Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking

PETER SNYDER, Brave Software, USA
ANTOINE VASTEL, University of Lille / INRIA, France
BENJAMIN LIVSHITS, Brave Software / Imperial College London, United Kingdom

Ad and tracking blocking extensions are popular tools for improving web performance, privacy and aesthetics. Content blocking extensions generally rely on filter lists to decide whether a web request is associated with tracking or advertising, and so should be blocked. Millions of web users rely on filter lists to protect their privacy and improve their browsing experience.

Despite their importance, the growth and health of filter lists are poorly understood. Filter lists are maintained by a small number of contributors who use undocumented heuristics and intuitions to determine what rules should be included. Lists quickly accumulate rules, and rules are rarely removed. As a result, users' browsing experiences are degraded as the number of stale, dead or otherwise not useful rules increasingly dwarf the number of useful rules, with no attenuating benefit. An accumulation of "dead weight" rules also makes it difficult to apply filter lists on resource-limited mobile devices.

This paper improves the understanding of crowdsourced filter lists by studying EasyList, the most popular filter list. We measure how EasyList affects web browsing by applying EasyList to a sample of 10,000 websites. We find that 90.16% of the resource blocking rules in EasyList provide no benefit to users in common browsing scenarios. We use our measurements of rule application rates to taxonomies ways advertisers evade EasyList rules. Finally, we propose optimizations for popular ad-blocking tools that (i) allow EasyList to be applied on performance constrained mobile devices and (ii) improve desktop performance by 62.5%, while preserving over 99% of blocking coverage. We expect these optimizations to be most useful for users in non-English locals, who rely on supplemental filter lists for effective blocking and protections.

CCS Concepts: • **Security and privacy** → *Browser security*; *Mobile and wireless security*; Social network security and privacy.

ACM Reference Format:
Peter Snyder, Antoine Vastel, and Benjamin Livshits. 2020. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2, Article 26 (June 2020), 24 pages. https://doi.org/10.1145/3392144

1 INTRODUCTION

As the web has become more popular as a platform for information and application delivery, users have looked for ways to improve the privacy and performance of their browsing. Such efforts include popup blockers, `hosts.txt` files that blackhole suspect domains, and privacy-preserving proxies (like Privoxy [1]) that filter unwanted content. Currently, the most popular filtering tools are ad-blocking browser extensions, which determine whether to fetch a web resource based on its

[1] http://www.privoxy.org/

Authors' addresses: Peter Snyder, Brave Software, USA, pes@brave.com; Antoine Vastel, University of Lille / INRIA, France, antoine.vastel@univ-lille.fr; Benjamin Livshits, Brave Software / Imperial College London, United Kingdom, ben@brave.com.

2476-1249/2020/6-ART26
https://doi.org/10.1145/3392144

URL. The most popular ad-blocking extensions are Adblock Plus [2], uBlock Origin [3] and Ghostery [4], all of which use filter lists to block unwanted web resources.

Filter lists play a large and growing role in making the web pleasant and useful. Studies have estimated that filter lists save users between 13 and 34% of network data, decreasing the time and resources needed to load websites [3, 17]. Others, such as Merzdovnik et al. [15] and Gervais et al. [4], have shown that filter lists are important for protecting users' privacy and security online. Users rely on these tools to protect themselves from coin mining attacks, drive-by-downloads [11, 24] and click-jacking attacks, among many others.

Though filter lists are important to the modern web, their construction is largely ad hoc and unstructured. The most popular filter lists—EasyList, EasyPrivacy, and Fanboy's Annoyance List—are maintained by either a small number of privacy activists, or crowdsourced over a large number of the same. The success of these lists is clear and demonstrated by their popularity. Intuitively, more contributors adding more filter rules to these lists provide better coverage to users.

However, the dramatic growth of filter lists carries a downside in the form of requiring ever greater resources for enforcement. Currently, the size and trajectory of this cost are not well understood. We find that new rules are added to popular filter lists 1.7 times more often than old rules are removed. This suggests the possibility that lists accumulate "dead" rules over time, either as advertisers adapt to avoid being blocked, or site popularity shifts and new sites come to users' attention. As a result, the cost of enforcing such lists grows over time, while the usefulness of the lists may be constant or negatively trending. Understanding the trajectories of both the costs and benefits of these crowdsourced lists is therefore important to maintain their usefulness to web privacy, security and efficiency.

This work improves the understanding of the efficiency and trajectory of crowdsourced filter lists through an empirical, longitudinal study. Our methodology allows us to identify which rules are useful, and which are "dead weight" in common browsing scenarios. We also demonstrate two practical applications of these findings: first in optimally shrinking filter lists so that they can be deployed on resource-constrained mobile devices, and second, with a novel method for applying filter list rules on desktops, which performs 62.5% faster than the current, most popular filtering tool, while providing nearly identical protections to users.

1.1 Research questions

For two months, we applied every day an up-to-date version of EasyList to 10,000 websites, comprising both the 5K most popular sites on the web and a sampling of the less-popular tail. We aimed to answer the following research questions:

(1) What is the growth rate of EasyList, measured by the number of rules?
(2) What is the change in the number of active rules (and rule "matches") in EasyList?
(3) Does the utility of a rule decrease over time?
(4) What proportion of rules are useful in common browsing scenarios?
(5) Do websites try to stealthily bypass new ad-blocking rules, and if so, how?
(6) What is the performance cost of "stale" filter rules to users of popular ad-blocking tools?

1.2 Contributions

In answering these questions, we make the following primary contributions:

[2] https://adblockplus.org/
[3] https://github.com/gorhill/uBlock
[4] https://www.ghostery.com

(1) **EasyList over time:** we present a 9-year historical analysis of EasyList to understand the lifetime, insertion and deletion patterns of new rules in the list.
(2) **EasyList applied to the web:** we present an analysis of the usefulness of each rule in EasyList by applying EasyList to 10,000 websites every day for over two months.
(3) **Advertiser reactions**: we document how frequently advertisers change URLs to evade EasyList rules in our dataset and provide a taxonomy of evasion strategies.
(4) **Faster blocking strategies**: we propose optimizations, based on the above findings, to make applying EasyList performant on mobile devices, and 62.5% faster on desktop environments, while maintaining over 99% of coverage.

1.3 Paper organization

The remainder of this paper is organized as follows. Sections 2 and 7 provides a brief background about tracking and ad-blockers as well as a discussion of the related work. Section 3 presents a 9-year analysis of EasyList's evolution. Section 4 presents how EasyList rules are applied on websites. Section 5 studies how our findings can improve ad-blocking applications on iOS and proposes two new blocking strategies to process requests faster. Finally, in Section 6 we present the limitations, and we conclude the paper in Section 8.

2 BACKGROUND

2.1 Online tracking

Tracking is the act of third parties viewing, or being able to learn about, a users' first-party interactions. Prior work [2, 10, 23] has shown that the number of third-party resources included in typical websites has been increasing for a long time. Websites include these resources for many reasons, including monetizing their website with advertising scripts, analyzing the behavior of their users using analytics services such as Google analytics or Hotjar, and increasing their audience with social widgets such as the Facebook share button or Twitter retweet button.

While third-party resources may benefit the site operator, they often work against the interest of web users. Third-party resources can harm users' online privacy, both accidentally and intentionally. This is particularly true regarding tracking scripts. Advertisers use such tracking tools as part of behavioral advertising strategies to collect as much information as possible about the kinds of pages users visit, user locations, and other highly identifying characteristics.

2.2 Defenses against tracking

Web users, privacy activists, and researchers have responded to tracking and advertising concerns by developing ad and tracker blocking tools. Most popularly these take the form of browser extensions, such as Ghostery or Privacy badger.[5] These tools share the goal of blocking web resources that are not useful to users, but differ in the type of resources they target. Some block advertising, others block trackers, malware or phishing. These tools are popular and growing in adoption [14]. A report by Mozilla stated that in September 2018, four out of the 10 most popular browser extensions on Firefox were either ad-blockers or tracker blockers. Adblock Plus, the most popular of all browser extension, was used by 9% of all Firefox users [6].

Ad and tracking blockers operate at different parts of the web stack.

- DNS blocking relies on a `hosts` files containing addresses of domains or sub-domains to block, or similar information from DNS. This approach can block requests with domain or

[5] https://www.eff.org/fr/node/99095
[6] https://data.firefox.com/dashboard/usage-behavior

sub-domain granularity, but cannot block specific URLs. Examples of domain-blocking tools include Peter Lowe's list [7], MVPS hosts [8] or the Pi-Hole project [9].
- Privacy proxies protect users by standing between the client and the rest of the internet, and filtering out undesirable content before it reaches the client. Privoxy is a popular example of an intercepting proxy.
- Web browsers can attempt to prevent tracking, either through browser extensions or as part of the browser directly. These tools examine network requests and page renderings, and use a variety of strategies to identify unwanted resources. Some tools, such as Privacy Badger use a learning-based approach, while most others use filter lists like EasyList [10] or EasyPrivacy [11].

2.3 EasyList

EasyList, the most popular filter list for blocking advertising, was created in 2005 by Rick Petnel. It has been maintained on Github [12] since November 2009. EasyList is primarily used in ad-blocking extensions such as Adblock Plus, uBlock origin and Adblock, and has been integrated into privacy oriented web browsers[13]. Tools also exist to convert EasyList formats to other privacy tools, like Privoxy [14],

EasyList consists of tens-of-thousands of rules describing web resources that should be blocked or hidden during display. The format also includes syntax for describing exceptions to more general rules. EasyList uses a syntax similar to regular expressions, allowing authors to generalize on patterns used in URLs.

EasyList provides two categories of benefit to users. First, EasyList describes URLs that should be blocked, or never fetched, in the browser. Blocking resources at the network layer provides both performance benefits (e.g. reduced network and processing costs) and privacy improvements (e.g. reduction in number of parties communicated with or removal of fingerprinting script code). Second, EasyList describes page elements that should be hidden at rendering time. These rules are useful when blocking at the network layer is not possible.

Element hiding rules can improve the user experience by hiding unwanted page contents, but cannot provide the performance and privacy improvements that network layer blocking provides.

There are three types of rules in EasyList:

(1) **Network rules**, that identify URLs of requests that should be blocked.
(2) **Element rules**, that indicate HTML elements that should be hidden.
(3) **Exception rules**, that contradict network rules by explicitly specifying URLs that should not be blocked, even though they match a network rule.

Figure 1 shows the constitution of EasyList in February 2019. Of the 71,217 rules making up EasyList, 33,703 (47.3%) were network rules, 6,125 (8.6%) were exception rules, and 31,389 (44.1%) were element rules.

[7] http://pgl.yoyo.org/adservers/
[8] http://winhelp2002.mvps.org/hosts.htm
[9] https://pi-hole.net/
[10] https://easylist.to/pages/about.html
[11] https://easylist.to/easylist/easyprivacy.txt
[12] https://github.com/easylist/easylist
[13] https://brave.com/
[14] https://projects.zubr.me/wiki/adblock2privoxy

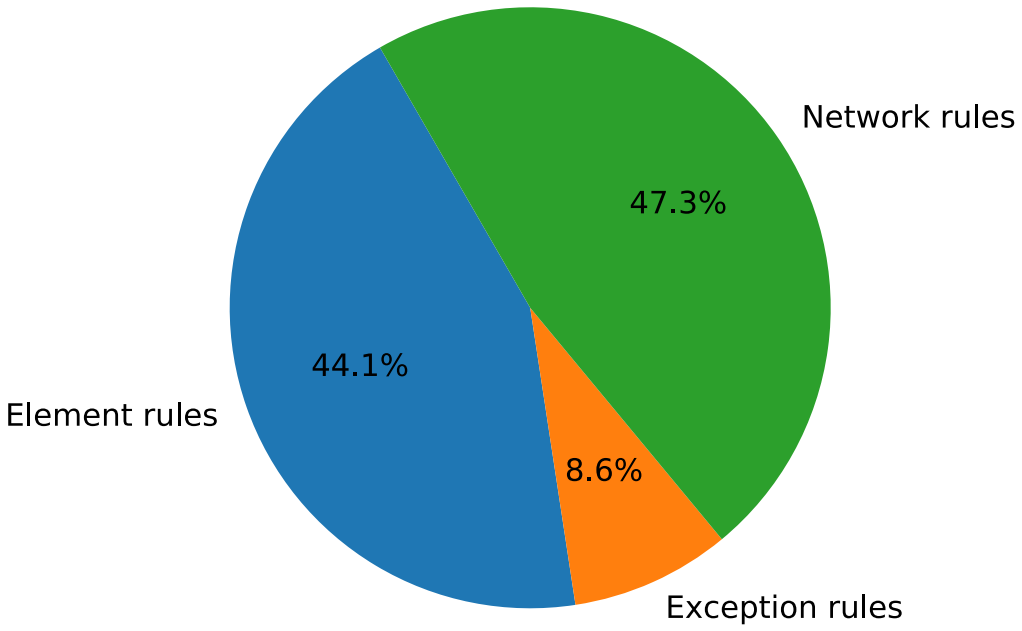

Fig. 1. Distribution of rules by type in EasyList.

3 COMPOSITION OF EASYLIST OVER TIME

This section measures how EasyList has evolved over its 9-year history, through an analysis of project's public commit history. The section proceeds by first detailing our measurement methodology, along with our findings that EasyList has grown to comprise nearly 70,000 rules and that it is primarily maintained by a very small number of people. The section concludes by showing that most rules stay more than 3.8 years in EasyList before they are removed, suggesting a huge accumulation of unused rules.

3.1 Methodology

EasyList is maintained in a public repository on GitHub. We use **GitPython** [15], a popular Python library, to measure commit patterns and authors in the EasyList repository over the project's 10-year history.

3.1.1 Measurement frequency. For every commit in the EasyList repository, we record the author and the type and number of rules modified in the commit.

First we grouped commits by day. We then checkout each day commits and measure which rules have been added and removed since the previous day. We use this per-day batching technique to

[15] https://gitpython.readthedocs.io/en/stable/index.html

avoid artifacts introduced by `git diff`, which we found causes over-estimations of the number of rules changed between commits.

3.1.2 Accounting for changes in repository structure. The structure of the EasyList repository has changed several times over the project's history. At different times, the list has been maintained in one file or several files. The repository has also included other distinct-but-related projects, such as EasyPrivacy. We use the following heuristics to attribute rules in the repository to EasyList.

When the repository consists of a single `easylist.txt` file, we check to see if the file either contains references (e.g. URLs or file paths) to lists hosted elsewhere, or contains only filter rules. When the `easylist.txt` file contains references to other lists, we treat EasyList as the union of all rules in all referenced external lists. When `easylist.txt` contains only filter rules, we treat EasyList as the content of the `easylist.txt` file.

When the repository consists of anything other than a single `easylist.txt` file, we consider EasyList to be the content of all the files matching the following regular expression `'easylist_*.txt'` and that are located in the main directory or in a directory called `easylist`.

3.2 Results

3.2.1 Rules inserted and removed. Figure 2 presents the change in the size of EasyList over time. It shows the cumulative number of rules inserted, removed, and present in the list over nine years. Rules are added to the list faster than they are removed, causing EasyList to grow larger over time. Over a 9-year period, 124,615 rules were inserted and 52,146 removed, resulting in an increase of 72,469 rules. EasyList's growth is mostly linear. One exception is the sharp change in 2013, when "Fanboy's list", another popular filter list, was merged into EasyList.

3.2.2 Modification frequency. We analyzed the distribution of the time between two commits in EasyList and observed that EasyList is frequently modified, with a median time between commits of 1.12 hours, and a mean time of 20.0 hours.

3.2.3 EasyList contributors. Contributors add rules to EasyList in two ways. First, potential contributors propose changes in the EasyList forum.[16] Second, contributors can post issues on the EasyList Github repository.

Though more than 6,000 members are registered on the EasyList forum, we find that only a small number of individuals decide what is added to the project. The five most active contributors are responsible for 72,149 of the 93,858 (76.87%) commits and changes. 65.3% of contributors made less than 100 commits.

3.2.4 Lifetime of EasyList rules. Figure 3 presents the distribution of the lifetime of rules in EasyList. The figure considers only rules that were removed during the project's history. Put differently, the figure shows how much time passed between when a rule was added to EasyList, and when it was removed, for the subset of rules that have been removed. We observe that 50% of the rules stayed more than 3.8 years (45.5 months) in EasyList before being removed.

4 EASYLIST APPLIED TO THE WEB

This section quantifies which EasyList rules are triggered in common browsing patterns. We conducted this measurement by applying EasyList to 10,000 websites every day for two months, and recording which rules were triggered, and how often.

This section first presents the methodology of a longitudinal, online study of a large number of websites. We then present the results of this measurement. We find that over 90% of rules in

[16] https://forums.lanik.us/

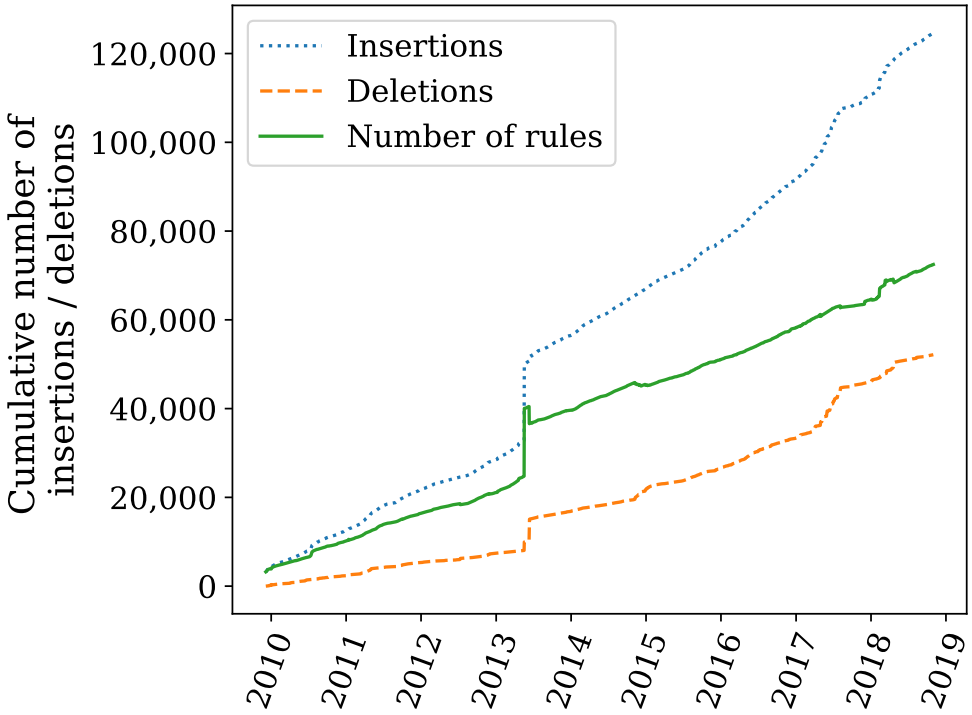

Fig. 2. Evolution of the number of rules in EasyList. Over a 10-year period, EasyList grew by more than 70,000 rules.

EasyList are never used. We also show that on average, 29.8 rules were added to the list every day, but that these new rules tend to be less used than rules that have been in EasyList for a long time.

The section concludes by categorizing and counting the ways advertisers react to new EasyList rules. We detect more than 2,000 situations where URLs are changed to evade rules and present a taxonomy of observed evasion strategies.

4.1 Omitting element rules

The results presented in this section describe how often, and under what conditions, network and exception rules apply to the web. However, as discussed in Section 2.3, EasyList contains *three* types of rules: network rules that block network requests, exception rules that prevent the application of certain network rules, and element rules that describe parts of websites that should be hidden from the user for cosmetic reasons.

We omit element rules from our measurement for three reasons. First, our primary concern is to understand how the growth and changes in EasyList affect the privacy and security of Web browsing, and element rules have no effect on privacy or security. In all modern browsers, hiding page elements has no affect on whether those elements are fetched from the network and, in the

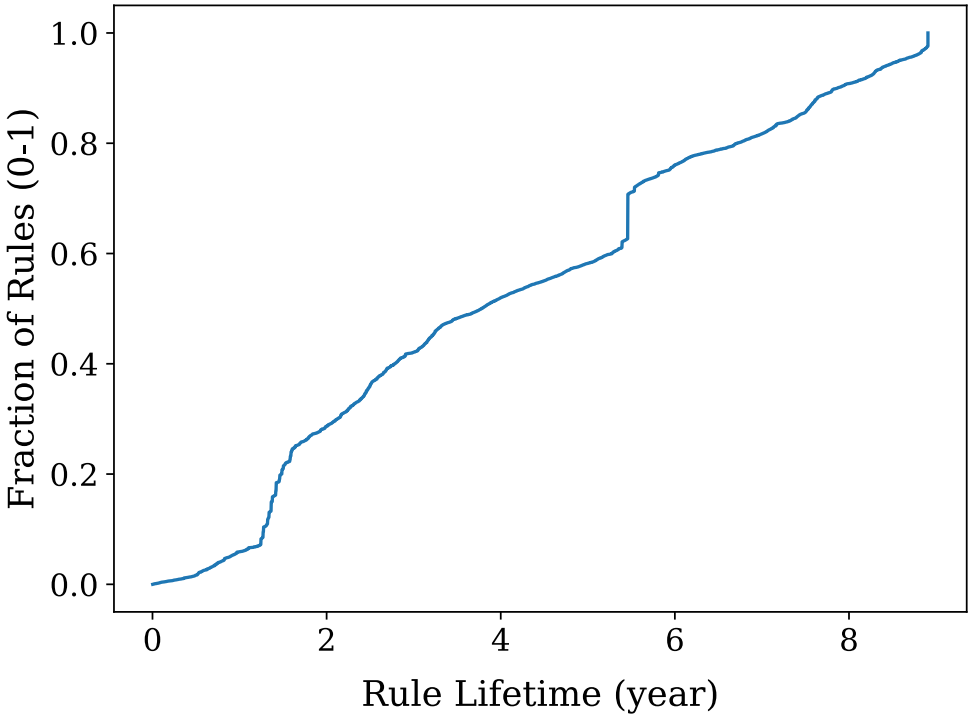

Fig. 3. Cumulative distribution function of the lifetime of the rules in EasyList. Half of the rules stay more than 3.8 years in EasyList.

case if `<iframe>` elements, rendered in memory [17]. In all but uncommon edge cases, relating mostly to memory availability, hidden elements are still fetched from network, though possibly with a lower priority.

Second, we omit element rules from the study because their application is highly variable, and would add a not-useful amount of dimensionality to the data. Element rules apply differently depending on how users interact with the page, since page layouts can change in response to user interaction and timer events. This is especially true in highly dynamic, client-side web applications, like those written in JavaScript frameworks like React [18] and Angular [19], since client-side page modifications can change which element rules apply. Network rules, in contrast, have far less (though not zero) variability over the life-cycle of a page [20].

Third, though least significantly, we omit element rules from consideration because there are common uses of EasyList where element rules are not applied, lessening the value of measuring

[17] In fact, many Web applications use this quirk of hidden iframes to create simplified, early versions of server-push communication, an approach called "long polling". Similarly, tracking scripts like Google Analytics use never-rendered, never-visible images for client-server communication.

[18] https://reactjs.org/

[19] https://angularjs.org/

[20] There are exceptions here, such as analytic scripts that initiate network requests to recorder, server side, user behaviors. However, note that these are relativity uncommon, since the initial analytic script is generally blocked.

Measurement	Counts
# days	74
# domains	10,000
# non-responsive domains	400
Avg # pages per day	29,776
Avg # pages per domain per day	3.74
Total # pages measured	3,260,479

Table 1. Statistics of the number of domains and sites measured during online EasyList measurement.

this portion of the list. The most common examples of such uses are privacy-preserving network proxies (e.g. Privoxy [21] and SquidGuard [22]).

4.2 Methodology

This subsection discusses how we measured how EasyList effects typical web browsing, including what sites were measured, the instrumentation used tool take the measurements, and what information was collected. The following subsection describes the results of executing this methodology.

4.2.1 Crawl description. To understand the usefulness of rules in EasyList, we applied EasyList to 10,000 websites every day for over two months (74 days). We selected these 10,000 websites from two groups:

(1) **Popular websites:** Websites from the top 5K Alexa, a ranking of sites online by popularity.
(2) **Unpopular websites:** 5,000 websites randomly selected from the top Alexa one-million, but not present in the set of popular websites. (i.e. rank 5,001–1 million)

We crawled the web using an instrumented version of Chromium to measure which filter rules were applicable when browsing a large number of websites in an anonymous browsing scenario. The crawls were launched from AWS Lambda instances located in the `us-east-1` region.

For each day of the experiment, we first visit the landing page of each URL in the popular and unpopular sets. We then randomly selected up to three URLs referenced in anchor tags, pointing to pages on the eTLD+1 domain, and visit these URLs. This resulted in between 10,000 and 40,000 pages being measured every day. Table 1 provides high level statistics of these measurements.

We use the Chrome devtools protocol [23] to record the following information about each network request made during page execution:

- Time of the request
- URL of the request
- URL of the domain that initiated the request
- Type of resource fetched (e.g. image, script, sub-documents)
- Hash of the response
- Size of the response

To avoid introducing side effects, we did not block any requests during our measurements. We instead first recorded all HTTP requests made when interacting with each site. Next, we applied EasyList offline using Brave's ad-blocker NodeJS module [24]. We determine if each request would

[21] https://www.privoxy.org/
[22] http://www.squidguard.org/index.html
[23] https://chromedevtools.github.io/devtools-protocol/
[24] https://github.com/brave/ad-block

have been blocked, excepted, or allowed. A "blocked" request is one that matches an EasyList network rule, indicating that the resource should not be fetched. An "excepted" request is one that matches a blocking rule, but also matches a "excepting" rule, indicating that the resource *should* be fetched anyway. An "allowed" request matches no EasyList rules.

4.2.2 Description of the dataset. Our dataset comprises every network request made during our crawl of 10,000 websites, for 74 days between July 24th, 2018 and October 5th, 2018. Among the 10,000 websites crawled daily, 400 (4.0%) never responded during the experiment. We attribute this to a mix of websites becoming inactive (common among unpopular websites [18]) and websites blocking IP addresses belonging to AWS to deter crawlers. The existence of such AWS-including IP blacklists has been documented in other work [6]. We discuss possible limitations to our study more in Section 6.

4.3 Results

4.3.1 Proportion of EasyList rules used. We consider a rule as used during a crawl if the rule matched at least one network request made during the crawl. We measure the proportion of network and exception rules used during our crawls. As noted above, we measure network and exception rules, but not element (e.g. cosmetic) rules because network rules impact performance and privacy.

We find that the vast majority of network and exception rules are never used. Only 9.84% (4, 038) of rules were used even once during our measurements.

An even smaller number of network and exception rules are *frequently* used. On average, only 5.14% of EasyList network and exception rules were used at least once per day (**Research Question 4**). We also observed that the number of active rules is stable over time (**Research Question 2**).

Figure 4 shows the cumulative distribution function of how often filter rules were used during the 74 days of the experiment. The distribution is skewed; the majority of rules are either not used (90.16%), or were used between 1 and 100 times (4.45%). Only 3.56% of the rules were used between 100 and 1,000 times, and 1.83% more than 1000 times.

4.3.2 Usefulness of EasyList additions. During the experiment, 2, 202 network and exception rules were added to EasyList, an average of 29.8 new rules per day (**Research Question 1**). We refer to rules added to EasyList during our measurement campaign as *new*; we call rules *old* if they existed in EasyList at the start of the measurement period.

The vast majority of rules, new and old, were not used during our measurements. Of the 2, 202 rules added during the study period, 208 (9.45%) were used at least once. Those measurements are roughly similar for old rules (9.84%). However, when considering only rules that were used at least once, we found that new rules were used nearly *an order of magnitude less* than old rules. This suggests a declining marginal usefulness per rule as EasyList accumulates more rules, possibly because the most troublesome resources are already blocked. If a new rule was used during the study, it was used an average of 0.65 times per day. Old rules were applied much more frequently, 6.14 times a day on average.

4.3.3 Impact of the age of rules. We also measure whether the age of a rule impacts its use. We find that the age of a rule significantly impacts the number of times rules are used. We present these findings in two ways: graphically and statistically.

Figure 5 presents the distribution of the age of the rules present in EasyList, as well as the average number of uses depending on the age of a rule. The distribution of the age of the rules (black bars) shows that there are rules from all ages in EasyList. The important number of 5-year-old rules is explained by the merge of the "Fanboy's list", another popular filter list, with

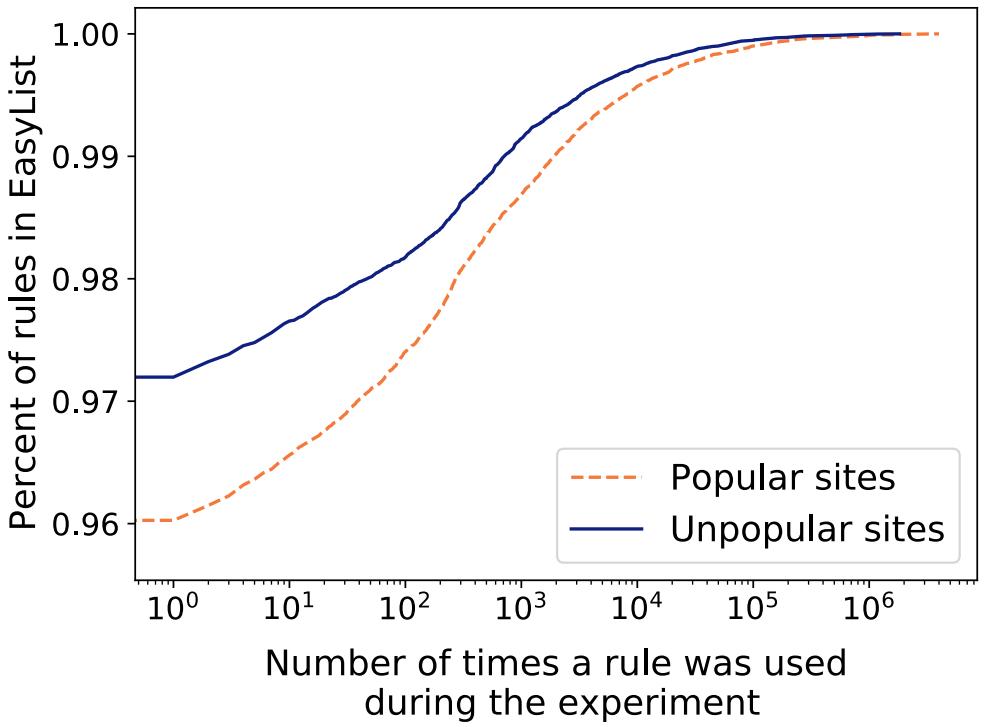

Fig. 4. Distribution of the number of times rules were used during the 74 days of the experiment, on popular (Alexa rank 1–5k) and unpopular (Alexa rank 5,001–1m).

EasyList in 2013. If we observe the average number of times rules are used in a day (grey bars), we see that the most useful rules are old. This is caused by generic rules blocking URLs that contained keywords such as "ads" or popular domains such as doubleclick.net. For example, the rule `.com/ads/$image,object,subdocument` was added to EasyList in September 2010 and was triggered 748,330 times during the experiment. We did not observe a linear relationship between the average use of a rule and its age.

Besides the graphical analysis, we also conduct statistic tests to determine whether the age of a rule impact its use. We use the Kolmogorov-Smirnov test to compare the distribution of the number of times rules are used depending on the duration they have been present in EasyList. For each year y_i between 1 and 8, we compare the distribution of the number of times rules that have been present y_i years in EasyList have been used with:

(1) the distribution of the number of times rules that have been present less than one year,
(2) the distribution of the number of times rules that have been present between y_{i-1} and y_i years.

For all the tests, we obtain p-values less than $10^{-}5$, which indicates that there are significant differences in the way rules are used depending on their age (**Research Question 3**).

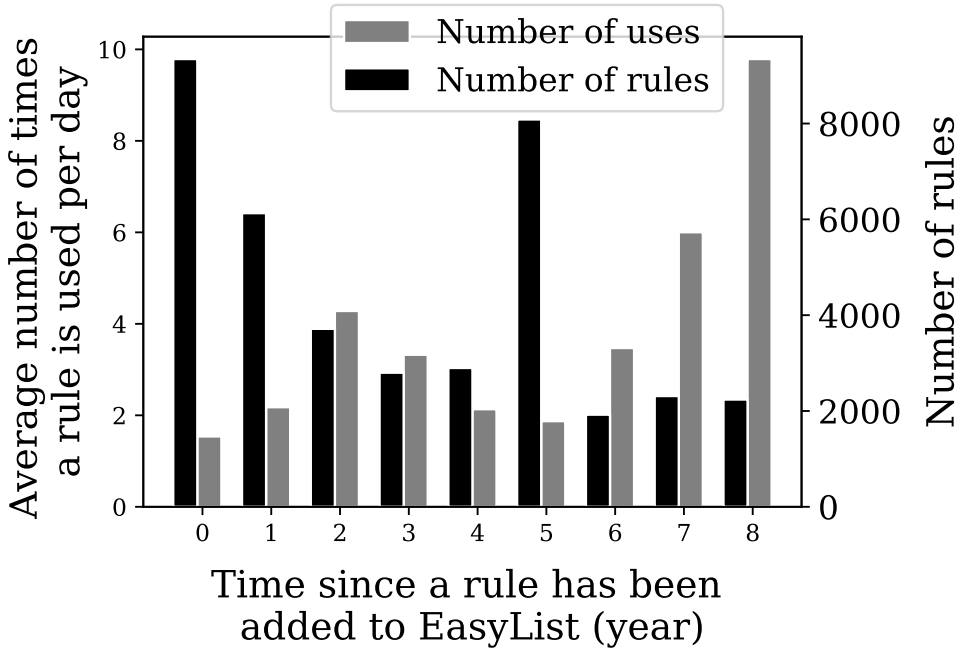

Fig. 5. The black bars represent the distribution of the age of the rules present in EasyList. The grey bars represent the average time a rule is used per day against the time it has been added to EasyList.

4.4 Advertiser Reactions

EasyList helps users avoid online advertising. Advertisers and websites that rely on advertising for their income do not want their content to be blocked and may try to circumvent rules in EasyList. There are several ways an advertiser may do this. One option is to try to detect the use of an adblocking tool [7]. Alternatively, the advertiser may manipulate the URLs that serve their content, to prevent matching filter rules. In this section, we measure how often, and in which ways, advertisers responded to EasyList rules. We do not observe a statistically significant reaction by advertisers *in general* but we note common patterns in avoidance strategies among a subset of advertisers (**Research Question 5**).

We measured advertiser reactions to EasyList rules through the following intuition: if a resource changed URL more frequently after being blocked by EasyList than before, it suggests an advertiser trying to evade EasyList. Similarly, if the number of times a resource was blocked spiked after the EasyList addition, and then reverted to its pre-rule block-rate, that would also suggest advertiser evasion.

We used this intuition through the following steps. First, we considered only rules that were added during the measurement period, and which remained in EasyList for at least 14 days. Second, we identified resources that were blocked by these new rules and looked to see if the same resource (as determined by the content's hash), was served by different URLs during the study. Third, we filtered out resources that were less than 50KB, to avoid resources that were trivially identical, like common logos and tracking pixels. Fourth, we measured whether the number of URLs a resource was served from changed significantly before and after being blocked by EasyList.

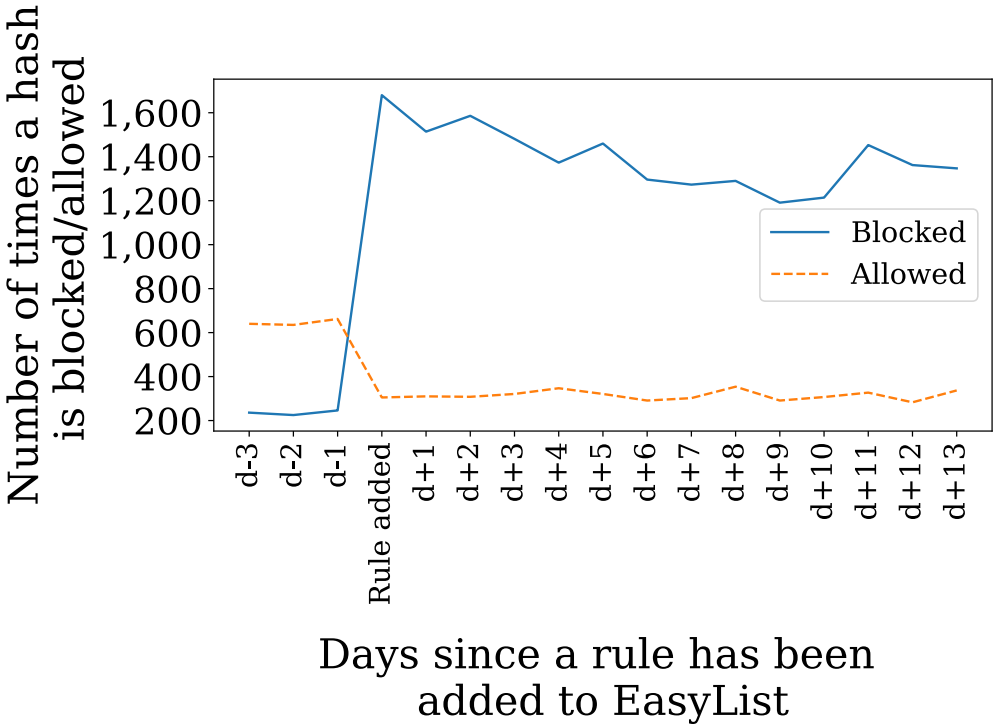

Fig. 6. Number of times resources are blocked or allowed after a rule is added to EasyList

Figure 6 presents the block and allow rates for resources affected by new rules. We did not find any population-wide trends of advertisers modifying URLs to avoid EasyList. If advertisers were, in general, successfully evading EasyList, we would observe a decrease in blocking over time. Block rates did not though, in general, revert to pre-rule levels over time.

4.5 Evasion Strategies

In this subsection, we present a partial taxonomy of the strategies used by advertisers to avoid EasyList rules. Figure 7 presents the number of times each evasion strategy was observed.

4.5.1 Changing Domains. Many advertisers changed the domains their resources were served from, either by subtly modifying the domain names or by completely changing them through domain generation algorithm style techniques. This happened 1,612 times and does not take into account resources that were moved to the first party. For example, the URL https://c.betrad.com/geo/ba.js?r170201 was blocked by the rule `||betrad.com^$third-party`. The resource was moved to a new domain, c.evidon.com, to avoid being blocked.

In total, resources were moved from 100 distinct domains to 157 distinct domains, representing a total of 195 distinct combinations of domains. The two most frequent transitions are resources moved from pagead2.googlesyndication.com to google.com and from cs03.etcodes.com to cs03.et-cod.com. The former occurred 499 times and the latter 185 times. We observe that the advertiser changed the domain that served the resources so that it looks the same when observed by a human but that does not match the filter anymore.

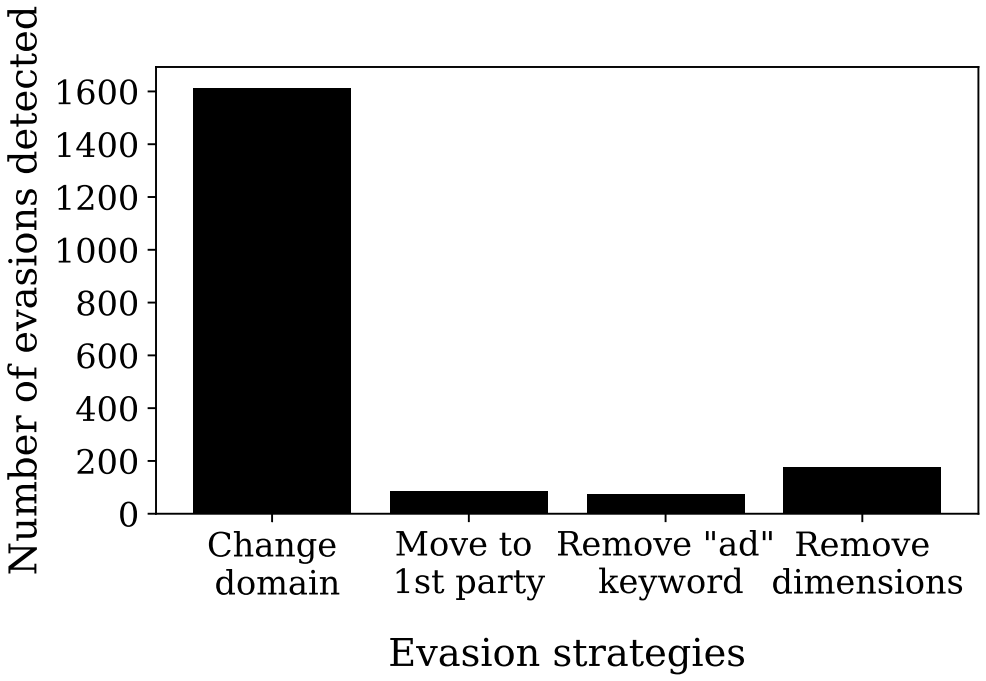

Fig. 7. Number of times each strategy has been used during the experiment.

4.5.2 Moving Resources to the First Party. Advertisers avoided EasyList rules by moving resources from third-party domains to the first party. It happened 84 times, among which 23 times resources were moved to another sub-domain of the first party. For example, we observed the domain cnn.com including resources from ssl.cdn.turner.com, which was blocked by the rule `||turner.com^*/ads/`. We then observed the same resource being served from cdn.cnn.com directly, which prevented the resource from matching the `||turner.com` domain in the filter rule.

4.5.3 Removing Ad Keywords from URLs. Keywords such as 'ads' or 'ad' trigger filter rules. We observed 73 URLs where these keywords were simply removed. For example, the URL https://etherscan.io/images/ad/ubex-20.png was blocked by the rule `/images/ad/*`. To bypass the filter rule, the URL was changed to https://etherscan.io/images/gen/ubex-20.png.

4.5.4 Removing Image Dimensions from URLs. Some URLs contain parameters to specify the dimension of the ad banners. These parameters also trigger filter rules. Advertisers can evade these rules by removing matching parameters from URLs. We observed 176 Evasions of this kind. For example, https://s0.2mdn.net/dfp/.../lotto_kumulacja_160x600_009/images/lotto_swoosh.png was blocked by the rule `_160x600_`. We observed an advertiser removing the dimension parameter from their URLs to avoid being blocked.

5 APPLICATIONS

In this section, we present two practical applications of the previous sections' findings: first, an optimized, reduced EasyList on resource constrained iOS devices, and second, a novel EasyList-based filtering strategy targeting desktop extensions, that provides nearly all of the blocking benefits of full EasyList, but with significantly improved performance.

5.1 Improving Content Blocking on iOS

We first present how content blocking in iOS differs from content blocking in other platforms. Then, we run a benchmark that measures how the size of a filter list impacts the time to launch a content-blocking application on iOS, and how our findings could help to decrease this time.

5.1.1 Overview of Content Blocking on iOS. iOS and Safari use a different strategy for content blocking than other browsers [25]. On most platforms, ad-blocking tools receive information about each request, such as the request URL, the expected resource type, etc. The extension can then apply whatever logic is desired to determine whether the request should be blocked. In most content blocking systems, this results in a large number of regular expressions (or similar text patterns) applied to the URL, along with some optimization to limit the number of rules that need to be considered.

iOS and Safari (along with Google's proposed Manifest v3 changes[26]) use a different approach, where extensions declare a static set URL patterns that should be blocked but *do not execute the rule application logic*. This protects the user from malicious extensions (since extensions cannot modify requests with malicious code), at the cost of requiring all rules be expressed in a format *that is less expressive than the EasyList format*. The result is that EasyList is generally expanded from EasyList's compact rule format to a larger number of iOS-compatible rules.

Limitations. There are two relevant limitations in iOS's blocking approach. First, iOS enforces a limit of 50K rules. This limit is not in Apple's documentation, but the main ad-blocking applications report it [27], and we observe the same during testing. Since Easylist alone contains 40K network rules, little room is left for either other popular lists (e.g. EasyPrivacy) or region specific EasyList supplements (e.g. EasyList China). iOS's restrictions thus limit the amount of protection users can deploy.

This limit on the number of rules is particularly harmful to non-English speaking users, who often need to rely on supplemental, region or language specific filter lists, that are applied in addition to EasyList. As EasyList itself is large enough to consume the iOS limit on filter rules, non-English users cannot use EasyList with their regional list, resulting in reduced protections [19].

Second, iOS compiles filter rules into a binary format each time rules are updated. As we show in the benchmark we conduct, users may have to wait for 14 seconds or more when a list composed of 40K rules is compiled. This is particularly unacceptable when launching an app for the first time when tricks like background compilation cannot be used to hide the cost from users.

5.1.2 Benchmark.

Approach. We use the findings of this work to decrease rule compilation cost (and increase the ability of users to include other lists of rules) on iOS devices by only compiling filter rules that are likely to be useful, instead of the full set of 40k rules. We show that reducing EasyList to only its useful rules provides a dramatically improved initial launch experience for users, and gives users

[25] https://developer.apple.com/documentation/safariservices/creating_a_content_blocker
[26] https://docs.google.com/document/d/1nPu6Wy4LWR66EFLeYInl3NzzhHzc-qnk4w4PX-0XMw8/edit
[27] https://help.getadblock.com/support/solutions/articles/6000099239-what-is-safari-content-blocking-, https://www.ublock.org/blog/introducing-ublock-safari-12/

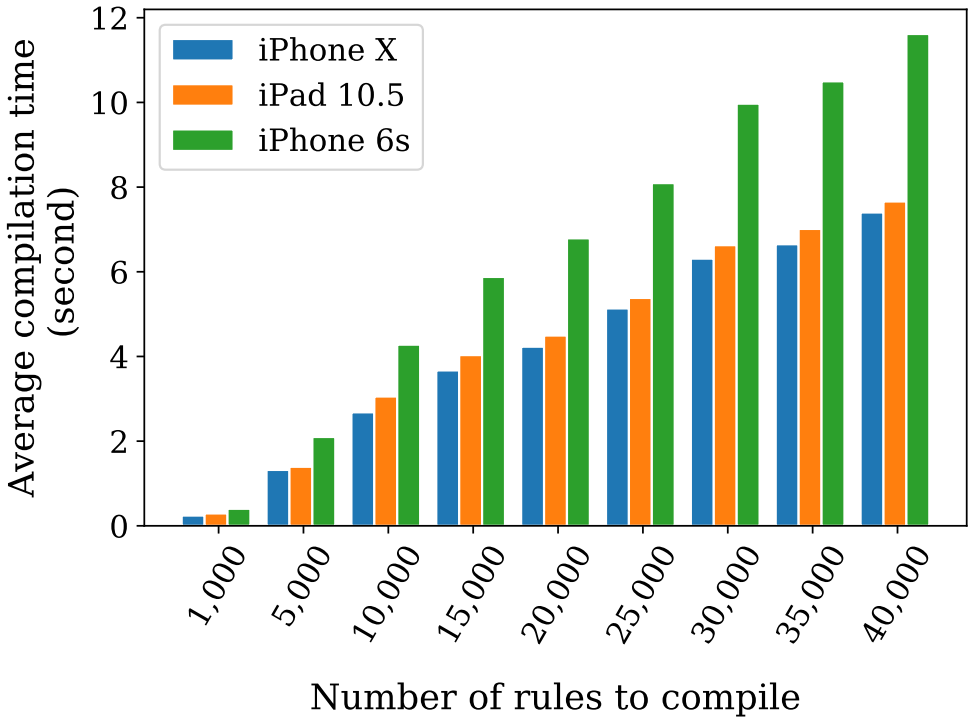

Fig. 8. Average time to compile a filter list depending on device and size of the list.

more flexibility to apply additional filter lists. The primary beneficiaries of this optimization are non-English speakers, and those on reduced capability mobile devices.

Evaluation Methodology. We first measure the costs of compiling different sizes of filter lists on different popular iOS devices. We generate lists that contain between 1,000 and 40,000 rules randomly selected from the set of network and exception rules in EasyList. For each of the lists, we use a fork of the "ab2cb" library [28] to convert the rules from the Adblock Plus format to the iOS JSON format. Then, for each device and each list, we compile each iOS filter list 5 times and report the average compilation time.

Results. Figure 8 shows the average compilation time for each device and each selected list size. The compilation times grows linearly with respect to the number of rules. While on average it takes 0.24 second to compile a list composed of 1,000 rules on an iPhone X, it grows to 7.4 seconds for a list composed of 40,000 rules.

Device kind also impacts compilation time. Compiling 40k rules on an iPhone 6s takes on average 11.62s, 4.2 seconds longer than on an iPhone X.

Thus, keeping only active rules has two main benefits in the case of ad-blockers running on iOS and Safari. First, it allows users to enjoy the benefits of EasyList while remaining well under the platform's 50K rule-limit. Second, it dramatically decreases the required compilation time for the first time the application is launched.

[28] https://github.com/brave/ab2cb

5.2 Improving ad-blocking performance

We also propose an optimized strategy for applying EasyList that provides nearly all of the benefits of traditional ad-blockers while improving filter list application speed. We describe our technique in three steps. First, we describe how current tools use EasyList for blocking (using AdBlock Plus, the most popular of such tools [29], as a representative example). Second, we present a "straw" blocking strategy that considers only frequently used filter rules. Third, we propose a novel hybrid strategy that achieves nearly the accuracy of existing techniques with the performance improvements of the "straw" strategy.

Our hybrid approach achieves over 99% of the coverage of the current most popular EasyList tool, but performs 62.5% faster. Because of the nature of the optimizations in this hybrid strategy, we expect it could be applied to other EasyList consuming tools to achieve similar performance improvements. This hybrid approach achieves this performance improvement at the cost of some user privacy, since resources blocked by infrequently used EasyList rules would still be loaded once. We note that this privacy "cost" is only paid once, while the performance improvement is ongoing, and so might be an appealing trade off to even privacy-sensitive users.

Strategy One: Synchronous Full EasyList. Most EasyList tools decide whether a network request should be blocked as follows:

(1) Use hardcoded heuristics, such as not blocking top-level documents or requests coming from non-web protocols. If any of these heuristics match, allow the request.
(2) Check the requested URL against the small number "exception" rules in EasyList. If any "exception" rules match, allow the request.
(3) See if the requested URL matches any of the "network" rules in EasyList. If any "network" rule matches, block the request.
(4) Otherwise, allow the request.

We note two performance impacting aspects of this strategy. First, it performs a large number of unnecessary computation, since every "exception" and "network" rule in EasyList is applied to outgoing request, even though the vast majority (over 90.16%) are very unlikely to be useful (again, based on the measurements described in Section 4). Second, this wasteful computation adds delay to a time-sensitive part of the system. These filter checks are conducted synchronously, blocking all outgoing network requests until all EasyList rules are considered.

Strategy Two: Synchronous Reduced List. Next, we describe a straw-man strategy, that improves performance by only considering the 9.84% of rules expected to be useful. This strategy is otherwise identical to strategy one and differs only in the number of EasyList rules considered. Instead of applying all of EasyList's 38,710 "network" and "exception" rules, this strategy only evaluates the 4,038 rules observed during the online measurements discussed in Section 4. The expected trade-off here is performance for coverage since resources that match rarely used filters will be allowed.

Strategy Three: Synchronous Reduced List, Asynchronous Complementary List. Finally, we present our proposed blocking strategy, a hybrid strategy that achieves nearly the coverage that full EasyList provides while achieving the performance improvements of the reduced list. Figure 9 outlines this hybrid strategy.

This strategy uses two steps:

(1) A synchronous, request-time matcher, that operates before each request is issued, but with a reduced version of EasyList.

[29] https://data.firefox.com/dashboard/usage-behavior

(2) An asynchronous background matcher, that applies the uncommon tail of EasyList, but only when a request has been allowed by the previous step.

The first step is identical to strategy two. Every outgoing network request is intercepted and blocked until the frequently-used subset of EasyList rules is considered. The step's goal is to minimize how long network requests are blocked, by minimizing the amount of synchronous work. The result is that benign network requests complete more quickly than current blocking tools.

The second step applies the remaining, long-tail of EasyList rules, but at a less performance-sensitive moment. If a network request is allowed by the first step, the request is issued, but the browser continues checking the now-issued URL against the rest of EasyList. This continued checking is done asynchronously so that it has minimal effect on the load time of the page.

If the asynchronous checker finds any rules that match the URL of the now-issued request, that rule is added to the set of rules applied by the synchronous matcher so that it will be quickly blocked in the future.

The result of this hybrid strategy is that commonly blocked requests are blocked quicker (because the synchronous blocking step is considering a smaller rule set), benign requests complete faster (again, because of the reduced rule list used in the synchronous blocker), and rare-but-undesirable URLs are adjusted to (because the asynchronous matcher moves matching filter rules into the synchronously-applied set).

5.2.1 Evaluation Methodology. We evaluated the performance of each strategy by implementing them in AdBlock Plus for Chrome [30]. In all strategies, we instrumented AdBlock Plus to measure the time needed to evaluate each outgoing network request against EasyList (or the relevant subset of EasyList). For the hybrid approach, we also added timing measurements to the asynchronous step.

We evaluated each of the three blocking strategies against the same selection of popular and unpopular websites discussed in Section 4. We conduct the crawl on an AWS T2 medium instance with 2 virtual CPUs and 4GB of RAM. Since Chromium does not support extensions in headless mode, [31] we use stock Chromium rendered with XVFB, and automated the system with the Puppeteer library [32]. All experiments were conducted with caching disabled.

For each strategy we visited each of the 10K websites in our sample. We allowed each website five seconds to respond and then allowed each website to execute for two seconds. The extension measures the time taken by AdBlock Plus (modified or stock) to decide whether to block each network request on each page.

5.2.2 Performance of Blocking Strategies. Table 2 presents the results of applying the above evaluation methodology against each of the three blocking strategies.

The first row presents measurements for the stock AdBlock Plus implementation, which uses a synchronous blocking strategy for all of EasyList. Unsurprisingly, this strategy takes the longest time to determine whether to block a network request. We note that this time is spent blocking each network request, which greatly impacts page load time.

The second row shows the performance of the second strategy; reducing EasyList to its most frequently used rules, and applying that reduced list synchronously. The result is faster performance, at the cost of an increased false positive false negative rate. The number of network requests blocked *goes up* because of "exception" pruned from EasyList. Privacy is also harmed, as nearly 18,000 more

[30] https://github.com/adblockplus/adblockpluschrome
[31] https://bugs.chromium.org/p/chromium/issues/detail?id=706008
[32] https://github.com/GoogleChrome/puppeteer

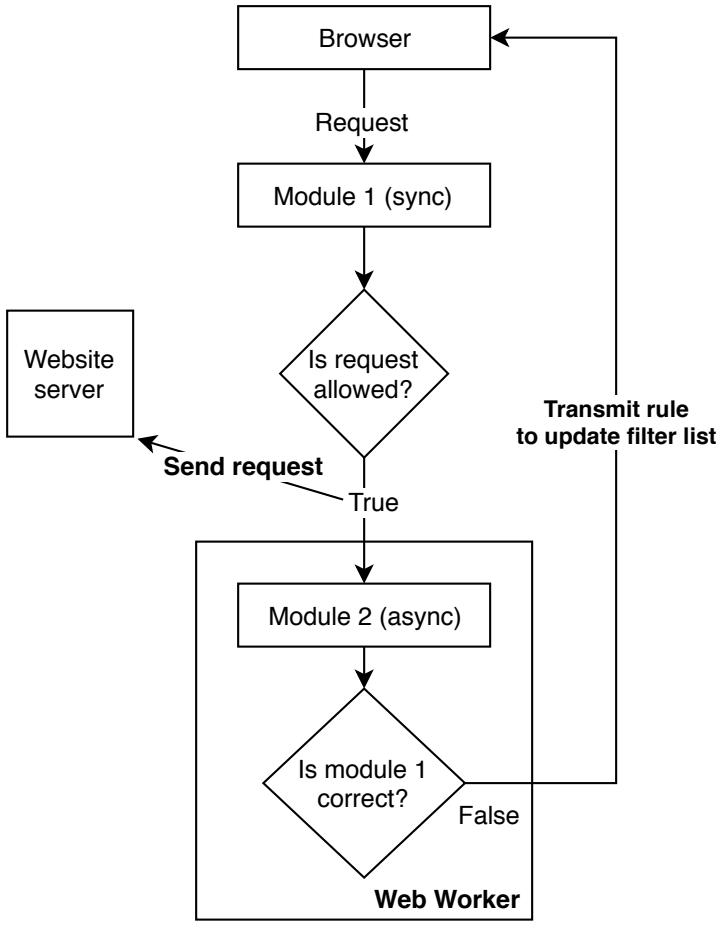

Fig. 9. Overview of the proposed hybrid strategy.

	Strategy	Num rules start	Num rules end	Median eval time per request (ms)	90th time per request (ms)	Num requests blocked	Num requests exceptioned	Num 3rd parties contacted
(1)	Easylist sync	39,132	39,132	0.30	0.50	30,149	11,905	322,604
(2)	Reduced list sync	3,259	3,259	0.10	0.30	30,611	2,508	340,301
(3)	Hybrid combined	39,132	39,132	0.30	0.60	31,584	14,444	338,841
(3.1)	Hybrid sync	3,259	3,445	0.20	0.30	31,446	14,396	-
(3.2)	Hybrid async	35,873	35,687	0.20	0.30	138	48	-

Table 2. Performance and coverage comparison for three EasyList application strategies.

third-parties are contacted during the evaluation, a result of some "network" rules missing in the reduced EasyList.

The remaining rows present the evaluation of the hybrid strategy. Rows four and five describe the synchronous and asynchronous modules of the hybrid strategy separately, while row three presents the combined effect. The most significant results of our evaluation are the following.

First, the synchronous module takes 0.21 ms on average to process a request. Perceived blocking time is reduced (compared to stock AdBlock Plus) by 62.5% (**Research Question 6**). Second, the hybrid strategy provides blocking coverage nearly identical to stock AdBlock Plus (> 99%), with only 138 false negatives on 10, 000 websites visited. The 48 "exception" rule errors do not impact the user since the rules that would have been excepted were not added to EasyList. Third, the evaluation shows the adaptive benefit of the hybrid model. The hybrid approach initially applied 3,259 rules synchronously, but after the 10,000 site evaluation, 186 rules from the uncommon async set were added to the synchronous set.

Second, we note the asynchronous portion of the hybrid approach applies its 35K rules faster than the reduced-list synchronous approach, which considers only 3,259 rules. This surprising observation is due to the synchronous portion of the hybrid approach doing some work (e.g. what kind of resource is being fetched) that can be reused in the asynchronous step.

Overall, we find that the hybrid approach is a successful, performant strategy. The hybrid strategy considers only the subset of EasyList that is likely to be useful in the performance critical path, and defers evaluating the less-likely-to-be-useful rules to a less critical decision point. The hybrid approach achieves these improvements with a minimal effect on blocking accuracy, and at a small (though not zero) privacy cost, on the order of one non-blocked resource per uncommonly used filter list rule.

Finally, we note that there are potential further performance improvements that might be achieved by pushing the hybrid approach further, and starting each user with an empty set of filter rules. This would increase the privacy cost of the hybrid approach, since a larger number of would-be-blocked resources would be fetched, but would result in an even smaller, further optimized set of rules that tightly matched each users' browsing patterns.

6 LIMITATIONS AND DISCUSSION

6.1 Web site selection generalizability

The findings in this study depend on having a sample of the web that generalizes to the types of websites users visit and spend time on. We treat the Alexa 5K, along with a sampling of less popular websites, as representative of typical browsing patterns. While we expect this set to be a good representative of the web as a whole (largely because the highly skewed distribution of website popularity means the most popular sites represent the majority of most user's browsing time), we note it here as a limitation, and that extending this work's measurements beyond the Alexa 5k would be valuable future work.

Additionally, this work considers each site's landing page, and up to three sub-pages linked to from the site's landing page, as representative of the site's content overall. If this is not the case, and deeply nested pages have different kinds and amounts of resources than higher-level pages, it would reduce the generalizability of this work's findings. We note this as an area for future work.

6.2 Web region and language generalizability

This work applies EasyList globally popular sites, as determined by the Alexa global rankings. This choice was made because EasyList itself targets English and "global" sites. Other lists target others languages and regions on the web. Some of these lists are maintained by the EasyList project [33],

[33]https://easylist.to/pages/other-supplementary-filter-lists-and-easylist-variants.html

others lists are created by other filtering tools [34] or crowdsource efforts [35]. It would be interesting future work to understand how EasyList performs on other regions of the web (as compared to English and "global" sites), and how EasyList's performance compares to region-and-language-specific lists.

6.3 Automated measurement generalizability

All of our results were generated from automated crawls, which also may have affected how generalizable our results are. It is possible that different kinds of resources are fetched and so different parts of EasyList are used, when users interact with websites in particular ways, such as logging in or using web-app like functionality. How generalizable automated crawl results are to the browsing experiences of real users is a frequently acknowledged issue in measurement studies (e.g. [20]), and one we hope the community can address with future work.

Additionally, all crawling done in this work was carried out from well known AWS IP addresses. This means that the results may be affected by the kinds of anti-crawling techniques sometimes deployed against Amazon IP addresses. This, in turn, could have affected the number and distribution of ads observed during measurement. While this is a common limitation of this kind of web-scale measurement, we note it as another limitation.

6.4 Relationship to filter list evasion

Many websites prefer for their included resources not be blocked by filter lists, for reasons ranging from monetization to anti-fraud efforts. Some sites and advertisers attempt to evade filter lists (and other blocking tools). Some of these techniques are presented in Section 4.4, and others have been detailed in other research and discussed in Section 7.4.

While, if effective, these evasion efforts would would reduce the usefulness of filter-list based blocking, evasion efforts are unlikely to be effective in the common case. First, advertisers and trackers are constrained in their ability to evade filter lists because frequently changing URLs would break existing sites that have hard coded references to well known URLs. Second, frequently changing URLs imposes a non-zero cost on advertisers and trackers by making caching difficult, and so increasing serving costs. Finally, though trackers may consider evading filter lists with more expensive techniques (e.g. domain generation algorithms, resource inlining, etc), many may be hesitant to do so because, in the long term, more sophisticated efforts will likely be defeated too, since the client has the ultimate ability to choose what content to fetch and render, for reasons described by Storey et. al. [21].

6.5 Varying resource blocking importance

Finally, our results consider every blocking action as equally useful. In our measurements, a rule that blocks ten resources is implicitly ten times more useful than a rule that only blocks one request. It is possible, though, that a less frequently used rule may be more beneficial to the user than a frequently used rule if the infrequently blocked rule is blocking a very malicious or performance harming resource. While we expect the most severe, security harming resources are most commonly dealt with through other blocking tools, such as SafeBrowsing [36], we acknowledge this limitation, but treat the more general question of "how beneficial is it to the user to block a given resource" beyond the scope of this work.

[34] e.g. https://kb.adguard.com/en/general/adguard-ad-filters
[35] e.g. https://filterlists.com/
[36] https://safebrowsing.google.com/

7 RELATED WORK

7.1 Online tracking

Recent studies document a growth of third-party tracking on the web [2, 10]. Yu et al. [23] found an increase in analytics services and social widgets. Englehardt et al [2] showed tracking code on more than 10% of the websites of the top Alexa 1M. Libert et al [12] showed that 180k pages of the top 1M Alexa websites had cookies spawned by the DoubleClick domain, a behavioral advertising company.

7.2 Defenses against tracking

The NoScript extension [37] enables to prevent JavaScript execution. While this approach blocks trackers, it also breaks websites that use JavaScript for legitimate purposes. This trade-off between privacy and usability is important. Yu et al. [23] find that privacy tools that break legitimate websites may lead to users deactivating such tools, harming user privacy. The most popular kind of tracking protection is browser extensions such as Ghostery, Disconnect or uBlock origin, as well as browsers such as Brave or Safari that enable to block third-party requests.

A variety of strategies have been proposed for identifying unwanted web resources. Privacy Badger uses a learning-based approach. Iqbal et al. [8] also proposed a machine learning approach that considers features extracted from HTML elements, HTTP requests, and JavaScript to determine if a request should be blocked. Storey et al. [21] propose a visual recognition approach targeting legally mandated advertising identifiers. Yu et al [23] proposed a crowdsourced approach where users collectively identify data elements that could be used to uniquely identify users. The majority of anti-tracking and ad-blocking tools rely on filter lists.

Different studies [17] show that tracker blockers and ad-blockers are popular among the general population. Malloy et al [13] showed that depending on the country, between 16% and 37% of the Internet users had an ad-blocker installed. Mathur et al [14] found that most users of anti-tracking tools use the tools to avoid advertising.

7.3 Effectiveness of anti-tracking tools

Gervais et al. [4] quantified the privacy provided by the main ad-blockers. They show that on average, using an ad-blocker with the default configuration reduce the number of third parties loaded by 40%. Merzdovnik et al. [15] showed that rule-based approaches can outperform Privacy Badger's learning-based approach. They also show that extensions that rely on community-based lists are less effective than extensions based on proprietary lists such as Disconnect or Ghostery when used with the correct settings. Their study demonstrates that besides blocking trackers, most of these extensions have a negligible CPU overhead. In some cases, it even leads to a decrease in the overall CPU usage of the browser.

7.4 Maintaining filter lists

In order to keep up with new domains creating and domains changing their behavior, it is crucial to maintain filter lists. Because it is a cumbersome task and it needs to be done carefully not to break websites, Gugelmann et al. [5] proposed an automated approach that relies on a set of web traffic features to identify privacy invasive services and thus help developers maintaining filter lists.

Alrizah et al. [1] studied the related problem of how filter lists maintainers detect and address blocking errors, and how advertisers attempt circumvent filter lists. They find that popular lists have non-trivial false positive and false negative rates, and that these errors are exploited by attackers (i.e. advertisers).

[37] https://noscript.net/

Other researchers have also documented strategies advertisers use to evade blocking. Wang et al [22] found advertisers randomizing HTML identifiers and structure. Facebook has applied this technique too [38]. Adversity has also been discussed by recent studies on anti-ad-blockers [7, 16, 25], i.e. scripts whose purpose is to detect and block ad-blockers to deliver advertising to more users. Iqbal et al. [7] conducted a retrospective measurement study of anti ad-block filter lists using the Wayback machine, and found that 8.7% of popular sites have at one time used anti-adblocking scripts.

8 CONCLUSION

This paper studies EasyList, the most popular filter list used for blocking advertising and tracking related content on the web. We find that the vast majority of rules in EasyList are rarely, if ever, used in common browsing scenarios. We measure the number of these "dead weight" rules, and effect on browser performance, through comparison with alternative, data-driven EasyList application strategies. We find that by separating the wheat from the chaff, and identifying the small subset of EasyList filter rules that provide common benefit for users, EasyList's benefits can be efficiently enjoyed on performance constrained mobile devices. We also use these findings to propose an alternate blocking strategy on desktops that improves performance by 62.5%, while capturing over $> 99\%$ of the benefit of EasyList.

More broadly, we hope this work will inform how similar crowdsourced security and privacy tools are developed and maintained. As previous work [9] has identified, such lists tend to accumulate cruft as they accumulate new rules. Over time, the benefit of such tools risks being outweighed by the amount of dead weight pulled with them. We hope the findings in this work highlight the need for regular pruning of these lists, to keep them lean and as helpful to users as possible.

9 ACKNOWLEDGEMENTS

The authors would like to thank Yan Zhu, who suggested the core of the optimization described in Section 5.2, and Ryan Brown (i.e. "Fanboy"), a core EasyList maintainer who was helpful throughout this project.

REFERENCES

[1] Mshabab Alrizah, Sencun Zhu, Xinyu Xing, and Gang Wang. Errors, misunderstandings, and attacks: Analyzing the crowdsourcing process of ad-blocking systems. 2019.

[2] Steven Englehardt and Arvind Narayanan. Online Tracking: A 1-million-site Measurement and Analysis Steven. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, (1):1388–1401, 2016.

[3] Kiran Garimella, Orestis Kostakis, and Michael Mathioudakis. Ad-blocking: A study on performance, privacy and counter-measures. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 259–262. ACM, 2017.

[4] Arthur Gervais, Alexandros Filios, Vincent Lenders, and Srdjan Capkun. Quantifying web adblocker privacy. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10493 LNCS:21–42, 2017.

[5] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. An Automated Approach for Complementing Ad Blockers' Blacklists. *Proceedings on Privacy Enhancing Technologies*, 2015(2):282–298, 2015.

[6] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. Cloak of visibility: detecting when machines browse a different web. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 743–758. IEEE, 2016.

[7] Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists. *ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, 13, 2017.

[38] https://newsroom.fb.com/news/2016/08/a-new-way-to-control-the-ads-you-see-on-facebook-and-an-update-on-ad-blocking/

[8] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. AdGraph: A Machine Learning Approach to Automatic and Effective Adblocking. 2018.

[9] Omer Katz and Benjamin Livshits. Toward an evidence-based design for reactive security policies and mechanisms. *arXiv preprint arXiv:1802.08915*, 2018.

[10] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. *Usenix Security*, 2016.

[11] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 674–686. ACM, 2012.

[12] Timothy Libert. Exposing the hidden web: An analysis of third-party HTTP requests on 1 million websites. *International Journal of Communication*, 9(1):3544–3561, 2015.

[13] M Malloy, M McNamara, A Cahn, and P Barford. Ad blockers: Global prevalence and impact. *Imc'16*, 14-16-Nove:119–125, 2016.

[14] Arunesh Mathur, Jessica Vitak, Arvind Narayanan, and Marshini Chetty. Characterizing the Use of Browser-Based Blocking Extensions To Prevent Online Tracking. *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 2018.

[15] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Block Me if You Can: A Large-Scale Study of Tracker-Blocking Tools. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, pages 319–333, 2017.

[16] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E. Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J. Murdoch. Ad-blocking and counter blocking: A slice of the arms race. *CoRR*, abs/1605.05077, 2016.

[17] Enric Pujol, Oliver Hohlfeld, and Anja Feldmann. Annoyed users: Ads and ad-block usage in the wild. In *Proceedings of the 2015 Internet Measurement Conference*, pages 93–106. ACM, 2015.

[18] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D Strowes, and Narseo Vallina-Rodriguez. A long way to the top: Significance, structure, and stability of internet top lists. *arXiv preprint arXiv:1805.11506*, 2018.

[19] Alexander Sjosten, Peter Snyder, Antonio Pastor, Panagiotis Papadopoulos, and Benjamin Livshits. Generation of filter lists for regions that are underserved. *WWW*, 2020.

[20] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–194. ACM, 2017.

[21] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. The Future of Ad Blocking: An Analytical Framework and New Techniques. 2017.

[22] Weihang Wang, Yunhui Zheng, Xinyu Xing, Yonghwi Kwon, Xiangyu Zhang, and Patrick Eugster. WebRanz: web page randomization for better advertisement delivery and web-bot prevention. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pages 205–216, 2016.

[23] Zhonghao Yu, Sam Macbeth, Konark Modi, and Josep M. Pujol. Tracking the Trackers. *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pages 121–132, 2016.

[24] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 373–380. ACM, 2014.

[25] Shitong Zhu, Xunchao Hu, Zhiyun Qian, Zubair Shafiq, and Heng Yin. Measuring and disrupting anti-adblockers using differential execution analysis. In *The Network and Distributed System Security Symposium (NDSS)*, 2018.

Received January 2020; revised February 2020; accepted March 2020