The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists

Umar Iqbal
The University of Iowa

Zubair Shafiq
The University of Iowa

Zhiyun Qian
University of California-Riverside

ABSTRACT

The increasing popularity of adblockers has prompted online publishers to retaliate against adblock users by deploying anti-adblock scripts, which detect adblock users and bar them from accessing content unless they disable their adblocker. To circumvent anti-adblockers, adblockers rely on manually curated anti-adblock filter lists for removing anti-adblock scripts. Anti-adblock filter lists currently rely on informal crowdsourced feedback from users to add/remove filter list rules. In this paper, we present the first comprehensive study of anti-adblock filter lists to analyze their effectiveness against anti-adblockers. Specifically, we compare and contrast the evolution of two popular anti-adblock filter lists. We show that these filter lists are implemented very differently even though they currently have a comparable number of filter list rules. We then use the Internet Archive's Wayback Machine to conduct a retrospective coverage analysis of these filter lists on Alexa top-5K websites over the span of last five years. We find that the coverage of these filter lists has considerably improved since 2014 and they detect anti-adblockers on about 9% of Alexa top-5K websites. To improve filter list coverage and speedup addition of new filter rules, we also design and implement a machine learning based method to automatically detect anti-adblock scripts using static JavaScript code analysis.

CCS CONCEPTS

• **Security and privacy → Browser security**; **Web application security**;

KEYWORDS

Adblocking, Anti-Adblocking, JavaScript, Machine Learning, Privacy, Static Code Analysis, The Wayback Machine

ACM Reference Format:
Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. 2017. The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists. In *Proceedings of IMC '17, London, United Kingdom, November 1–3, 2017,* 13 pages.
https://doi.org/10.1145/3131365.3131387

IMC '17, November 1–3, 2017, London, United Kingdom

ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131387

1 INTRODUCTION

Millions of users around the world use adblockers on desktop and mobile devices [31]. Users employ adblockers to get rid of intrusive and malicious ads as well as improve page load performance and protect their privacy. Since online publishers primarily rely on ads to monetize their services, they cannot monetize their services if a user employs an adblocker to remove ads. Online publishers have lost billions of dollars in advertising revenues due to adblocking [56]. Online publishers use two strategies to recoup lost advertising revenues.

First, many online publishers and advertisers have become part of the acceptable ads program [1], which allows their ads to be whitelisted if they conform to the acceptable ads guidelines. Small- and medium-sized publishers can enroll in the acceptable ads program for free, however, large publishers need to pay about 30% of the additional revenue recouped by whitelisting of acceptable ads. Popular adblockers, such as Adblock Plus, use the acceptable ads filter list to whitelist acceptable ads. While some stakeholders in the advertising ecosystem think that the acceptable ads program offers a reasonable compromise for users and publishers, there are lingering concerns about the acceptable ads criteria and the transparency of the whitelisting process [66].

Second, many online publishers have started to interrupt adblock users by employing anti-adblock scripts. These anti-adblock scripts allow publishers to detect adblock users and respond by showing notifications that ask users to disable their adblocker altogether, whitelist the website, or make a donation to support them. Most publishers rely on third-party anti-adblock scripts provided by vendors such as PageFair and Outbrain as well as community scripts provided by the IAB and BlockAdblock.

Adblockers employ anti-adblock filter lists to remove anti-adblock scripts and notifications by anti-adblockers. Similar to the filter lists such as EasyList and EasyPrivacy for blocking ads and trackers respectively, there are several crowdsourced anti-adblock filter lists that are used by adblockers to circumvent anti-adblokcers. These anti-adblock filter lists represent the state-of-the-art solution to anti-adblockers for adblock users, but little is known about the origin, evolution, and effectiveness of anti-adblock filter lists in circumventing anti-adblockers.

To fill this gap, in this paper we present the first comprehensive study of anti-adblock filter lists. We analyze the complete history of crowdsourced anti-adblock filter lists and conduct a retrospective measurement study to analyze their coverage over time. We also develop a lightweight machine learning based approach to automatically detect anti-adblock scripts using static JavaScript code analysis. Our approach can complement crowdsourcing to speed up the creation of new filter rules as well as improve the overall coverage of filter lists.

We address three research questions in this paper.

(1) *How have the filter rules in anti-adblock filter lists evolved over time?* We analyze two anti-adblock filter lists, Anti-Adblock Killer List and Combined EasyList (Adblock Warning Removal List + Anti-Adblock sections in EasyList). Our analysis reveals that they are implemented differently. More specifically, the Combined EasyList uses a few broadly defined filter rules and then uses many more exception rules to take care of false positives. In contrast, the Anti-Adblock Killer List tends to contain high precision filter rules that target specific websites. We also find that while the Combined EasyList is updated almost daily, the Anti-Adblock Killer List is being updated approximately once every month for the last one year.
(2) *How has the coverage of anti-adblock filter lists evolved over time?* We test different versions of anti-adblock filter lists on historical snapshots of Alexa top-5K websites archived by the Wayback Machine to study their coverage. We find that the Anti-Adblock Killer List triggers on 8.7% websites and the Combined EasyList only triggers on 0.4% websites currently. We further test both anti-adblock filter lists on Alexa top-100K live websites. We find that the Anti-Adblock Killer List triggers on 5.0% websites and the Combined EasyList only triggers on 0.2% websites. While the Anti-Adblock Killer List clearly seems to provide better coverage than the Combined EasyList, it has not been updated by its authors since November 2016.
(3) *How can we improve creation of anti-adblock filter lists?* To aid filter list authors in maintaining anti-adblock filter lists, we investigate a machine learning based automated approach to identify anti-adblock scripts. Our method conducts static analysis of JavaScript code to fingerprint syntactic features of anti-adblock scripts. The evaluation shows that our method achieves up to 99.7% detection rate and 3.2% false positive rate. Our proposed machine learning based automated approach can aid filter list authors to more quickly update anti-adblock filter lists as well as improve its coverage.

Our work is motivated by recent studies of different filter lists [44, 58, 66] that are used by adblockers. Our findings highlight inherent limitations of manual curation and informal crowdsouricng to maintain anti-adblock filter lists. Our work can help filter list authors to automatically and quickly update anti-adblock filter lists.

Paper Organization. The rest of the paper is organized as follows. §2 provides a brief background and discussion of related work on anti-adblocking. §3 discusses the evolution of anti-adblock filter lists. §4 presents our methodology to crawl historical snapshots of Alexa top-5K websites from the Wayback Machine and our retrospective coverage analysis of anti-adblock filter lists. §5 discusses our machine learning based approach to automatically detect anti-adblock scripts using static analysis. Finally, we conclude in §6 with an outlook to future research directions.

2 BACKGROUND & RELATED WORK

2.1 Background

Online publishers, advertisers, and data brokers track users across the Web to serve them personalized content and targeted advertisements. Online tracking is conducted using cookies, beacons, and fingerprinting [44]. Online tracking has raised serious privacy and surveillance concerns. Web tracking allows advertisers to infer sensitive information about users such as their medical and financial conditions [57]. Nation states can piggyback on web tracking information to conduct mass surveillance [45]. To combat privacy and surveillance concerns, one solution is to block trackers and the other is to block advertisements.

Tracker blockers remove network requests to known tracker domains. Tracker blocking extensions are available for all major web browsers. Ghostery [22] is one of the most popular tracker blocking extensions. It is used by more than 2.6 million Google Chrome users [23] and 1.3 million Mozilla Firefox users [24]. Another popular tracker blocking extension is the EFF's Privacy Badger [32], which is used by more than 532K Google Chrome users [33] and 116K Mozilla Firefox users [34]. Mainstream browsers such as Apple Safari and Mozilla Firefox have developed built-in tracking prevention solutions. Apple has recently launched Intelligent Tracking Prevention [67] in Safari to mitigate excessive tracking. Mozilla also has a tracking prevention solution [28] in the private browsing mode of Firefox.

Adblocking extensions can remove both advertisements and trackers. Like tracker blockers, adblocking extensions are also available for all major web browsers. Two of the popular adblockers are Adblock Plus and AdBlock. Adblock Plus [4] is used by more than 10 million Google Chrome users [5] and 19 million Mozilla Firefox users [6]. AdBlock [2] is used by more than 10 million Google Chrome users [3]. Some new web browsers such as Cliqz [13] and Brave [12] now have built-in adblockers. Google Chrome has also recently announced that they will block ads [63] on websites that do not comply with the Better Ads Standards set by the Coalition for Better Ads [14].

Adblockers have become much more popular than tracker blockers because they provide benefits such as removal of intrusive ads, protection from malware, and improved page load time in addition to privacy protection against online tracking. A recent survey [31] showed that 43% users install adblockers to get rid of interruptive or too many ads, 30% to avoid spread of malware, 16% to boost the page load time, and 6% to protect their privacy.

Adblockers rely on crowdsourced filter lists to block advertisements and trackers. EasyList [16] is the most widely used filter list to block advertisements. There are also some language-specific filter lists to block advertisements such as EasyList Dutch [18], EasyList Germany [18], and EasyList Spanish [18]. EasyPrivacy [18] is the most widely used filter list to block trackers. Other tracker blocking filter lists include Fanboy's Enhanced Tracking List [19], Disconnect.me [15], Blockzilla [11], and NoTrack Blocklist [30]. Adblockers such as Adblock Plus and AdBlock, as well as web browsers such as Cliqz and Brave, are subscribed to EasyList. Adblockers typically allow users to subscribe to different filter lists and incorporate custom filter rules.

Filter list rules are regular expressions that match HTTP requests and HTML elements. Adblockers block HTTP requests and hide HTML elements if they match any of the filter rules. Below we provide a brief overview of the syntax of HTTP request filter rules and HTML element filter rules [20].

HTTP Request Filter Rules: HTTP request rules match against URLs in the HTTP request headers. As shown in Code 1, these rules may be domain specific with a domain anchor (||) or a domain tag (**domain=**). Rule 1 blocks HTTP requests to example1.com. Rule 2 blocks HTTP requests to example1.com to load JavaScript. Rule 3 blocks HTTP requests to example1.com to load JavaScript on example2.com. Rule 4 blocks HTTP requests to download example.js JavaScript on example2.com.

```
! Rule 1
||example1.com
! Rule 2
||example1.com$script
! Rule 3
||example1.com$script,domain=example2.com
! Rule 4
/example.js$scipt,domain=example2.com
```

Code 1: HTTP request filter rules.

HTML Element Filter Rules: HTML element rules match against HTML elements loaded in a web page and hide the matched elements. Code 2 shows three examples of HTML element filter rules. Rule 1 hides the HTML element with ID examplebanner on example.com. Rule 2 hides the HTML element with class name examplebanner on example.com. Rule 3 hides the HTML element with ID examplebanner on any website.

```
!Rule 1
example.com###examplebanner
!Rule 2
example.com##.examplebanner
!Rule 3
###examplebanner
```

Code 2: HTML element filter rules.

Exception Rules: Exception rules override filter rules by allowing HTTP requests and HTML elements that match other filter rules. Exception rules are generally used to protect against false positives by filter rules that cause site breakage. Code 3 shows two examples of exception rules. Rule 1 allows HTTP requests to example.com to load JavaScript. Rule 2 allows HTML element with ID examplebanner on example.com.

```
! Rule 1
@@||example.com$script
! Rule 2
example.com#@##elementbanner
```

Code 3: Exception rules for HTTP requests and HTML elements.

2.2 Related Work

Online Tracking. Prior research has demonstrated the widespread nature of online tracking. Krishnamurthy and Wills [51] showed that top 10 third party trackers had grown from 40% in October 2005 to 70% in September 2008 for 1200 popular websites. In another study, Ihm et al. [49] reported that the popularity of search engines (google.com or baidu.com) and analytics sites (google-analytics.com) had increased from 2006 to 2010. Lerner et al. [53] conducted a retrospective study of online tracking on top-500 websites using the Internet Archive's Wayback Machine [38]. They reported that the percentage of websites contacting at least 5 separate third parties has increased from 5% in early 2000s to 40% in 2016. They also reported that the coverage of top trackers on the web is increasing rapidly, with top 5 trackers now covering more than 60% of the top 500 websites as compared to less than 30% ten years ago. Englehardt and Narayanan [44] showed that a handful of third parties including Google, Facebook, Twitter, Amazon, and AdNexus track users across a significant fraction of the Alexa top one million websites. For example, they reported that Google alone tracks users across more than 80% of Alexa top one million websites.

Tracker Blocking. Tracker blocking tools have had mixed success in blocking online trackers. Roesner et al. [64] found that defenses such as third-party cookie blocking, Do Not Track (DNT), and popup blocking are not effective in blocking social media widgets tracking. Englehardt and Narayanan [44] demonstrated that existing tracker blocking tools like Ghostery, Disconnect, EasyList, and EasyPrivacy are less effective against obscure trackers. Merzdovnik et al. [58] found that popular tracker blocking tools have blind spots against stateless trackers and trackers with smaller footprints. To improve tracker blockers, Gugelmann et al. [48] proposed an automated approach that learns properties of advertisements and analytics services on existing filter lists and identifies new services to be included in adblockers' filter lists. Yu et al. [69] proposed an approach, inspired by k-anonymity, to automatically filter data sent to trackers that have the potential to uniquely identify a user. Their approach aims to block third-party tracker requests while avoiding blanket blocking of 3rd parties based on blacklists. Ikram et al. [50] proposed a one-class machine learning approach to detect trackers with high accuracy based on their JavaScript code structure. In the same spirit as prior research on improving tracker blocking tools, in this paper we propose a machine learning approach to detect anti-adblock filter lists. As we discuss later, our approach is customized to capture syntactic behaviors of anti-adblockers.

Adblocking and Anti-Adblocking. Prior research has focused on analyzing the prevalence of adblockers. Pujol et al. [61] analyzed network traces from a major European ISP and found that 22% of the most active users used Adblock Plus. Malloy et al. [55] conducted a measurement study of 2 million users and found that 18% of users in the U.S. employ adblockers. The increasing popularity of adblockers has pushed online publishers to retaliate against adblock users.

First, some online publishers have started to manipulate ad delivery to evade filter lists. For example, publishers can keep changing third-party domains that serve ads to bypass filter list rules. More recently, Facebook manipulated HTML element identifiers [42] to bypass their ads through filter lists. To address this problem, Storey

et al. [65] proposed a perceptual adblocking method for visually identifying and blocking ads based on optical character recognition and fuzzy image matching techniques. The key idea behind perceptual adblocking is that ads are distinguishable from organic content due to government regulations (e.g., FTC [36]) and industry self-regulations (e.g., AdChoices [41]). The authors [65] reported that their perceptual adblocking approach is robust to ad delivery manipulation by online publishers.

Second, some online publishers have started to employ anti-adblock scripts to detect adblockers. Anti-adblock scripts detect adblock users and prompt them to disable their adblocker. Adblockers currently rely on manually curated anti-adblock filter lists, e.g., Anti-Adblock Killer List, to block anti-adblock scripts. Rafique et al. [62] manually analyzed top 1,000 free live streaming websites and reported that 16.3% websites attempted to detect adblockers. Nithyanand et al. [60] crawled Alexa top-5K websites and manually analyzed JavaScript snippets to find that 6.7% websites used anti-adblocking scripts. In our prior work, Mughees et al. [59] used a machine learning based approach with A/B testing to analyze changes in page content due to anti-adblockers. We found that only 686 out of Alexa-100K websites visibly react to adblockers. In contrast to prior efforts to study anti-adblock deployment, in this paper we conduct a retrospective measurement study of anti-adblock filter lists that are currently used by adblockers to circumvent anti-adblockers. In our prior work [59] we used machine learning to detect anti-adblockers that visibly react to adblockers, while in this paper we use machine learning to fingerprint anti-adblock scripts. Storey et al. [65] proposed stealth (hide adblocking) and active (actively counter adblock detection) adblocking approaches. For stealth adblocking, they partially implemented a rootkit-style approach that intercepts and modifies JavaScript API calls that are used by publishers to check the presence of ad elements. Their approach is complementary to our approach. For active adblocking, they implemented a signature-based approach to remove anti-adblock scripts using manually crafted regular expressions. In contrast, our proposed machine learning based approach can automatically identify anti-adblock scripts based on their syntactic features.

Next, we analyze the evolution of popular anti-adblock filter lists (§3), measure their historic coverage on popular websites (§4), and develop machine learning based approach to detect anti-adblock scripts (§5).

3 ANALYZING ANTI-ADBLOCK FILTER LISTS

We first introduce anti-adblockers and then analyze popular anti-adblock filter lists.

3.1 How Anti-Adblocking Works?

Anti-adblockers employ baits to detect adblockers. These baits are designed and inserted in web pages such that adblockers will attempt to block them. To detect the presence of adblockers, anti-adblockers check whether these baits are blocked. Anti-adblockers use HTTP and HTML baits to detect adblockers. Below we discuss both of them separately.

For HTTP baits, anti-adblockers check whether the bait HTTP request is blocked by adblockers. Code 4 illustrates the use of HTTP bait by the anti-adblocker on businessinsider.com, which requests a bait URL www.npttech.com/advertising.js and checks whether it is successfully retrieved. Code 4 dynamically creates an HTTP request bait and calls `onLoad` event in case of success and `onError` event in case of failure. Both events call the `setAdblockerCookie` function with a parameter of either true or false. `setAdblockerCookie` event sets the value of the cookie `__adblocker` to either true or false depending on the input value.

```
var script = document.createElement("script")
    ;
script.setAttribute("async", true);
script.setAttribute("src", "//www.npttech.com
    /advertising.js");
script.setAttribute("onerror", "
    setAdblockerCookie(true);");
script.setAttribute("onload", "
    setAdblockerCookie(false);");
document.getElementsByTagName("head")[0].
    appendChild(script);

var setAdblockerCookie = function(adblocker)
    {
    var d = new Date();
    d.setTime(d.getTime() + 60 * 60 * 24 * 30
        * 1000);
    document.cookie = "__adblocker=" + (
        adblocker ? "true" : "false") + ";
        expires=" + d.toUTCString() + ";
        path=/";
}
```

Code 4: Anti-adblocker JavaScript code for HTTP bait.

For HTML baits, anti-adblockers check if the CSS properties of the bait HTML element are modified. Code 5 illustrates the use of HTML bait by a popular third-party anti-adblocker called BlockAdBlock [10], which creates a div bait and checks whether the bait is removed. The `_creatBait` function creates a div element and sets its CSS properties. The `_checkBait` function checks whether the div element's CSS properties such as `offsetHeight`, `offsetTop`, and `offsetWidth` are changed.

```
BlockAdBlock.prototype._creatBait = function
    () {
  var bait = document.createElement('div');
  bait.setAttribute('class',
      this._options.baitClass);
  bait.setAttribute('style',
      this._options.baitStyle);
  this._var.bait =
      window.document.body.appendChild(bait);
  this._var.bait.offsetParent;
  this._var.bait.offsetHeight;
  this._var.bait.offsetLeft;
  this._var.bait.offsetTop;
  this._var.bait.offsetWidth;
  this._var.bait.clientHeight;
  this._var.bait.clientWidth;
  if (this._options.debug === true) {
   this._log('_creatBait', 'Bait has been
       created');
  }
```

```
};

BlockAdBlock.prototype._checkBait = function(loop) {
    var detected = false;
    if(window.document.body.getAttribute('abp') !== null
    || this._var.bait.offsetParent === null
    || this._var.bait.offsetHeight == 0
    || this._var.bait.offsetLeft == 0
    || this._var.bait.offsetTop == 0
    || this._var.bait.offsetWidth == 0
    || this._var.bait.clientHeight == 0
    || this._var.bait.clientWidth == 0) {
      detected = true;
    }
};
```

Code 5: BlockAdBlock JavaScript code for creating and checking a bait.

3.2 Anti-Adblock Filter Lists

Using the aforementioned techniques, anti-adblockers detect adblockers and prompt users to disable their adblockers if they want to view page content. To circumvent anti-adblockers, adblockers currently rely on anti-adblock filter lists. The rules of these filter lists are designed to handle HTTP requests and HTML elements that are used by anti-adblockers. For example, the filter list rules may allow or block HTTP requests and HTML elements to avoid detection by anti-adblockers. Code 6 shows two examples of anti-adblock rules. Rule 1 blocks third-party HTTP requests to pagefair.com which is a well-known anti-adblock vendor. Rule 2 hides the HTML element with ID `noticeMain` which displays an anti-adblock notice on smashboards.com.

```
! Rule 1
||pagefair.com^$third-party
! Rule 2
smashboards.com###noticeMain
```

Code 6: Example anti-adblock filter rules.

The most widely used anti-adblock filter lists are: (1) Anti-Adblock Killer List [8], (2) Adblock Warning Removal List [37], and (3) EasyList [16]. The first two are dedicated to target anti-adblockers, however, EasyList's main purpose is to block ads. Several sections in EasyList specifically contain anti-adblocking filter rules. In this paper, we only analyze anti-adblock sections of EasyList. Anti-Adblock Killer List started in 2014, Adblock Warning Removal List started in 2013, and anti-adblock sections in EasyList were created in 2011. These anti-adblock filter lists rely on informal crowdsourced input from their users (e.g., via feedback forums) to add new filter rules or remove/modify old filter rules. These anti-adblock filter lists have been regularly updated since their creation.

Anti-Adblock Killer. Anti-Adblock Killer List was created by "reek" in 2014. The list is maintained on GitHub [8] and users submit feedback by reporting issues on the GitHub page [9]. On average, the list adds or modifies 6.2 filter rules for every revision. Figure 1(a) visualizes the temporal evolution of different types of filter

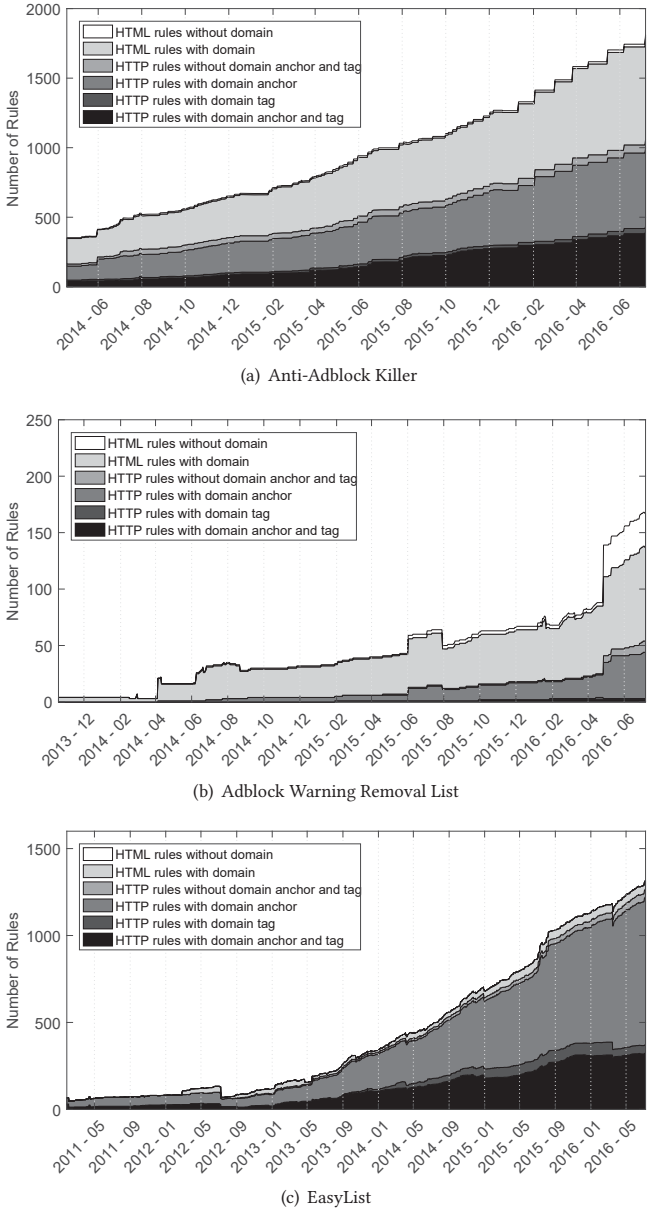


Figure 1: Temporal evolution of anti-adblock filter lists.

rules in the Anti-Adblock Killer List. We observe a steady increase in number of filter rules. The filter list started with 353 initial filter rules and it has expanded to 1,811 filter rules by July 2016. The stair step pattern in the number of filter rules starting November 2015 indicates that the update cycle of the filter list increased to approximately once a month. For this span, the list adds or modifies 60 filter rules for every revision on average. The most common types of filter rules are HTTP request rules (with domain anchor and both domain anchor and tag) and HTML element rules (with domain). The most recent version of the filter list has 58.5% HTTP request rules and 41.5% HTML element rules. 31.0% filter rules are

HTTP request rules with only domain anchor, 2.1% are HTTP request rules with only domain tag, 22.0% are HTTP request rules with both domain anchor and tag, and 3.4% are HTTP request rules without domain anchor and tag. 40.0% filter rules are HTML element rules with domain and 1.5% are HTML element rules without domain.

Adblock Warning Removal List. Adblock Warning Removal List was created by the EasyList filter list project [16] in 2013. The list is maintained by multiple authors and relies on user feedback on their forum [17]. On average, the list adds or modifies 0.2 filter rules every day. Figure 1(b) visualizes the temporal evolution of different types of filter rules in the Adblock Warning Removal List. In contrast to the Anti-Adblock Killer List, it is noteworthy that this list contains a larger fraction of HTML filter rules. The HTML element filter rules hide anti-adblock warning popups that are displayed when anti-adblockers detect adblockers. The filter list started in 2013 with 4 filter rules and it has expanded to 167 filter rules by July 2016. While the filter list initially grows slowly, we note a significant spike in the number of filter rules in April 2016 after which the rate of addition of new rules increases as well. The spike observed in April 2016 corresponds to the addition of French language section in the filter list. The most recent version of the filter list has 32.3% HTTP request rules and 67.7% HTML element rules. 24.5% filter rules are HTTP request rules with only domain anchor, 0.6% are HTTP request rules with only domain tag, 1.2% are HTTP request rules with both domain anchor and tag, and 6.0% are HTTP request rules without domain anchor and tag. 49.7% filter rules are HTML element rules with domain and 18.0% are HTML element rules without domain.

EasyList. EasyList's primary purpose is to block ads. All major adblockers are subscribed to EasyList by default. As discussed earlier, several sections in EasyList specifically contain anti-adblocking filter rules. EasyList started adding anti-adblock rules in 2011. Our analysis here focuses only on the anti-adblock sections of EasyList. On average, the list adds or modifies 0.6 anti-adblock filter rules every day. Figure 1(c) visualizes the temporal evolution of different types of anti-adblock filter rules in EasyList. The filter list started with 67 filter rules in 2011 and it has expanded to 1,317 filter rules by July 2016. We observe a steady increase in the number of HTTP request filter rules. It is noteworthy that the filter list contains a relatively small fraction of HTML element filter rules. The most recent version of the filter list has 96.3% HTTP request rules and 3.7% HTML element rules. 64.6% filter rules are HTTP request rules with only domain anchor, 3.6% are HTTP request rules with only domain tag, 24.6% are HTTP request rules with both domain anchor and tag, and 3.5% are HTTP request rules without domain anchor and tag. 3.7% filter rules are HTML element rules with domain. The filter list does not contain any HTML element rule without domain.

3.3 Comparative Analysis of Anti-Adblock Lists

Next, we compare and contrast different anti-adblocking filter lists. To this end, we decide to combine EasyList and Adblock Warning Removal List because (1) they are both managed by the EasyList filter list project [16] and (2) they are complementary — Adblock Warning Removal List mostly contains HTML element filter rules and EasyList mostly contains HTTP request filter rules. The most recent version of the Combined EasyList contains 1,483 rules.

Alexa Rank	Anti-Adblock Killer List	Combined EasyList
1-5K	112	124
5K-10K	49	69
10K-100K	280	312
100K-1M	334	359
>1M	640	530

Table 1: Distribution of domains in filter lists across Alexa rankings.

Since a vast majority of filter list rules contain domain information, we first compare the number of domains in both filter lists. Anti-Adblock Killer List and Combined EasyList include 1,415 and 1,394 domains, respectively. To our surprise, these filter lists have only 282 domains in common. To analyze why the filter lists are targeting different sets of domains, we break down the set of domains based on the Alexa popularity ranks and their category. Table 1 provides the breakdown of domains in both filter lists based on their Alexa ranks. While there are some differences, popularity distributions of domains in both filter lists are fairly similar. For domain categorization, we use McAfee's URL categorization service [26] and manually merge similar categories together. Figure 2 shows the distribution of domains in both filter lists based on McAfee's URL categorization. We plot top 15 categories and group remaining categories as others. Similar to the distribution of Alexa popularity ranks, the categorization trend is also similar across both filter lists. Below, we further investigate the differences in the filter lists by analyzing whether they are implemented differently.

To analyze whether these filter lists are implemented differently, we study exception and non-exception conditions in filter rules. We label the domains as exception or non-exception on the basis of rules in which they appeared. If the rule is an exception rule, we label the domain as exception domain. If the rule is a non-exception rule, we label the domain as non-exception domain. For the Combined

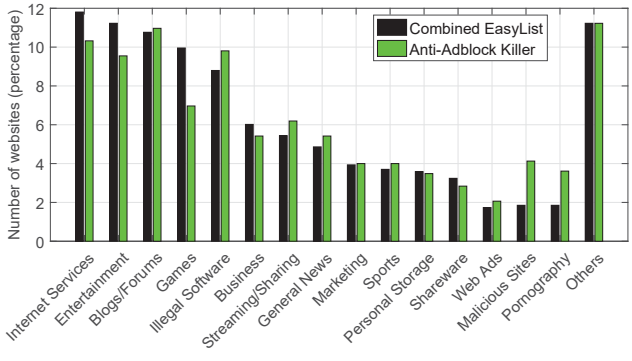


Figure 2: Categorization of domains in anti-adblock filter lists.

EasyList, we note a ratio of approximately 4:1 for exception to non-exception domains. For the Anti-Adblock Killer List, we note a ratio of approximately 1:1 for exception to non-exception domains.

We next investigate the difference in proportion of exception and non-exception domains in both filter lists. We surmise that the Combined EasyList has many exception domains because it works with a large number of adblock rules in the full EasyList. For example, rule 1 in Code 7 is an adblocking rule that blocks every URL that ends with `/ads.js?` and rule 2 is an exception rule that overrides rule 1 and allows `/ads.js?` on numerama.com. Our analysis revealed that numerama.com was using /numerama.com/ads.js as a bait HTTP request. More specifically, in Code 8, we note that `canRunAds` will be undefined if bait HTTP request is blocked and `adblockStatus` will be set to `active`. Combined EasyList has many such exception rules. In contrast, we note that the Anti-Adblock Killer List has a larger fraction of non-exception filter rules that block anti-adblock scripts on specific domains. Our results highlight that these anti-adblock filter lists have different approaches to writing filter rules.

```
! Rule 1
/ads.js?
! Rule 2
@@||numerama.com/ads.js
```

Code 7: URL that is blocked on other websites but allowed on numerama.com.

```
canRunAds = true;
var adblockStatus = 'inactive';
if (window.canRunAds === undefined) {
  adblockStatus = 'active';
}
```

Code 8: Use of HTTP bait by numerama.com.

We next analyze the implementation of filter rules for 282 overlapping domains in the Combined EasyList and Anti-Adblock Killer List. Domain specific rules are implemented through domain tag (**domain**=), domain anchor (||), and HTML element tag (##). Code 9 shows the filter rule implementation by both filter lists for yocast.tv using HTML element tag. The Combined EasyList hides HTML element with ID `ra9e` on yocast.tv. Whereas the Anti-Adblock Killer List hides HTML element with ID `notice` on yocast.tv. Code 10 shows the filter rule implementation by both filter lists for pagefair.com using domain anchor. The Combined EasyList blocks all HTTP requests from pagefair.com by its general adblocking rules and focuses on specific websites in its anti-adblocking rules. Whereas the Anti-Adblock Killer List blocks all HTTP requests from pagefair.com on any domain. Overall, we note that both filter lists often have different rules to circumvent anti-adblockers even for the same set of domains.

```
! Combined EasyList
yocast.tv###ra9e
! Anti-Adblock Killer list
yocast.tv###notice
```

Code 9: Implementation of yocast.tv for the Combined EasyList and Anti-Adblock Killer List.

```
! Combined EasyList
||pagefair.com/static/adblock_detection/js/
    d.min.js$domain=majorleaguegaming.com
! Anti-Adblock Killer list
||pagefair.com^$third-party
```

Code 10: Implementation of pagefair.com for the Combined EasyList and Anti-Adblock Killer List.

Finally, we investigate which of these anti-adblock filter lists is more prompt in adding new filter rules. To this end, we compare and contrast the addition time of 282 overlapping domains in both filter lists. We find that 185 domains appear first in the Combined EasyList, 92 domains appear first in the Anti-Adblock Killer List, and 5 domains appear on the same day on both filter lists. Figure 3 plots the distribution of overlapping domains in terms of the difference in days when they appear in each list. These results seem to indicate that the Combined EasyList is more prompt in adding new rules. The difference can be explained in part because the Combined EasyList is used by default in many popular anti-adblockers, thus we expect it to get more user feedback from its much larger user base than that for the Anti-Adblock Killer List. We also note as a caveat that we should expect many domains to appear in the Combined EasyList before the Anti-Adblock Killer List because the Combined EasyList started more than two years before the Anti-Adblock Killer List.

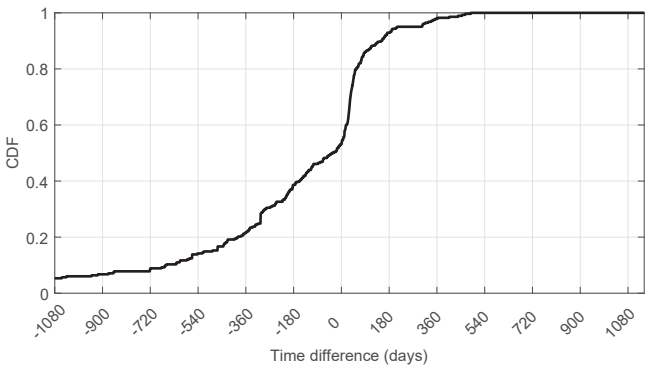


Figure 3: Distribution of time difference (number of days) between the Combined EasyList and Anti-Adblock Killer List for addition of rules that target overlapping domains.

We have identified several key differences in the filter rule implementation across two popular anti-adblock filter lists. However, our comparative analysis of filter lists provides an incomplete picture of their behavior. First, the behavior of individual filter rules is dependent on other rules in the filter list. Second, the effectiveness of these filter lists is dependent on complex and often dynamic website behavior. Therefore, to fully understand and compare the behavior of the Combined EasyList and Anti-Adblock Killer List, we need to run them on actual websites and see how different filter rules are triggered.

4 ANALYZING FILTER LIST COVERAGE USING THE WAYBACK MACHINE

We now conduct a retrospective study to analyze how anti-adblock filter lists trigger on popular websites. Our goal is to study the evolution of anti-adblock prevalence. To collect historical snapshots of popular websites, we rely on the Internet Archive's Wayback Machine [38] which has archived historic snapshots of 279 billion web pages since 1996 [40].

We crawl Alexa top-5K websites for the last five years from the Wayback Machine and check for the presence of anti-adblockers using the Combined EasyList and Anti-Adblock Killer List. Figure 4 provides an overview of our measurement methodology. We first identify domains that are not archived by the Wayback Machine. We then request the Wayback Availability JSON API to collect the URLs for the monthly archives of websites. We remove outdated URLs for which the Wayback Machine does not have a snapshot close to our requested date. We automatically crawl the remaining URLs and store their request/responses in the HTTP Archive (or HAR) format. We then remove partial HAR files and use the remaining to match against anti-adblock filter lists. Below we explain our methodology in more detail.

4.1 Crawling the Wayback Machine

The Wayback Machine archives snapshots of popular websites several times everyday. While popular websites frequently change their content, they much less frequently change their codebase and dependance on third-party scripts such as anti-adblocking scripts. This allows us to reduce the data set that we need to crawl. We decide to only crawl one snapshot per month for a website. We attempt to crawl a total of approximately 300K URLs from the Wayback Machine. A serial crawler implementation would take several months to completely crawl all URLs. To speed up crawling, we parallelize crawlers using 10 independent browser instances.

For each of the Alexa top-5K domains, we first check whether the domain is archived by the Wayback Machine. The Wayback Machine may decide to not archive a domain due to that domain's robots.txt exclusion policy, at the request of the domain's administrator, or for undefined reasons.[1] The Wayback Machine does not archive 153 domains because of their robots.txt exclusion policy, 26 domains because of domain's administrator request, and 54 domains because of undefined reasons.

For the remaining domains, we request the Wayback Availability JSON API [39] to collect the URLs for the monthly archives of the domains' homepages. The Wayback Availability JSON API returns a URL that is closest to the requested date. If a website is not archived by the Wayback Machine, an empty JSON response is returned by the Wayback Availability JSON API. If a URL is returned we check its timestamp and discard URLs for which the time difference between the requested date and the actual date of URL is more than 6 months.

We open the remaining URLs in a fully functional web browser using Selenium WebDriver [35] and collect all HTTP requests and responses. We configure a profile of Mozilla Firefox [27] browser with Firebug [21] and NetExport [29] plugins. For each website we visit, we store all HTTP requests/responses in a HAR file [25] and the page content in a HTML file. We get multiple HAR files for some websites that keep on refreshing. For such websites, we take a union of all HTTP requests in HAR files. We get incomplete HAR files for websites that are not completely archived by the Wayback Machine. We discard the partial HAR files, whose size is less than 10% of the average size of HAR files over a year.

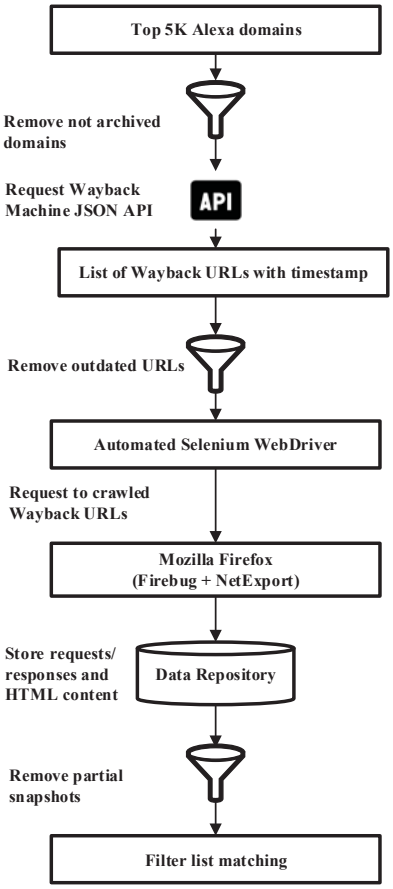


Figure 4: Overview of our measurement methodology to crawl historical snapshots from the Wayback Machine and match against anti-adblock filter lists.

Figure 5 shows the timeseries of number of missing monthly snapshots by the Wayback Machine. We note that the number of missing snapshots has decreased from 1,524 in August 2011 to 984 in July 2016. Outdated URLs account for the highest proportion of missing snapshots. The number of outdated URLs has decreased from 1,239 in August 2011 to 532 in July 2016. However, the number of not archived URLs has increased from 262 in August 2011 to 374 in July 2016. Our analysis of not archived URLs shows that it is generally due to HTTP 3XX redirects. For 3XX redirects, the Wayback Availability JSON API returns an empty JSON object. The number of partial snapshots has also increased from 23 in

[1]The Internet Archive project recently announced [47] a policy change to ignore robots.txt directives.

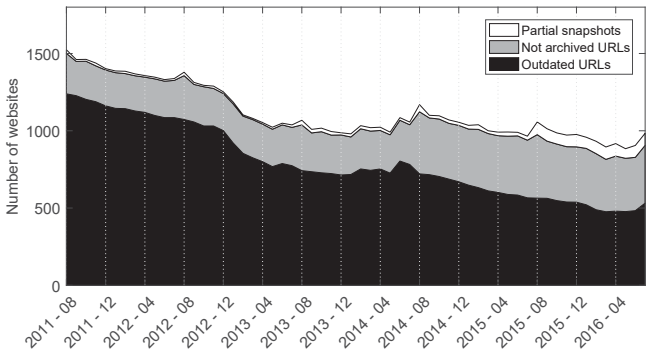


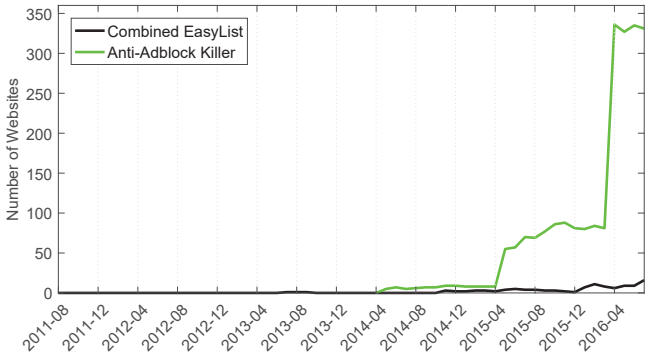
Figure 5: Number of websites over time excluded from analysis.

August 2011 to 78 in July 2016. Our analysis of partial snapshots shows that it is due to anti-abuse policies implemented by websites against bots. Some domains show an error page if they detect the Wayback Machine's crawling bots. For correctly archived websites, we verified that the Wayback Machine serves the same snapshots to a normal user (using a regular browser) and our crawlers. To this end, we manually analyzed a sample of randomly selected 20 URLs and did not find any difference in content served to a normal user and our crawler. Our analysis corroborates the findings of Lerner et al. [53] that the Wayback Machine archives most websites reliably. It is noteworthy that we miss location specific content because we rely on the content archived by the Wayback Machine. The archived content is specific to the location of the Wayback Machine's servers.

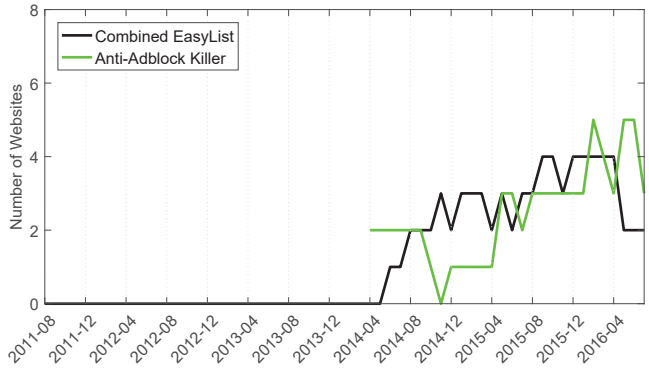
4.2 Anti-Adblock Detection on the Wayback Machine

After crawling the historic data of Alexa top-5K domains, we check for the presence of anti-adblockers using the Anti-Adblock Killer List and Combined EasyList. We use the historic versions of these filter lists at that point in time because it will portray the retrospective effectiveness of these filter lists. For detection with HTTP request filter rules, we extract the HTTP requests from the HAR files and see whether they trigger HTTP rules in the filter lists. For detection with HTML element filter rules, we open the stored HTML page in a web browser with adblocker enabled and see whether it triggers HTML rules in the filter lists.

To detect the presence of anti-adblockers with HTTP request filter rules, we extract HTTP request URLs from the HAR files crawled from the Wayback Machine. To archive a website, the Wayback Machine replaces all live URLs with its own reference by prepending http://web.archive.org to the URL. To run HTTP request filter rules against these URLs, we truncate the Wayback Machine references. Note that we do not truncate Wayback escape [52, 53] URLs because they are not archived with the Wayback Machine reference. We use `adblockparser` [7] to match all URLs of a website against HTTP request filter rules. We label a website as anti-adblocking if any of its URLs is matched against any of the HTTP request filter rules. Figure 6(a) shows the number of anti-adblocking websites in Alexa top-5K websites that are detected by



(a) Number of websites that trigger HTTP rules



(b) Number of websites that trigger HTML rules

Figure 6: Temporal evolution of websites that trigger HTTP and HTML filter rules of the Anti-Adblock Killer List and Combined EasyList.

HTTP request filter rules in Anti-Adblock Killer List and Combined EasyList. For the Anti-Adblock Killer List, the number of matched websites has increased from 0 in April 2014 to 331 in July 2016. For the Combined EasyList, the number of matched websites has increased from 0 in August 2011 to 16 in July 2016. In contrast to our comparative analysis of these filter lists in §3, it is noteworthy that the number of matched websites for the Anti-Adblock Killer List is much higher than that for the Combined EasyList. Our analysis of matched URLs also reveals that a vast majority of websites use third party scripts for anti-adblocking. More specifically, more than 98% of matched websites for the Anti-Adblock Killer List use scripts from third party anti-adblocking vendors such as Optimizel, Histats, PageFair, and Blockadblock.

To detect the presence of anti-adblockers with HTML element filter rules, we open the HTML webpage crawled from the Wayback Machine in a fully functional web browser and analyze whether it triggers any HTML element filter rules in the filter lists. We configure a profile of Mozilla Firefox browser with Adblock Plus enabled and subscribe to the Anti-Adblock Killer List and Combined EasyList. For each crawled HTML webpage, we open the webpage in the browser and wait 180 seconds for it to load completely and for HTML element filter rules to trigger. After that, we analyze the Adblock Plus logs and extract the triggered HTML element

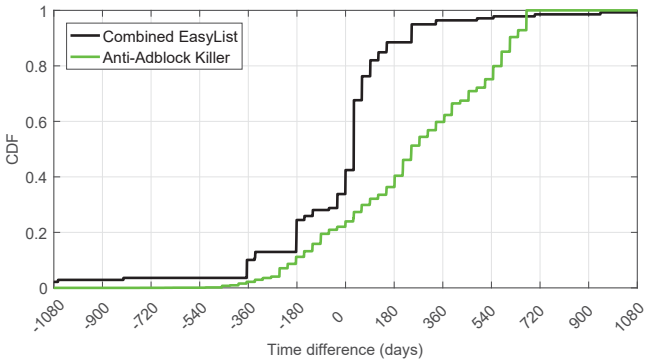


Figure 7: Distribution of time difference (number of days) between the day an anti-adblocker was added to a website and the day it was detected by Combined EasyList and Anti-Adblock Killer List.

filter rules for the crawled HTML webpages. Figure 6(b) shows the number of anti-adblocking websites in Alexa top-5K websites that are detected by HTML element filter rules in the Anti-Adblock Killer List and the Combined EasyList. We note that the number of matching HTML element rules is much less than that with HTTP request filter rules. For the Anti-Adblock Killer List, the number of matched websites remains between 0 and 5 from April 2014 to July 2016. For the Combined EasyList, the number of matched websites remains between 0 and 4 from August 2011 to July 2016.

Next, we compare the speed of the Anti-Adblock Killer List and the Combined EasyList in adding a new filter rule for an anti-adblocker after its addition. Figure 7 plots the distribution of time difference in terms of number of days when an anti-adblocker was added to a website and when a filter rule was added to detect it. The results indicate that the Combined EasyList is more prompt than the Anti-Adblock Killer List in adding new filter rules for anti-adblockers that are added to a website. In the Combined EasyList, filter rules are defined for 82% anti-adblockers within 90 days of their addition to a website. However, in the Anti-Adblock Killer List, filter rules are defined for only 32% anti-adblockers within 90 days of their addition to a website. It is noteworthy that for a fraction of anti-adblockers filter rules are present on the Combined EasyList and Anti-Adblock Killer List even before they are added to a website. This can happen when the filter list uses generic rules to block third-party anti-adblockers. In the Combined EasyList, filter rules are present for 42% anti-adblockers before their addition to a website. In the Anti-Adblock Killer List, filter rules are present for 23% anti-adblockers before their addition to a website.

4.3 Anti-Adblock Detection on the Live Web

After retrospectively analyzing the coverage of anti-adblock filter lists using the Wayback Machine, we next study their coverage on the live Web in April 2017. We crawl Alexa top-100K websites and check for the presence of anti-adblockers using the Anti-Adblock Killer List and Combined EasyList. For detection on live Web, we use the most recent version of filter lists. Overall, our results on the live Web corroborate the findings of our retrospective analysis using the Wayback Machine. For example, we observe that the coverage of the Anti-Adblock Killer List is much more than that of the Combined EasyList. For the Anti-Adblock Killer List, the number of websites that trigger HTTP request filter rules is 4,931 out of 99,396. For the Combined EasyList, the number of websites that trigger HTTP request filter rules is 182 out of 99,396. Furthermore, we find that the number of websites that trigger HTML element filter rules is much smaller. Specifically, the number of websites that trigger HTML element filter rules is 11 for the Anti-Adblock Killer List and 15 for the Combined EasyList. We again note that a vast majority of websites use third party anti-adblock scripts. For the Anti-Adblock Killer List, 97% of the matched websites use anti-adblocking scripts from third party vendors.

5 DETECTING ANTI-ADBLOCK SCRIPTS

Since anti-adblock filter lists are currently manually maintained, it is challenging for their authors to keep them up-to-date. The two popular anti-adblock filter lists are implemented differently and have varying coverage and update frequency. For example, the anti-adblock filter list with better coverage tends to be updated less frequently. We note that it is challenging for anti-adblock filter lists to keep pace with anti-adblockers that quickly evolve in response to adblockers [59]. Therefore, to help anti-adblock filter list authors, we next investigate a machine learning based automated approach to detect anti-adblock scripts.

Online publishers use JavaScript to implement client side anti-adblocking logic. We plan to fingerprint anti-adblocking JavaScript behavior by doing static code analysis. The basic idea is to extract syntactic features from anti-adblocking JavaScript code and build a light weight machine learning classifier. Our approach is inspired by prior machine learning based JavaScript analysis approaches to detect malware [43] and trackers [50].

Figure 8 shows the workflow of our proposed approach. We first unpack JavaScript files using the Chrome V8 engine. We construct Abstract Syntax Trees (ASTs) of the parsed scripts and then extract different syntactic features. We use supervised machine learning to train a classifier (AdaBoost + SVM) for distinguishing between anti-adblocking and non anti-adblocking scripts. Below we discuss different steps of our static JavaScript code analysis approach to detect anti-adblockers.

Unpacking Dynamic JavaScript. The dynamic nature of JavaScript makes it challenging to do static analysis. For example, JavaScript code is often packed using `eval()` function. Such code unpacks itself right before it is executed. To cate for dynamically generated code, we use Chrome V8 engine to unpack `eval()` function by intercepting calls to the `script.parsed` function. `script.parsed` function is invoked every time `eval()` is evaluated or new code is added with `<iframe>` or `<script>` tags.

Gathering Labeled Data. In order to train a machine learning classifier, we need labeled examples of anti-adblocking and non anti-adblocking scripts. We utilize more than one million JavaScript snippets that were collected as part of our retrospective measurement study of Alexa top-5K websites using the Wayback Machine. Our anti-adblock data set consists of JavaScript snippets that matched HTTP request filter rules of crowdsourced anti-adblock filter lists. We use 372 of these anti-adblocking scripts as positive examples. To collect negative examples, we use the remaining scripts that

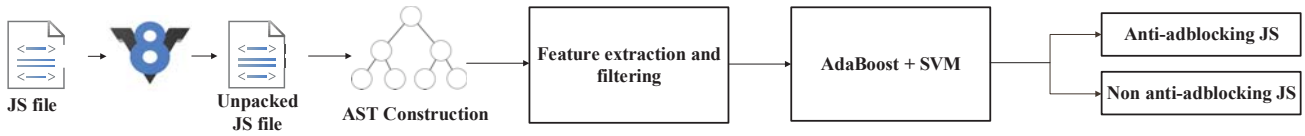


Figure 8: Overview of anti-adblock script detection approach.

the filter lists did not identify as anti-adblockers. We aim for a class imbalance of approximately 10:1 (negative:positive) in our labeled data. We manually verify a randomly selected 10% sample of ground truth for positive examples. We find that a vast majority of the scripts are served from known anti-adblocking vendors such as Optimizely and Blockadblock.

Feature Extraction. To extract features, we map the parsed scripts to ASTs, which are syntactic representations of JavaScript code in a tree format. After constructing the ASTs, we extract features according to the hierarchical tree structure. We define a feature as a combination of *context* and *text*. *Context* is the place where the feature appears, such as loop, try statement, catch statement, if condition, switch condition, etc. *Text* is code that appears in the context. We extract three types of feature sets based on different selection criterion of *text*. For the first type (**all**), we consider all text elements including JavaScript keywords, JavaScript Web API keywords, identifiers, and literals. For the second type (**literal**), we consider text elements only from literals, i.e., we remove JavaScript keywords, JavaScript Web API keywords, and identifiers. These features are very general because they do not contain identifiers and keyword specific text in JavaScript code. For the third type (**keyword**), we consider text elements only from native JavaScript keywords and JavaScript Web API keywords, i.e., we remove identifiers and literals. As we do not consider text elements from identifiers and literals, **keyword** features are not impacted by the randomization of identifers and literals. However, these features are susceptible to polymorphism. These three feature sets provide us varying levels of generalization. We expect more general features to be robust to minor implementation changes, however they may lose some useful information due to generalization.

Table 2 shows some extracted features and their types for Code 5. **Literal** features capture textual properties of JavaScript code such as `abp` and `0` in Table 2. **Keyword** features capture syntactic properties of JavaScript code such as `clientHeight` and `clientWidth` in Table 2. In contrast, **all** features can contain text such as `_checkBait` and `_creatBait` that represents names of variables, functions, etc.

Features	Types
MemberExpression:BlockAdBlock	**all**
MemberExpression:_creatBait	**all**
MemberExpression:_checkBait	**all**
Literal:abp	**all**, **literal**
Literal:0	**all**, **literal**
Literal:hidden	**all**, **literal**
Identifier:clientHeight	**all**, **keyword**
Identifier:clientWidth	**all**, **keyword**
Identifier:offsetHeight	**all**, **keyword**
Identifier:offsetWidth	**all**, **keyword**

Table 2: Some features extracted from BlockAdBlock JavaScript.

Feature Selection. For these three types of feature sets, we extract a total of 1,714,827, 1,211,029, and 16,620 distinct features. Note that each feature is binary, i.e., its value is 1 or 0 depending on whether the feature is present/absent in a script. We construct a vector space to map scripts. Scripts with similar features are placed close to each other as compared to scripts with dissimilar features. We construct such a vector space by defining the mapping function as:

$$\phi : x \rightarrow (\phi_s(x))_{s \in S}$$

$$\phi_s(x) = \begin{cases} 1 & \text{if } x \text{ contains the feature } s \\ 0 & \text{otherwise} \end{cases}$$

We map each script x to a vector space with the mapping function ϕ. The mapping function ϕ assign 1 for a feature s that is present in script x and 0 otherwise, and S is the set of all possible features. Each instance of anti-adblocking and non anti-adblocking class is mapped with a mapping function $\phi(x)$.

After constructing our feature space, we remove irrelevant features. First, we remove features that do not vary much. To this end, we compute variance of each feature and remove it if its variance is less than 0.01. After applying this filter we are left with 68,510, 32,226, and 6,171 features for three types of feature sets. Second, we also remove duplicate features. After applying this filters we are left with 33,832, 12,974, and 5,785 features for three types of feature sets. Third, since we still have a large number of features, we want to select important features that strongly correlate with either positive or negative class. We further filter the remaining features using chi-square correlation [68].

$$\chi^2 = \frac{N \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)}$$

Where N is the total number of scripts. A is the number of positive samples where the binary feature is present. B is the number of negative scripts where the binary feature is present. C is the number of positive samples where the binary feature is absent. D is the number of negative samples where the binary feature is absent. Based on chi-square values, we rank features in the order of their importance. We select top 10K, 5K, 1K, and 100 features from three types of feature sets for further analysis.

Classifier Training. We decide to use AdaBoost, a boosting algorithm, due to the imbalance of anti-adblockers in the wild. AdaBoost [46] is an ensemble classifier that aims to create a strong meta-classifier from multiple weak classifiers. Models are built from

the training data, each subsequent model attempts to correct the errors from the previous model. AdaBoost is expressed as:

$$f(x) = sign(\Sigma_{t=1}^{T} \alpha_t h_t(x))$$

Here x is the input vector, $h_t(x)$ is the component classifier, and α_t is the weight of each classifier. The distribution of weights is uniform at the start. At each cycle t, weight distribution is updated according to the result of component classifier $h_t(x)$. Training samples that are misclassified get higher weights. This process continues for T cycles and the results are combined at the end. Component classifiers with lower training errors get higher weights. We use SVM as the component classifier for AdaBoost. AdaBoost with SVM using RBF (Radial Basis Function) as its kernel tends to perform better for imbalanced classification problems [54].

Results & Evaluation. Next we evaluate the effectiveness of classifier using our labeled set of anti-adblocking and non anti-adblocking scripts. To check the classifier performance we do 10-fold cross validation. We use 9-folds as training samples and the remaining fold as testing sample and repeat this process 10 times. We report results in the form of True Positive (TP) rate and False Positive (FP) rate. TP rate is the fraction of correctly classified anti-adblock scripts. FP rate is the fraction of incorrectly classified non anti-adblock scripts. Table 3 lists the classification results for different feature sets. We include SVM without AdaBoost for baseline comparison. The TP rate is above 99.2% for all configurations. The FP rate varies from 3.2% to 9.1% for different configurations. We achieve the highest TP rate of 99.7% and the lowest FP rate of 3.2% for AdaBoost + SVM classifier using top-1K features from the **keyword** feature set.

Classifier	# Features	TP rate (%)	FP rate (%)
Feature set: all			
AdaBoost + SVM	10K	99.6	3.9
AdaBoost + SVM	1K	99.2	8.9
AdaBoost + SVM	100	99.2	8.9
SVM	10K	99.2	8.6
SVM	1K	99.2	8.9
SVM	100	99.2	8.4
Feature set: literal			
AdaBoost + SVM	10K	99.6	3.9
AdaBoost + SVM	1K	99.2	9.1
AdaBoost + SVM	100	99.2	8.9
SVM	10K	99.2	8.4
SVM	1K	99.2	8.9
SVM	100	99.2	8.6
Feature set: keyword			
AdaBoost + SVM	5K	99.6	3.7
AdaBoost + SVM	1K	99.7	3.2
AdaBoost + SVM	100	99.2	8.9
SVM	5K	99.2	8.4
SVM	1K	99.2	8.4
SVM	100	99.2	8.6

Table 3: Accuracy of our machine learning based approach in detecting anti-adblocking scripts.

We further test our model on anti-adblocking scripts detected on Alexa top 100K live websites (§4.3). Out of 5,070 detected anti-adblocking websites, we extract 2,701 unique anti-adblocking scripts. We exclude the scripts in Alexa top 5K websites because they are used to train our model. We classify the 2,701 extracted scripts with the AdaBoost + SVM classification model with top-1K features configuration. We achieve a TP rate of 92.5%.

Our machine learning approach can be used in an offline or an online manner by adblockers. In the offline scenario, filter list authors can periodically crawl popular websites and run our trained model to identify new anti-adblock scripts. This will substantially reduce the manual labor required by filter list authors in maintaining filter lists as they only have to analyze the scripts detected by our model. The manual analysis of anti-adblocking scripts by filter list authors will help in reducing false positives that cause site breakage. In the online scenario, our trained machine learning model can be directly shipped in adblockers which would scan all scripts to detect and remove anti-adblock scripts on the fly.

6 CONCLUSION

In this paper, we presented a retrospective measurement study of anti-adblocking. We reported that anti-adblocking has significantly increased over the last few years. Our analysis of anti-adblock filter lists revealed limitations of manually curated filter lists. We note that two popular anti-adblock filter lists are not only implemented differently, they have different coverages and update frequency. We also presented a machine learning based approach to automatically identify anti-adblock scripts. Our proposed approach can be used to augment the manual efforts filter list in an offline manner or incorporated in adblockers for online detection.

The tussle playing out between online publishers and adblockers is set to have a major impact on the Internet. Adblockers are changing the status quo of ad-driven monetization of online content and services. We believe that the large-scale retrospective analysis of anti-adblocking provided by our work is important to inform future discussions surrounding adblocking, both technical and economic.

ACKNOWLEDGMENTS

We would like to thank our shepherd, Matteo Varvello, and the anonymous reviewers for their useful feedback on this paper. This work is supported in part by the National Science Foundation under grant numbers 1715152 and 1719147, and by a seed grant from the Data Transparency Lab (DTL).

REFERENCES

[1] Acceptable ads program. https://adblockplus.org/acceptable-ads.
[2] AdBlock. https://getadblock.com/.
[3] AdBlock, Chrome web store. https://chrome.google.com/webstore/detail/adblock/gighmmpiobklfepjocnamgkkbiglidom?hl=en-US.
[4] Adblock Plus. https://adblockplus.org/.
[5] Adblock Plus, Chrome web store. https://chrome.google.com/webstore/detail/adblock-plus/cfhdojbkjhnklbpkdaibdccddilifddb.
[6] Adblock Plus, Mozilla Firefox add-on. https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/.
[7] Adblock rules list parser. https://github.com/shawa/adblockparser.
[8] Anti-Adblock Killer. https://github.com/reek/anti-adblock-killer.
[9] Anti-Adblock Killer List Forum. https://github.com/reek/anti-adblock-killer/issues.

[10] BlockAdBlock. https://github.com/sitexw/BlockAdBlock/blob/master/blockadblock.js.
[11] Blockzilla. https://zpacman.github.io/Blockzilla/.
[12] Brave Browser. https://brave.com/.
[13] Cliqz Browser. https://cliqz.com/us/.
[14] Coalition for Better Ads. https://www.betterads.org/.
[15] Disconnect.me filter list. https://disconnect.me/.
[16] EasyList. https://easylist.to/.
[17] EasyList Forum. https://forums.lanik.us.
[18] EasyList variants. https://easylist.to/pages/other-supplementary-filter-lists-and-easylist-variants.html.
[19] Fanboy's Enhanced Tracking List. https://fanboy.co.nz/.
[20] Filter lists syntax. https://adblockplus.org/en/filter-cheatsheet.
[21] Firebug. http://getfirebug.com/.
[22] Ghostery. https://www.ghostery.com/.
[23] Ghostery, Chrome web store. https://chrome.google.com/webstore/detail/ghostery/mlomiejdfkolichcflejclcbmpeaniij?hl=en-US.
[24] Ghostery, Mozilla Firefox add-on. https://addons.mozilla.org/en-US/firefox/addon/ghostery/.
[25] HAR File. https://en.wikipedia.org/wiki/.har.
[26] McAfee's URL categorization service. https://www.trustedsource.org/.
[27] Mozilla Firefox. https://www.mozilla.org/en-US/firefox/.
[28] Mozilla Firefox tracker blocking. https://testpilot.firefox.com/experiments/tracking-protection.
[29] NetExport. https://getfirebug.com/wiki/index.php/Firebug_Extensions.
[30] NoTrack Blocklist. https://github.com/quidsup/notrack.
[31] PageFair, 2017 Global Adblock Report. https://pagefair.com/downloads/2017/01/PageFair-2017-Adblock-Report.pdf.
[32] Privacy Badger. https://www.eff.org/privacybadger.
[33] Privacy Badger, Chrome web store. https://chrome.google.com/webstore/detail/privacy-badger/pkehgijcmpdhfbdbbnkijodmdjhbjlgp?hl=en-US.
[34] Privacy Badger, Mozilla Firefox add-on. https://addons.mozilla.org/en-US/firefox/addon/privacy-badger17/.
[35] Selenium. http://docs.seleniumhq.org/.
[36] Truth In Advertising, Federal Trade Commission. https://www.ftc.gov/news-events/media-resources/truth-advertising/.
[37] Warning removal list. https://easylist-downloads.adblockplus.org/antiadblockfilters.txt.
[38] Wayback Machine. https://archive.org/web/.
[39] Wayback Machine API. https://archive.org/help/wayback_api.php.
[40] Wayback Machine Archive Details. https://archive.org/about/.
[41] YourAdChoices. http://youradchoices.com/.
[42] A. Bosworth. A New Way to Control the Ads You See on Facebook, and an Update on Ad Blocking. https://newsroom.fb.com/news/2016/08/a-new-way-to-control-the-ads-you-see-on-facebook-and-an-update-on-ad-blocking/, 2016.
[43] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *USENIX Security Symposium*, 2011.
[44] S. Englehardt and A. Narayanan. Online Tracking: A 1-million-site Measurement and Analysis. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.
[45] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies That Give You Away:The Surveillance Implications of Web Tracking . In *World Wide Web (WWW) Conference*, 2015.
[46] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Sciences*, 1997.
[47] M. Graham. robots.txt meant for search engines don't work well for web archives. https://blog.archive.org/2017/04/17/robots-txt-meant-for-search-engines-dont-work-well-for-web-archives/, 2017.
[48] D. Gugelmann, M. Happe, B. Ager, and V. Lenders. An Automated Approach for Complementing Ad Blockers' Blacklists. In *Privacy Enhancing Technologies Symposium (PETS)*, 2015.
[49] S. Ihm and V. S. Pai. Towards Understanding Modern Web Traffic. In *ACM Internet Measurement Conference (IMC)*, 2011.
[50] M. Ikram, H. J. Asghar, M. A. Kaafar, A. Mahanti, and B. Krishnamurthy. Towards Seamless Tracking-Free Web:Improved Detection of Trackers via One-class Learning . In *Privacy Enhancing Technologies Symposium (PETS)*, 2017.
[51] B. Krishnamurthy and C. E. Wills. Privacy Diffusion on the Web: A Longitudinal Perspective. In *World Wide Web (WWW) Conference*, 2009.
[52] A. Lerner, T. Kohno, and F. Roesner. Rewriting History: Changing the Archived Web from the Present. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.
[53] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *USENIX Security Symposium*, 2016.
[54] X. Li, L. Wang, and E. Sung. AdaBoost with SVM-based component classifiers. In *Engineering Applications of Artificial Intelligence*, 2007.
[55] M. Malloy, M. McNamara, A. Cahn, and P. Barford. Ad Blockers: Global Prevalence and Impact. In *ACM Internet Measurement Conference (IMC)*, 2016.
[56] J. Marshall. The Rise of the Anti-Ad Blockers. https://www.wsj.com/articles/the-rise-of-the-anti-ad-blockers-1465805039, 2016.
[57] J. R. Mayer and J. C. Mitchell. Third-Party Web Tracking: Policy and Technology. In *IEEE Symposium on Security and Privacy*, 2012.
[58] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl. Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools. In *IEEE European Symposium on Security and Privacy*, 2017.
[59] M. H. Mughees, Z. Qian, and Z. Shafiq. Detecting Anti Ad-blockers in the Wild . In *Privacy Enhancing Technologies Symposium (PETS)*, 2017.
[60] R. Nithyanand, S. Khattak, M. Javed, N. Vallina-Rodriguez, M. Falahrastegar, J. E. Powles, E. D. Cristofaro, H. Haddadi, and S. J. Murdoch. Adblocking and Counter-Blocking: A Slice of the Arms Race. In *USENIX Workshop on Free and Open Communications on the Internet*, 2016.
[61] E. Pujol, O. Hohlfeld, and A. Feldmann. Annoyed Users: Ads and Ad-Block Usage in the Wild. In *ACM Internet Measurement Conference (IMC)*, 2015.
[62] M. Z. Rafique, T. V. Goethem, W. Joosen, C. Huygens, and N. Nikiforakis. It's Free for a Reason: Exploring the Ecosystem of Free Live Streaming Services. In *Network and Distributed System Security Symposium (NDSS)*, 2016.
[63] S. Ramaswamy. Building a better web for everyone. https://www.blog.google/topics/journalism-news/building-better-web-everyone/, 2017.
[64] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web . In *USENIX Symposium on Networked Systems Design and Implementation (NDSI)*, 2012.
[65] G. Storey, D. Reisman, J. Mayer, and A. Narayanan. The Future of Ad Blocking: An Analytical Framework and New Techniques. In *arXiv:1705.08568*, 2017.
[66] R. J. Walls, E. D. Kilmer, N. Lageman, and P. D. McDanie. Measuring the Impact and Perception of Acceptable Advertisements. In *ACM Internet Measurement Conference (IMC)*, 2015.
[67] J. Wilander. Apple Safari Intelligent Tracking Prevention. https://webkit.org/blog/7675/intelligent-tracking-prevention, 2017.
[68] Y. Yang and J. O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *International Conference on Machine Learning*, 1997.
[69] Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol. Tracking the Trackers. In *World Wide Web (WWW) Conference*, 2016.