter ads.

Consequences of Filter Lists' Limitations. We highlighted the errors and the misunderstandings, and we provided intensive analysis of 15 attacks on the core of the systems – filter lists. One of the main goals is to contribute a lighthouse for research and industry communities about systems used by hundreds of millions. We notice there is a considerable amount of research depends on EasyList for their evaluation measurement [13, 31, 64, 87, 90]. The lack of understanding the limitations of EasyList may lead to inaccurate results. Recent works start to address the issues of filter lists. For example, Zhu at el. [90] also highlighted the issue of "ad rotation networks" and "automatically generated domain names", and proposed solutions. We show that EasyList community is also adding new syntaxes to overcome these attacks, and researchers should cross-compare the results to provide new gains. Another recent work [31] proposed a machine learning approach to block ads automatically which can block 16% more ads than the filter lists. Systems as such help to address the false negatives of the filter list but may also introduce new false positives. Our study has shown that more than one-third of the community effort was to correct FP errors, which is a factor that cannot be overlooked.

Advanced Attacks Against Adblockers. We presented the 15 possible countermeasures against ad blockers as the basic evasion attacks. There are some advanced countermeasures. For instance, Carasso in a Google patent[16] illustrated a method of delivering ads using a bypass loader and a bypass proxy. The loader checks the presence of ad blockers, and then the ads are delivered using a bypass proxy that changes its domain frequently. Indeed, the method is combined of two types of evasion attacks: changing the Ad-Server Domains and Anti-ad Blocker.

Limitations. First, our dataset covered the historical data back to 2009. We could not find any data before November 2009, or infer

when the rules were added in the first snapshot. Second, when extracting the ground-truth error reports, we chose a conservative approach to link the reported errors to the EasyList updates. This was to ensure the correctness of the linking and the reliability of the results, but we had to sacrifice the coverage. It is a trade-off between the scale of the data and the accuracy of the analysis. We chose the latter. Although the scale is smaller, the samples are still very diverse, covering different reporter types, editors, websites types, and website ranking ranges. Finally, the Internet Archive limited us from extracting data from web pages that were not archived. A similar limitation is indicated in [41].

Other filter lists. Our work focused on the ad-blocking filter list and its weakness. Other types of filter lists share with ad-blocking filter list some of the weaknesses. For example, Sinha et. al. [69] illustrated that the reputation-based blacklist, namely those used to block unsolicited emails, have considerable false positive errors. Another example is the phishing blacklists. Sheng et. al. [67] stated that phishing detection by heuristics took a long time to appear on blacklists. However, studying the lifetime of false positives in other filter lists is a plan for our future work.

7 RELATED WORK

In this section, we briefly discussed the related works in the following two categories: issues related to crowdsourcing, and analyzing the ad-blocking ecosystem.

Crowdsourcing Accuracy and Security Issues. Recent research about crowdsourcing quality is mainly about the systems in the following fields: information retrieval [35], financial incentives [43], labeling [42, 68], natural language [9, 70] and geographic information [23]. There are also many works which attempt to enhance the accuracy and quality of crowdsourcing [4, 8, 37–39, 42]. For example, Le at al. [39] suggested training the crowd.

Analysis of Ad-blocking Systems and Filter Lists. The relationship between Internet users, ad publishers, and ad blockers has been studied by substantial research from different angles. One angle is the reaction between the ad blockers and the websites [51, 53, 89]. For instance, Zhu at el. [89] presented a differential execution analysis method to detect and analyze anti-ad blockers. Another angle was studied by Pujol et al. [60], who analyzed the interactions between Internet users who use ad-blocking software and the ad ecosystem. Vratonjic et al. [78] have investigated the consequences of ad blocking on the websites from the perspective of a business model, while Miroglio et al. [48] have investigated the effects of ad blocking on Internet users.

As to Filter Lists, Wills and Uzunoglu [87] have studied the third-party domains of filter lists and suggested ways to improve ad-blocking tools to prevent requests to a different type of these domains. The whitelist of EasyList were studied by Walls et al. [82]. The privacy filter lists have been investigated on a high level by Gervais et al. [25] and [46] in order to study tracker-blocking methodologies. Our approach is complementary to existing works: We conducted a deeper analysis of the accuracy of filter lists.

8 CONCLUSION

This paper presented an intensive study of the behavior of the ad-blocking system and its community using two longitudinal datasets. The central conclusions are (1) false positive in ad-blocking systems using filter list is non-trivial, (2) a large number of false positive errors have long lifetimes that occurs in a wide range of websites as a consequence of many reasons, (3) the mechanism of dealing with false negative errors has some deficiencies and weaknesses, (4) the system has many vulnerabilities impacting its accuracy. The paper presented the vulnerabilities by exhibiting 15 different ways to circumvent the system. We hope the findings build lighthouses for any future work to evolve ad blocking and optimize crowdsourcing mechanisms.

ACKNOWLEDGEMENT

We would like to thank our shepherd Georgios Smaragdakis and the anonymous reviewers for their helpful feedback. This project was in part supported by NSF grants CNS-1750101, CNS-1717028, CNS-1618684, CNS-1718459. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

REFERENCES

[1] 2017. About /robots.txt. http://www.robotstxt.org/robotstxt.html
[2] AdGuard. 2019. AdGuard Ad Filters. https://kb.adguard.com/en/general/adguard-ad-filters
[3] Aexa. 2016. Drive More Website Traffic with Competitive Analysis. https://www.alexa.com/siteinfo
[4] Ahmet Aker, Mahmoud El-Haj, M-Dyaa Albakour, Udo Kruschwitz, et al. 2012. Assessing Crowdsourcing Quality through Objective Tasks.. In *Proceedings of the 2012 International Conference on Language Resources and Evaluation (LREC)*.
[5] Alexa. 2019. Alexa Web Information Service. https://awis.alexa.com/
[6] Altice. 2019. Optimum by Altice. https://www.optimum.com/alticeone
[7] Internet Archive. 2019. Wayback Machine. https://archive.org/web/
[8] Pavel Atanasov, Phillip Rescober, Eric Stone, Samuel A Swift, Emile Servan-Schreiber, Philip Tetlock, Lyle Ungar, and Barbara Mellers. 2016. Distilling the wisdom of crowds: Prediction markets vs. prediction polls. *Management science* (2016).
[9] Kartik Audhkhasi, Panayiotis G Georgiou, and Shrikanth S Narayanan. 2012. Analyzing quality of crowd-sourced speech transcriptions of noisy audio for acoustic model adaptation. In *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.
[10] Pradeep Bangera, Syed Hasan, and Sergey Gorinsky. 2017. An advertising revenue model for access ISPs. In *2017 IEEE Symposium on Computers and Communications (ISCC)*. 582–589.
[11] David Barton. 2017. The Rules of Adblocking: How block list work. http://pagefair.com/blog/2016/behind-the-scenes-adblocking-lists-and-countermeasures.
[12] Muhammad Ahmad Bashir, Sajjad Arshad, Engin Kirda, William Robertson, and Christo Wilson. 2018. How tracking companies circumvented ad blockers using websockets. In *Proceedings of the 2018 ACM Conference on Internet Measurement Conference (IMC)*.
[13] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. 2014. Leveraging machine learning to improve unwanted resource filtering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop (AISec)*. 95–102.
[14] Chromium blog. 2018. Under the hood: How Chrome's ad filtering works. blog.chromium.org/2018/02/how-chromes-ad-filtering-works.html
[15] Ryan Brown. 2019. Fanboy Filter Lists. https://www.fanboy.co.nz/
[16] Adam Carasso. 2015. Systems and methods to bypass online advertisement blockers. US Patent 9,177,335.
[17] Google Chrome. 2019. Chrome Web Store-Extensions. https://chrome.google.com/webstore/category/extensions
[18] Wikipedia contributors. 2019. HTTP/ 301. https://en.wikipedia.org/wiki/HTTP/_301 [Online; accessed 13-May-2019].
[19] Matthew Cortland. 2017. PageFair adblock report: The state of the blocked web presents a combined picture of desktop and mobile adblock usage for the first time. https://pagefair.com/blog/2017/adblockreport/

[20] EasyList. 2019. Automated Adserver Update. https://github.com/easylist/easylist/commit/f31a2d3800547615b82cb89333586828749a7e88
[21] EasyList. 2019. Overview of EasyList. https://easylist.to
[22] Fanboy. 2019. Create a tool/script to pickup revolving adservers. https://issues.adblockplus.org/ticket/5323
[23] Giles M Foody, L See, Steffen Fritz, Marijn Van der Velde, Christoph Perger, Christian Schill, and Doreen S Boyd. 2013. Assessing the accuracy of volunteered geographic information arising from multiple contributors to an internet based collaborative project. *Transactions in GIS* (2013).
[24] EasyList Forum. 2019. EasyList Forum:The Easy Subscriptions for Adblock, Adblock Plus and uBlock Origin. https://forums.lanik.us/
[25] Arthur Gervais, Alexandros Filios, Vincent Lenders, and Srdjan Capkun. 2017. Quantifying web adblocker privacy. In *Proceedings of the 2017 European Symposium on Research in Computer Security (ESORICS)*.
[26] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. 2015. An automated approach for complementing ad blockers' blacklists. In *Proceedings of the 2015 Privacy Enhancing Technologies Symposium (PETS)*.
[27] HosterStats. 2019. About HosterStats.com. http://www.hosterstats.com/AboutHosterStats.php
[28] Dan Hubbard. 2016. Cisco Umbrella 1 Million. https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/
[29] Gargoyle Software Inc. 2018. HtmlUnit. http://htmlunit.sourceforge.net
[30] Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. 2017. The ad wars: retrospective measurement and analysis of anti-adblock filter lists. In *Proceedings of the 2017 Internet Measurement Conference (IMC)*.
[31] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. 2018. Adgraph: A machine learning approach to automatic and effective adblocking. *arXiv preprint arXiv:1805.09155* (2018).
[32] Martin Johns. 2014. Script-templates for the content security policy. *Journal of Information Security and Applications (JISA)* (2014).
[33] Dixon Jones. 2012. Majestic Million CSV now free for all, daily. https://blog.majestic.com/development/majestic-million-csv-daily/
[34] jQuery. 2019. :contains() Selector. https://api.jquery.com/contains-selector/
[35] Gabriella Kazai, Jaap Kamps, Marijn Koolen, and Natasa Milic-Frayling. 2011. Crowdsourcing for book search evaluation: impact of hit design on comparative system ranking. In *Proceedings of the 2011 ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR)*.
[36] Christoph Kerschbaumer, Sid Stamm, , and Stefan Brunthaler. 2016. Injecting CSP for Fun and Security. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)*.
[37] Aniket Kittur and Robert E Kraut. 2008. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work (CSCW)*.
[38] Aniket Kittur, Jeffrey V Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. 2013. The future of crowd work. In *Proceedings of the 2013 ACM Conference on Computer Supported Cooperative Work (CSCW)*.
[39] John Le, Andy Edmonds, Vaughn Hester, and Lukas Biewald. 2010. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *Proceedings of the 2010 ACM SIGIR Workshop on Crowdsourcing for Search Evaluation (CSE)*.
[40] Ada Lerner, Tadayoshi Kohno, and Franziska Roesner. 2017. Rewriting history: Changing the archived web from the present. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[41] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. 2016. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *Proceedings of the 2016 USENIX Security Symposium (USENIX Security)*.
[42] Hongwei Li, Bin Yu, and Dengyong Zhou. 2013. Error rate analysis of labeling by crowdsourcing. In *Proceedings of the 2013 International Conference on Machine Learning Workshop: Machine Learning Meets Crowdsourcing (ICML Workshop)*.
[43] Winter Mason and Duncan J Watts. 2009. Financial incentives and the performance of crowds. In *Proceedings of the 2009 ACM SIGKDD Workshop on Human Computation (KDD Workshop)*.
[44] Mercurial. 2019. Mercurial: Changeset. https://www.mercurial-scm.org/wiki/ChangeSet
[45] Mercurial. 2019. Mercurial source control management: Adblock Plus. https://hg.adblockplus.org/
[46] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. 2017. Block me if you can: A large-scale study of tracker-blocking tools. In *Proceedings of the 2017 IEEE European Symposium on Security and Privacy (EuroS&P)*.
[47] Michael. 2011. EasyList Statistics: August 2011. https://easylist.to/2011/09/01/easylist-statistics:-august-2011.html
[48] Ben Miroglio, David Zeber, Jofish Kaye, and Rebecca Weiss. 2018. The Effect of Ad Blocking on User Engagement with the Web. In *Proceedings of the 2018 World Wide Web Conference (WWW)*.
[49] Mozilla. 2018. Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/
[50] Mozilla. 2019. Content Security Policy (CSP). https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
[51] Muhammad Haris Mughees, Zhiyun Qian, and Zubair Shafiq. 2017. Detecting anti ad-blockers in the wild. In *Proceedings of the 2017 Privacy Enhancing Technologies Symposium (PETS)*.
[52] Gabi Nakibly, Jaime Schcolnik, and Yossi Rubin. 2016. Website-targeted false content injection by network operators. In *Proceedings of the 2016 USENIX Security Symposium (USENIX Security)*.
[53] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E Powles, ED Cristofaro, Hamed Haddadi, and Steven J Murdoch. 2016. Adblocking and counter blocking: A slice of the arms race. In *Proceedings of the 2016 USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.
[54] Adblock Plus. 2017. Source: synchronizer.js. https://adblockplus.org/jsdoc/adblockpluscore/synchronizer.js.html
[55] Adblock plus. 2019. Adblock plus Reports. https://issues.adblockplus.org/report
[56] Adblock Plus. 2019. EasyList of Adblock Plus. https://easylist-downloads.adblockplus.org/easylist.txt
[57] Adblock Plus. 2019. EasyPrivacy of Adblock Plus. https://easylist-downloads.adblockplus.org/easyprivacy.tpl
[58] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2018. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156* (2018).
[59] PropellerAds. 2019. PropellerAds: AdTech that helps affiliates succeed. https://propellerads.com/about-us/
[60] Enric Pujol, Oliver Hohlfeld, and Anja Feldmann. 2015. Annoyed users: Ads and ad-block usage in the wild. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference (IMC)*.
[61] PureVPN. 2018. How to Disable WebRTC in Firefox and Chrome Browsers. https://www.purevpn.com/blog/disable-webrtc-in-chrome-and-firefox/
[62] PUREVPN. 2018. WebRTC Leaks Vulnerability-SOLVED (For all Browsers). https://restoreprivacy.com/webrtc-leaks/
[63] Quantcast. 2018. Top Websites. https://www.quantcast.com/top-sites/US/1
[64] Kent Rasmussen, Alex Wilson, and Abram Hindle. 2014. Green mining: energy consumption of advertisement blocking methods. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS)*. 38–45.
[65] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D Strowes, and Narseo Vallina-Rodriguez. 2018. A long way to the top: significance, structure, and stability of internet top lists. In *Proceedings of the 2018 Internet Measurement Conference (IMC)*.
[66] Wayback CDX Server. 2018. Wayback CDX Server API. https://github.com/internetarchive/wayback/tree/master/wayback-cdx-server
[67] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Faith Cranor, Jason Hong, and Chengshan Zhang. 2009. An empirical analysis of phishing blacklists. In *Proceedings of the Sixth Conference on Email and Anti-Spam (CEAS)*.
[68] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 2008 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.
[69] Sushant Sinha, Michael Bailey, and Farnam Jahanian. 2008. Shades of Grey: On the effectiveness of reputation-based âĂIJblacklistsâĂİ. In *Proceedings of the 2008 International Conference on Malicious and Unwanted Software (MALWARE)*. 57–64.
[70] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng. 2008. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[71] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. 2017. The future of ad blocking: An analytical framework and new techniques. *arXiv preprint arXiv:1705.08568* (2017).
[72] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, et al. 2015. Ad injection at scale: Assessing deceptive advertisement modifications. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.
[73] uAssets. 2019. uAssets: resources for uBlock Origin (uBO) uMatrix: static filter lists ready-to-use rulesets etc. https://github.com/uBlockOrigin/uAssets
[74] uBlock. 2019. uBlock-issues. https://github.com/uBlockOrigin/uBlock-issues
[75] uBlockOrigin. 2019. Adblock Plus filters explained. https://www.chromestatus.com/feature/5675755719622656
[76] upOnit. 2019. UPONIT:Who We Are. https://uponit.com/who-we-are/
[77] Antoine Vastel, Peter Snyder, and Benjamin Livshits. 2018. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *arXiv preprint arXiv:1810.09160* (2018).
[78] Nevena Vratonjic, Mohammad Hossein Manshaei, Jens Grossklags, and Jean-Pierre Hubaux. 2013. Ad-blocking games: Monetizing online content under the threat of ad avoidance. In *The Economics of Information Security and Privacy*.
[79] W3C. 2018. HTML 5.2:W3C Recommendation. https://www.w3.org/TR/2017/REC-html52-20171214/

[80] W3C. 2019. The Relational Pseudo-class: ':has()'. https://drafts.csswg.org/selectors-4/#has-pseudo
[81] W3C. 2019. The World Wide Web Consortium (W3C). https://www.w3.org/
[82] Robert J Walls, Eric D Kilmer, Nathaniel Lageman, and Patrick D McDaniel. 2015. Measuring the impact and perception of acceptable advertisements. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference (IMC)*.
[83] Weihang Wang, Yunhui Zheng, Xinyu Xing, Yonghwi Kwon, Xiangyu Zhang, and Patrick Eugster. 2016. Webranz: web page randomization for better advertisement delivery and web-bot prevention. In *Proceedings of the 2016 ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*.
[84] Webrecorder. 2019. Webrecorder. https://webrecorder.io
[85] WebRTC. 2019. Optimum by Altice. https://webrtc.org/faq/#what-is-webrtc
[86] Williams. 2018. Adblock Plus and (a little) more. https://adblockplus.org/blog/100-million-users-100-million-thank-yous
[87] Craig E Wills and Doruk C Uzunoglu. 2016. What ad blockers are (and are not) doing. In *Proceedings of the 2016 IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*.
[88] Savvas Zannettou, Jeremy Blackburn, Emiliano De Cristofaro, Michael Sirivianos, and Gianluca Stringhini. 2018. Understanding web archiving services and their (mis) use on social media. In *Proceedings of the twelfth International AAAI Conference on Web and Social Media(ICWSM)*.
[89] Shitong Zhu, Xunchao Hu, Zhiyun Qian, Zubair Shafiq, and Heng Yin. 2017. Measuring and Disrupting Anti-Adblockers Using Differential Execution Analysis. In *Proceedings of the 2017 Network and Distributed System Security Symposium(NDSS)*.
[90] Shitong Zhu, Umar Iqbal, Zhongjie Wang, Zhiyun Qian, Zubair Shafiq, and Weiteng Chen. 2019. ShadowBlock: A Lightweight and Stealthy Adblocking Browser. In *Proceedings of the 2019 World Wide Web Conference (WWW)*.