Muhammad Haris Mughees, Zhiyun Qian, and Zubair Shafiq

# Detecting Anti Ad-blockers in the Wild

**Abstract:** The rise of ad-blockers is viewed as an economic threat by online publishers who primarily rely on online advertising to monetize their services. To address this threat, publishers have started to retaliate by employing *anti ad-blockers*, which scout for ad-block users and react to them by pushing users to whitelist the website or disable ad-blockers altogether. The clash between ad-blockers and anti ad-blockers has resulted in a new arms race on the Web. In this paper, we present an automated machine learning based approach to identify anti ad-blockers that detect and react to ad-block users. The approach is promising with precision of 94.8% and recall of 93.1%. Our automated approach allows us to conduct a large-scale measurement study of anti ad-blockers on Alexa top-100K websites. We identify 686 websites that make visible changes to their page content in response to ad-block detection. We characterize the spectrum of different strategies used by anti ad-blockers. We find that a majority of publishers use fairly simple first-party anti ad-block scripts. However, we also note the use of third-party anti ad-block services that use more sophisticated tactics to detect and respond to ad-blockers.

**Keywords:** ad-blockers, anti ad-blockers

DOI 10.1515/popets-2017-0032
Received 2016-11-30; revised 2017-03-15; accepted 2017-03-16.

## 1 Introduction

The online advertising industry has been largely fueling the Web for the past many years. According to the Interactive Advertising Bureau (IAB), the annual online advertising revenues in the United States totaled $59.6 billion for 2015 [1]. Online advertising plays a critical role in allowing web content to be offered free of charge to end-users, with the implicit assumption that end-users agree to watch ads to support these "free" services. Unfortunately, the economic magnetism of online advertising has made it an attractive target for various types of abuses. First, the online advertising ecosystem incentivizes the widespread tracking of users across websites raising *privacy* [2] and *surveillance* [3] concerns. To show targeted ads to users, advertisers track users across the web using cookies, beacons, and fingerprinting [4]. Second, the online advertising ecosystem not only *lacks transparency* but also does not provide any meaningful control for users to limit tracking and the use of personal information. Third, hackers are increasingly launching *malvertising* campaigns, where they use online advertising to target malware at a large number of users [5]. Finally, many publishers choose to place ads that interfere (*e.g.* autoplay, pop-ups, and animation) with the organic content and *annoy* users [6].

Ad-blockers have become popular in recent years and they can block ads and/or trackers seamlessly without requiring any user input. A wide range of ad-block extensions are available for popular web browsers. Adblock Plus is the most popular ad-block extension [7, 8]. 22% of the most active residential broadband users of a major European ISP use Adblock Plus [9]. Another recent study showed that 18% of users in the U.S. and and 32% of users in Germany have installed ad-blockers [8]. According to PageFair, more than 600 million people around the world use ad-blockers on desktop and mobile devices [10, 11]. To the online advertising industry and content publishers, ad-blockers are becoming a growing threat to their business model.

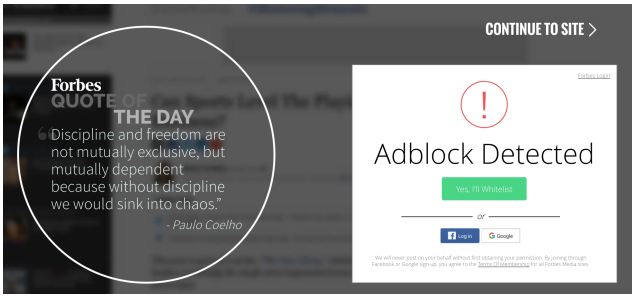
To combat ad-blockers, two strategies have emerged: (1) publishers such as Google and Microsoft have enrolled in the acceptable ads program [12] to have their ads whitelisted; and (2) publishers have begun to detect the presence of ad-blockers and may refuse to serve any user with ad-blocker turned on. The latter strategy has emerged as an increasingly popular solution to counter ad-blockers. For example, Yahoo! Mail [13], WIRED [14], and Forbes [15] reportedly did so recently. The anti ad-block phenomenon has been manually studied by researchers in [16, 17] on a relatively small scale due to lack of automated anti ad-block detection methods. To fill this gap, in this work we perform automated detection and measurement of the anti ad-block phenomenon in the wild. Specifically, we are interested in understanding: (1) how many websites are reacting to

**Muhammad Haris Mughees:** University of Illinois Urbana-Champaign, E-mail: mughees2@illinois.edu
**Zhiyun Qian:** University of California-Riverside, E-mail: zhiyunq@cs.ucr.edu
**Zubair Shafiq:** The University of Iowa, E-mail: zubair-shafiq@uiowa.edu

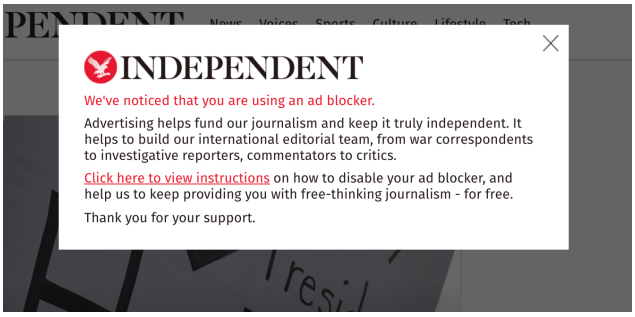
ad-blockers; and (2) what type of technical approaches are used.

**Key Contributions.** The key contributions and findings of the paper are the following:

- We propose a machine learning based approach to automatically identify websites that use anti ad-blockers. Our key idea is that websites that employ anti ad-blockers will make distinct changes to their web page content for ad-block users (*e.g.* displaying a pop-up message) as compared to users without ad-blockers. We extract these distinct features and feed them to train machine learning models. The approach is promising with precision of 94.8% and recall of 93.1%.
- Using our proposed approach, we conduct a measurement study of Alexa top-100K websites. The results show that 686 websites visibly react to ad-blockers. Out of these websites, 165 websites show intrusive notifications that cannot be dismissed by users without disabling their ad-blocker or whitelisiting the website. We also find that some websites have started to ask ad-block users to pay a subscription fee (42 websites) or make a donation (30 websites).
- We cluster different anti ad-block approaches by analyzing their JavaScript snippets. Our analysis shows that anti ad-block approaches range from fairly simple to much more sophisticated. We find that many websites use simple anti ad-block scripts that are served from first-party domains. Others use third-party anti ad-block services that are much more sophisticated.




(a) forbes.com




(b) independent.co.uk

**Fig. 1.** Examples of ad-block detection responses.

## 2 Background & Related Work

In this section, we first provide a brief background of ad-blockers and anti ad-blockers and then discuss relevant prior work.

### 2.1 Background

**The popularity of ad-blockers.** The issues with online ads have resulted in the proliferation of ad-blocking software. Ad-blocking software (or ad-blocker) is an effective tool that blocks ads seamlessly, generally published as extensions in web browsers such as Chrome and Firefox. More recently, Apple has also allowed content blocking plugins on iOS devices [18]. Other popular relevant tools such as Ghostery [19] are primarily focused on protecting user privacy by blocking trackers. Most ad-blockers include filter lists to remove both ads (*e.g.* EasyList [20]) and/or trackers (*e.g.* EasyPrivacy [21]). Recent reports have shown that the number of users using ad-blocking software has rapidly increased worldwide. According to PageFair, more than 600 million people around the world use ad-blockers on desktop and mobile devices [10, 11].

**How do ad-blockers work?** Ad-blockers eliminate ads by either *page element removal* or *web request blocking*. For page element removal, ad-blockers use various CSS selectors to access the elements and remove them. For web request blocking, ad-blockers look for particular URLs and remove the ones which belong to advertisers. For both of these actions, ad-blockers are dependent on *filter lists* that contain the set of rules (as regular expressions) specifying the element selectors and domains to remove. There are various kinds of filter lists available which can be included in ad-blockers. Each of these lists serves a different purpose. For example, Adblock Plus by default includes EasyList [20], which provides rules for removing ads from English websites. EasyPrivacy [21] helps ad-blockers to protect user privacy by removing trackers.

**The rise of anti ad-blockers.** The online advertising industry sees ad-blocking tools as a growing


(a) (b) (c)

**Fig. 2.** Web page load evolution for www.vipleague.tv. **(a)** The original website content is loaded. **(b)** Ad-blocker removes ads from the page. **(c)** Anti ad-blocker blocks the content and shows a pop-up notification asking the user to disable ad-blocking software.

threat to the ad-powered "free" web business model. The widespread use of ad-blockers has prompted a cat-and-mouse game between publishers and ad-blocking software. IAB recently released a script [22] to DEAL (**D**etect, **E**xplain, **A**sk, **L**imit) with ad-blockers. Using such scripts, publishers have started to detect whether users are visiting their websites while using ad-blocking software (detection step). Once detected, publishers notify users to turn off their ad-blocking software (reactions step). These notifications can range from a mild non-intrusive message which is integrated inside website content to more aggressive blocking of website content and/or functionality. Figure 1 shows a couple of examples of ad-block detection responses. To detect the use of ad-blocking software, publishers employ anti ad-block scripts in their pages. When a user with the ad-blocking software opens such a website, these scripts typically monitor the visibility of ads on the page to identify the use of ad-blockers. If ads are found hidden or removed by an ad-blocker, publishers take countermeasures according to their policies.

```
1 //step 1: set timeout
2 var myVar = setInterval(function() {
3   myFunc()
4 }, 2000);
5
6 function myFunc() {
7
8   // step 2: condition check
9   if (window.iExist === undefined || (!$(
10  "#XUinXYCfBvqpyDHOrOAVClxoWJemrlPpfCdWfiyAzNY")
11  .is(":visible") && (($(".vip_052x003").height()
12  < 100 && !$("#vipchat").length) &&
13  $(".vip_09x827").height() < 25))) {
14
15      //step 3: response
16      $("#XUinXYCfBvqpyDHOrOAVClxoWJemrlPpfCdWfiy
17      AzNY").css("width:100%;height:100%;position
18      :fixed;z-index:999999;top:0");
19      $("#XUinXYCfBvqpyDHOrOAVClxoWJemrlPpfCdW
20      fiyAzNY").show()
21  }
22  else if ($("#XUinXYCfBvqpyDHOrOAVClxoWJemrlPpfC
23  dWfiyAzNY").is(":visible") && $(".vip_052x003")
24  .height() > 249) {
25
26      $("#XUinXYCfBvqpyDHOrOAVClxoWJemrlPpfCdWfi
27      yAzNY").hide()
28  }
29 }
```

**Fig. 3.** Ad-block detection JavaScript extracted from http://www.vipleague.tv

**Illustration of anti ad-blockers.** To understand how anti ad-blockers operate, let's walk through the complete cycle of a typical anti ad-blocker. Figure 2 shows the web page loading process of http://www.vipleague.tv, which detects the presence of ad-blockers and subsequently reacts to them, on a web browser with Adblock Plus. A JavaScript is employed by the publisher which attempts to detect the presence of ad-blocker. Figure 3 shows the JavaScript employed by http://www.vipleague.tv. The functionality of the JavaScript can be divided into three parts: timeout, condition check, and response. In Figure 2, we note that the web browser starts loading the HTML and other resources included in the HTML code. While the content is loading, ad-block extension kicks in and starts evaluating the HTML code and page content to remove potential ads. Since the ad-blocker starts working after a small delay, the anti ad-block script has to wait some time before monitoring the ads. In Figure 3, the timeout is set at 2000 milliseconds. Once the timer expires (typically a few seconds), the condition check is executed to verify the presence/absence of ads, *e.g.* by checking the *height*, *width*, or *visibility* of ad frames. If the script detects that ads were removed or hidden, then the response step is executed. As discussed earlier, the implementation details of this step varies across publishers. A few publishers gently request users to remove/disable their ad-blockers, while others aggressively show a page-wide notification and/or block content. For example, in Figure 3, the publisher responds by changing CSS properties of a div to show a pop-up message that asks the user to disable ad-blocker or whitelist the site.

## 2.2 Related Work

Below, we discuss prior literature on online advertising, ad-blockers, and anti ad-blockers.

**Online Advertising.** Online advertising relies on sophisticated tracking of users across the web to target personalized ads. Roesner et al. [23] conducted an active measurement study of third-party tracking on the Web. They found a total of 7264 (524 unique) trackers on the Alexa top 500 websites. They found that a few trackers cover a large fraction of popular websites. More specifically, they reported that Google Analytics and DoubleClick (both owned by Google) are used on 40%–60% of the top 500 websites. Using real users' browsing history, they reported that a few trackers cover as much as 66% of the pages visited by a user. Metwalley et al. [24] conducted a passive measurement study to determine the extent of tracking on the Web. They found more than 400 tracking services, of which top 100 regularly track more than 50% of users. They found that 80% of the users are tracked by at least one tracking service within a second after starting web browsing. Lerner et al. [25] conducted a longitudinal active measurement study using the Wayback Machine. The authors found that the scope and complexity of online tracking has dramatically increased over the last 20 years. They showed that popular websites use more third-party trackers now than ever before. The authors 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 [4] showed that a handful of third parties including Google, Facebook, Twitter, and AdNexus track users across a majority of top 1 million websites. The authors showed that Google alone tracks users across more than 80% of the top 1 million websites. The authors also showed that stateless tracking based on Canvas, WebRTC, and AudioContext fingerprinting is employed across thousands of websites.

Researchers have also studied security aspects of online advertising. Li et al. conducted the first large-scale study of malicious advertising (called *malvertising*) on the Web [26]. Their analysis of 90,000 websites showed that not only malicious ads affect top websites but they also evade detection by various cloaking techniques. Zarras et. al. also conducted a large-scale study to determine the extent at which users are exposed to malicious advertisements [5]. Their measurement study of more then 60,000 ads showed that around 1% ads exhibit malicious behavior. They also showed that a few ad networks are more prone to malicious advertisements than others.

**Ad-blockers.** Due to the popularity of ad-blockers, researchers have tried to study the prevalence of ad-blockers. Pujol et al. conducted a measurement study using passive network traces of thousands of users from a European ISP to quantify ad-block usage [9]. Their results show that 22% of users use AdBlock Plus. They also found that ad-block users still generate significant ad traffic due their enrollment in the *acceptable ads program*. Walls et al. [27] conducted a study of the whitelists used by ad-blockers for allowing the acceptable ads. They analyzed the evolution of ad-block whitelists and performed active measurements on popular websites. Their analysis showed that the whitelist contains around 5,936 filters and 3,545 unique publisher domains. They reported that whitelists are inclined towards top ranked Alexa websites (59% filters are for top 5000 websites). Gugelmann et al. [28] proposed a methodology to complement manual filter lists of ad-blockers by automatically blacklisting intrusive ads. They trained a classifier on HTTP traffic statistics and identified around 200 new advertising and tracking services.

**Anti Ad-blockers.** Since the arms race between ad-blockers and anti ad-blockers is a relatively recent phenomenon, prior research is limited to small scale and manual analysis of anti ad-blockers. Researchers [16, 17] have recently reported anecdotal evidence of ad-block detection and retaliation by publishers. Rafique et al. [16] conducted manual analysis to report that 163 out of the top 1000 free live video streaming aggregators employ anti ad-blockers. Nithyanand et al. [17] clustered JavaScript snippets and manually analyzed the clusters to identify third-party anti ad-blockers. They found that 6.7% of top 5000 Alexa websites employ anti ad-blockers. Unfortunately such manual analysis is hard to scale up. In contrast, we use machine learning models to conduct large scale analysis of anti ad-blockers. Furthermore, while Nithyanand et al. [17] consider all websites that include anti ad-block scripts, we identify those anti ad-blockers that alter content to restrict access in response to ad-block detection because users only get affected by anti ad-blockers that retaliate by restricting user access. To the best of our knowledge, we present the first automated large-scale measurement study of anti ad-blockers in the wild.

# 3 Detecting Anti Ad-blockers

In this section, we design and implement our approach for automatically identifying websites that employ anti ad-blockers. The main premise of our approach is that websites conducting ad-block detection make distinct changes to their web page content for ad-block users as compared to users without ad-blockers. Our goal is to identify, quantify, and extract such distinct features that can be leveraged for training machine learning models to automatically detect websites that employ anti ad-blockers.

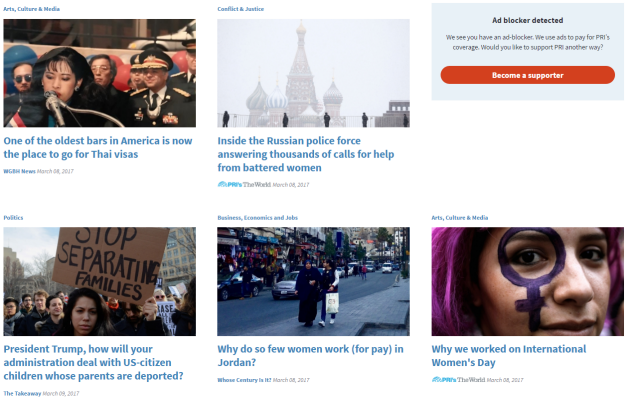



**Fig. 4.** Non-intrusive changes made by anti ad-blockers on www.pri.org (the top right news item is replaced with a warning about ad-blocker).

## 3.1 Overview

We want to identify distinct features that capture the changes made by anti ad-blockers to the HTML structure of web pages. One plausible approach is to apply image analysis which is commonly used to detect phishing websites [29, 30]. However, in pilot experiments, we found that the visible changes made by anti ad-blockers can be non-intrusive and hard to distinguish from other dynamically generated content. For instance, news websites may simply replace one news item with a different one that talks about ad-blocker (Figure 4). More details are given in §3.5. Therefore, we decided to analyze changes in the HTML content of web pages.

The changes made by anti ad-blockers to the HTML content can be categorized into: (1) addition of extra DOM nodes, (2) change in the style of existing DOM nodes, and (3) changes in the textual content. Below, we provide an overview of our proposed features and also discuss how they capture the changes made by anti ad-blockers in response to ad-block detection.

**Node changes.** In order to show notification to users with ad-blockers, websites dynamically create and add new DOM nodes. Thus, node additions in the DOM can potentially indicate anti ad-blockers. We can log the total number of DOM elements inserted in a web page.

**Style changes.** A few websites include notifications which are in their page content but hidden. If these websites detect the use of ad-blockers, they change the visibility of their notification. To cover such cases, we can log attribute changes to DOM elements of a web page.

**Text changes.** Some websites change the textual content (*i.e.* text-related nodes) in response to ad-block detection. Therefore, we can log changes in the textual content of a web page and addition of text-related nodes in a web page.

**Structure changes.** In addition to the above-mentioned features, we also consider other features like innerHTML to detect whether the structure is modified and track changes in URL to detect redirection.

## 3.2 Methodology

Figure 5 provides an overview of our methodology to automatically detect anti ad-blockers. We conduct A/B testing to compare the contents of a web page with and without ad-blocking software. To automate this process, we use the Selenium Web Driver [31] to open separate instances of the Chrome web browser, with and without Adblock Plus (we also attempted different configurations for Adblock Plus). We implement a custom Chrome browser extension to record changes in the content of web pages during the page load process. Our extension records the structure of the DOM tree, all textual content, and HTML code of the web page. We implement a feature extraction script to process the collected data and generate a feature vector for each website. We feed the extracted features to a supervised classification algorithm for training and testing. We train the machine learning model using a labeled set of websites with and without anti ad-blockers. Below we describe these steps in detail.

**Ad-blocker configurations.** Since different ad-blocker configurations can affect the results (some may trigger more anti ad-blocking than others), we decide to configure Adblock Plus in three different ways (each

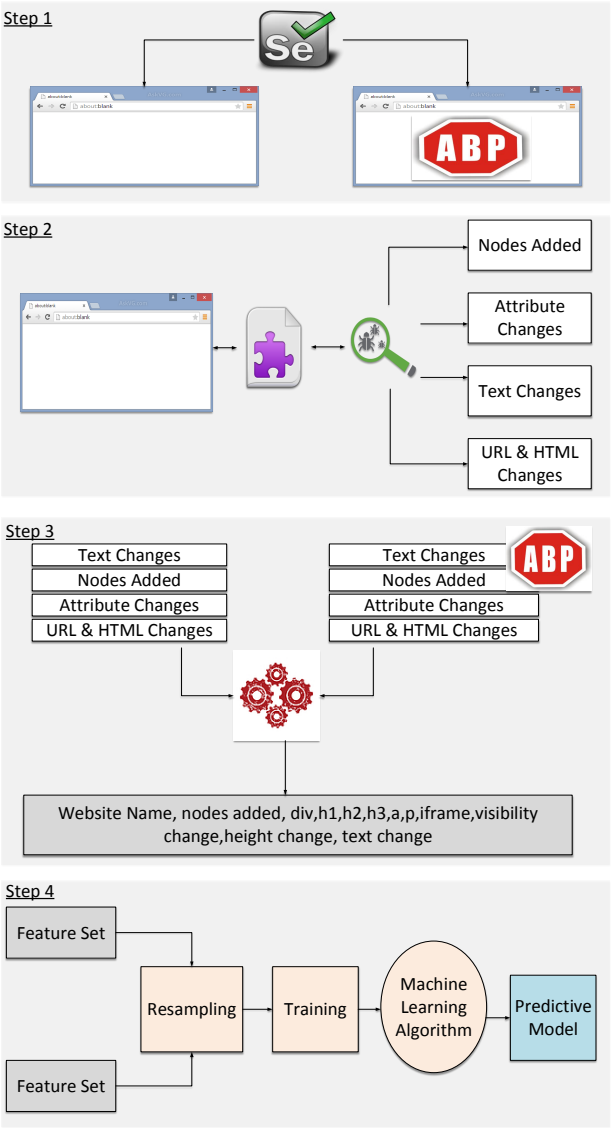



**Fig. 5.** Overview of our methodology for detecting anti ad-blockers.

configuration should catch more anti ad-blocking websites than the last):

- EasyList + Acceptable Ads. This represents the configuration for most users. By default, Adblock Plus includes EasyList to remove ads but allows some acceptable ads [12] to be displayed.
- EasyList. In this configuration, we use EasyList but disallow acceptable ads. Adblock Plus will block all ads (including the acceptable ads).
- EasyList - Anti Ad-block rules. In this configuration, we disallow acceptable ads as well as eliminate filter rules in EasyList that can circumvent anti ad-blockers (these rules suppress anti ad-blocking which makes the websites look as if no anti ad-blocking were employed). EasyList has recently started to include filter rules (1461 at the time of writing) to circumvent anti ad-blockers. These rules are separated in *!Anti-Adblock* sections of EasyList.

**A/B testing.** We implement a web automation tool using the Selenium Web Driver [31] to conduct measurements. For A/B testing, our tool first loads a website without Adblock Plus, and then opens it with Adblock Plus in a separate browser instance. However, we find that many websites host dynamic content that changes at a very small timescales. For example, some websites include dynamic images (*e.g.* logos), which can introduce noise in our A/B testing. Similarly, many news websites update their content frequently which can also add noise. Thus, we may incorrectly attribute these changes to the ad-blocker or the anti ad-blocker used by a publisher. To mitigate the impact of such noise, our tool opens two instances of each website without Adblock Plus in parallel (at the same time) and excludes content that changes across both instances.

**Data collection.** To collect data while a web page is loading, we use DOM Mutation Observers [32] to track changes in a DOM (*e.g.* `DOMNodeAdded`, `DOMAttrModified`, etc.). The changes we track include addition of new DOM nodes or scripts, node attribute changes like class change or style change, removal of nodes, changes in text, etc. We implement the data collection module as a Chrome extension. The extension is preloaded in the browser instances that are launched by our web automation tool. As soon as a web page starts loading, the extension attaches an observer listener with it. Whenever an event occurs, the listener fires and we record the information. For example, we record the identifier, type, value, name, parent nodes, and attributes of the corresponding node. For each attribute change, in addition to above-mentioned information, we record the name of attribute which changes like style or class and its old and new value. We also log page level data such as the complete DOM tree, innerText, and innerHTML as well.

**Feature extraction.** We then process the output of data collector to extract a set of informative features which can distinguish HTML content changes due to anti ad-blockers. Let **A** denote the data collected with an ad-blocker, and let **B** and **B'** denote the data collected by loading a web page twice without an ad-blocker. We provide details of the feature extraction

**Table 1.** Features used to identify anti ad-blockers

| Feature Set | Description |
|---|---|
| Nodes | Change in number of `div` nodes |
| | Change in number of `h1` nodes |
| | Change in number of `h2` nodes |
| | Change in number of `h3` nodes |
| | Change in number of `img` nodes |
| | Change in number of `table` nodes |
| | Change in number of `p` nodes |
| | Change in number of `anchor` nodes |
| | Change in number of `iframe` nodes |
| | Change in number of `text` nodes |
| | Total number of changes in nodes |
| Attributes | Change in number of display attributes |
| | Change in number of visibility attributes |
| | Change in number of height attributes |
| | Change in number of width attributes |
| | Change in number of opacity attributes |
| | Change in number of maxheight attributes |
| | Change in number of background size attributes |
| | Total number of changes in attributes |
| Textual | Change in number of lines |
| | Change in number of words |
| | Change in number of characters |
| | Keywords (binary as presence/absence) |
| Structural | Cosine similarity of HTML |
| | URL change (binary as yes/no) |

process below. Table 1 includes the list of all features used in our study.

• *Node features.* For each instance, we extract DOM related nodes because our pilot experiments revealed that websites using anti ad-blockers add only DOM related nodes. More specifically, we extract the list of `anchor`, `div`, `h1`, `h2`, `h3`, `img`, `table`, `p`, `iframe` and `text` nodes for each instance. Once we have a list of DOM nodes for each instance, we compare **A** vs. **B'** and **B** vs. **B'** to obtain the list of differences between these nodes. We denote these lists as **AB'** and **BB'** lists. As explained earlier, to remove node differences due to dynamic content of websites, we cross-validate nodes in **AB'** with **BB'** using their properties. Our key idea is that if a publisher ads random nodes to a web page, they may have different identifiers but most of the other properties will be fairly similar. Thus, we remove the nodes from **AB'** that also appear in **BB'**.

• *Attribute features.* For each instance, we extract changes in the style of DOM related nodes. More specifically, we focus on changes to the display-related node properties. For instance, we log whether the visibility property of a node changes from hidden to non-hidden. We also log changes to other display properties, *e.g.* the number of changes in height, width, and opacity of nodes. Similar to node features, we compare **A**, **B**, and **B'** to eliminate attribute changes from **AB'** that also appear in **BB'**.

• *Textual features.* We get the list of all text nodes in **A**, **B**, and **B'**. Using the lists, we extract changes in number of lines, worlds, and characters. We again compare **A**, **B**, and **B'** to eliminate changes in textual features from **AB'** that also appear in **BB'**. We also use seven keywords (adblock, ad-block, ad block, whitelist, blockadblock, pagefair, fuckadblock) as binary (presense/absense) features in **A**.

• *Structural features.* We compare differences in the overall page HTML using the *cosine similarity* metric. If the cosine similarity between **A** and **B**/**B'** is very low, it indicates significant content change. To check for potential URL redirections, we use change in URL as a binary (yes/no) feature.

**Model training and testing.** We feed the extracted features to a machine learning classifier for automatically detecting websites that employ anti ad-blockers. However, in order to train the classification algorithm, we need a sufficient number of labeled examples of websites that detect ad-blockers (*i.e.* positive samples) and websites that do not detect ad-blockers (*i.e.* negative samples). To get positive samples, we open websites with Adblock Plus and manually analyze whether the detect and respond to ad-blockers. Specifically, we analyze Alexa top-1K websites and some listed in crowdsourced lists [33, 34]. Overall, we identify a total of 200 positive training samples. Since a majority of Alexa top-1K websites do not deploy anti ad-blockers, we use them as negative training samples.

## 3.3 Feature Analysis

We analyze the extracted features to quantitatively understand their usefulness in detecting anti ad-blockers. We first visualize the distributions of a few features. Figure 6 plots the cumulative distribution functions (CDF) of two features. We observe that websites which employ anti ad-blockers tend to change more lines and add `div`

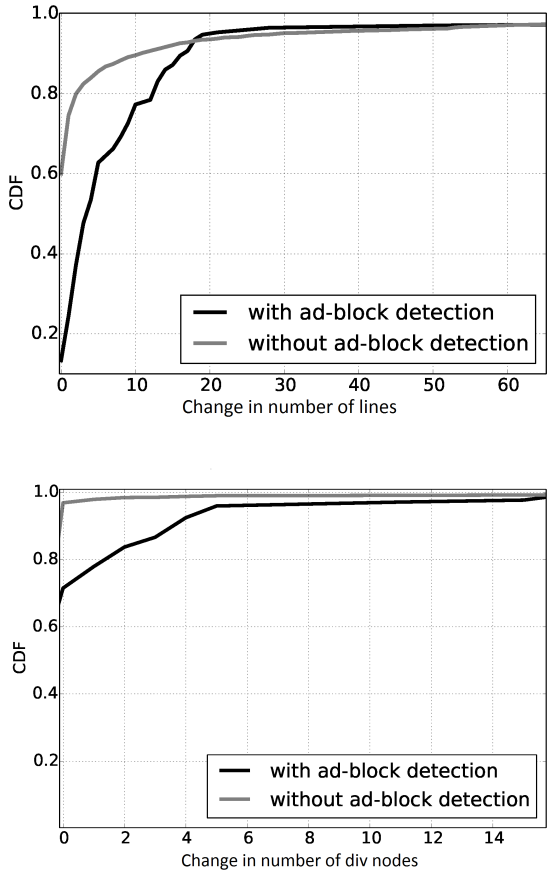



**Fig. 6.** Distribution of features used to identify anti ad-blockers

nodes than other websites. These distributions confirm our intuition that anti ad-blockers make changes in the page content that are distinguishable.

To systematically study the usefulness of different features, we employ the concept of *information gain* [35], which uses entropy to quantify how our knowledge of a feature reduces the uncertainty in the class variable. The key benefit of information gain over other correlation-based analysis is that it can capture non-monotone dependencies. Let $H(X)$ denote the entropy (*i.e.* uncertainty) of feature $X$. H is defined as:

$$H = -\sum_i p_i \log p_i$$

Let $H(Y)$ denote the entropy (*i.e.* uncertainty) of the binary class variable $Y$. Information gain is computed as:

$$IG(Y|X) = H(Y) - H(Y|X).$$

We can normalize information gain, also called *relative information gain*, as:

$$\frac{H(Y) - H(Y|X)}{H(Y)}.$$

**Table 2.** Features ranked based on information gain

| Name | Information Gain |
|---|---|
| Change in number of words | 35.44% |
| Change in number `text` nodes | 27.89% |
| Change in number of lines | 18.13% |
| Total number of changes in nodes | 17.37% |
| Change in number of characters | 17.19% |
| Change in number of `div` nodes | 13.01% |
| Change in number of height attributes | 10.67% |
| Change in number of display attributes | 8.67% |
| Total number of changes in attributes | 7.20% |
| Change in number of `img` nodes | 5.82% |

Using this, we can quantify what an input feature informs us about the presence of anti ad-blockers. Table 2 ranks the top 10 features based on their information gain. We note that textual features tend to have high information gain. They are followed by node and style based features (*e.g.* change in number of `div` nodes, change in number of height attributes).

## 3.4 Classifier Evaluation

We use the standard $k$-fold cross validation methodology to validate the accuracy of the trained models. For this purpose we select $k = 5$, divide the data into 5 folds where one fold is for testing while the other four folds are used for training. To quantify the classification accuracy of the trained models, we use the standard ROC metrics: precision, recall, and area under ROC curve (AUC). We try different machine learning classification methods. We tuned parameters of each of these models to optimize their classification performance. Table 3 summarizes the classification accuracy of these classifiers. We note that the random forest classifier, which is a combination of tree classifiers, outperforms the C4.5 decision tree and the naive Bayes classifiers. The random forest classifier achieves 93.1% recall, 94.8% precision, and 96.0% AUC.

**Table 3.** Effectiveness of different classifiers

| Classifier | Recall | Precision | AUC |
|---|---|---|---|
| **Random Forest** | **93.1%** | **94.8%** | **96.0%** |
| **C4.5 Decision Tree** | 87.0% | 89.0% | 91.3% |
| **Naive Bayes** | 82.0% | 82.4% | 89.0% |


**Fig. 7.** Visualization of decision tree model for anti ad-blockers

To further evaluate the effectiveness of different feature sets in identifying anti ad-blockers, we conduct experiments using stand alone feature sets and then evaluate their all possible combinations. We divide the features into node features, attribute features, textual features, and structural features. Among stand alone feature sets, as expected, textual features provide the best classification accuracy. We also observe that using combinations of feature sets does improve the classification accuracy. The best classification performance is achieved when all feature sets are used together.

To further gain some intuition from the trained machine learning models, we visualize a pruned version of the decision tree model trained on labeled data in Figure 7. As expected from the information gain analysis, we note that a text feature (change in number of words) is the root node of the decision tree. Moreover, changes in visibility and number of div nodes are indicative of anti ad-blockers. It is interesting to note that the top three features in the decision tree belong to different feature categories. This shows that different feature sets complement each other, rather than capturing similar information, which we also observed earlier when evaluating different combinations of features.

## 3.5 In the Wild Evaluation

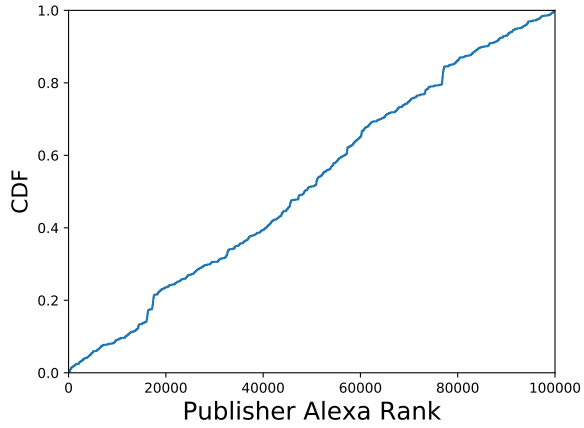
**Detecting anti ad-blockers in the wild.** We now apply our trained random forest model on the homepage of Alexa top 100K websites to gain an overall picture of anti ad-blockers in the wild. The experiments are conducted in February 2017. We report the results for the three different Adblock Plus configurations as mentioned in Section 3.2.

1. For the first configuration of EasyList + Acceptable Ads, the model predicted that 642 websites detect and respond to ad-blockers. We manually inspect all identified websites to understand its detection accuracy. Unfortunately, it is sometimes difficult for us to determine the ground truth when subtle and hard to identify changes (*e.g.* embedded text) are made, especially when the language is not English. Nevertheless, we conservatively estimated that 556 websites are true positives.
2. For the second configuration of EasyList, the model predicted that 651 websites detect and respond to ad-blockers. Upon manual inspection, 558 websites are true positives. In theory, more websites should react to this configuration as compared to the first configuration because websites are not allowed to display any ads (including the acceptable ads). We surmise that only small portion of ad-block users change the default setting, and it is therefore not causing significant loss of revenue to publishers. If more users were to disallow the acceptable ads, we expect the results to change.
3. For the third configuration of EasyList - Anti Adblock rules, the model predicted that 786 websites detect and respond to ad-blockers. Upon manual inspection, 686 websites are true positives. This represents an increase of 130 websites that employ anti ad-blocking compared to the first configuration. It is interesting to note that anti ad-block filter rules in EasyList are only moderately effective (130/686=19%) in evading anti ad-blockers.

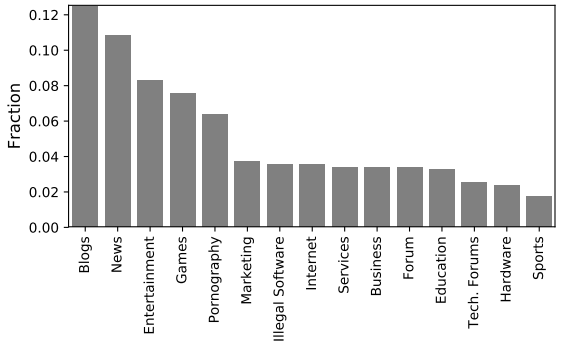
Due to the limitations of our methodology, as we discuss later, the results represent a lower bound of anti ad-block usage in the wild.

**Characterizing ad-block detection responses.** We manually analyze ad-block detection responses of the 686 identified websites to categorize how they respond to ad-blockers. We look at two different aspects: (1) how aggressive are the response messages; and (2) what do they ask the users to do.

We categorize the aggressiveness of the ad-block detection responses into three types: (a) non-intrusive notification, typically well integrated with the webpage content (409 websites); (b) intrusive notification (*e.g.* flashy popup) but it can be dismissed by the user (112 websites); and (c) intrusive notification which cannot be dismissed by the user without disabling ad-blocker or whitelisiting the website (165 websites). Our find-




(a) Alexa rank distribution




(b) Categories

**Fig. 8.** Characteristics of websites that employ anti ad-blockers

ings show that a majority of the websites are still conservative and do not want to lose users by showing intrusive notifications that can annoy users. However, a substantial number of these websites are taking a much more aggressive stance against ad-blockers, *e.g.* www.forbes.com, by not allowing users to view content if ad-block usage is detected.

We categorize the content of response messages into three types: (a) users are asked to disable the ad-blocker or whitelist the website (644 websites); (b) users are asked to donate (30 websites); and (c) users are asked to pay a subscription fee (42 websites). Note that the responses of some websites fall into multiple categories. We find that a small but non-trivial fraction of websites are actually considering alternative monetization models such as donations and paid subscriptions.

**Characterizing websites that employ anti ad-blockers.** We characterize the websites that use anti ad-blockers in terms of their popularity (using Alexa ranks) and categorization (using McAfee's URL categorization service). Figure 8(a) shows that websites employing anti ad-blockers are very much uniformly distributed across the Alexa top 100K, without any obvious skews. This can happen as websites that heavily rely on ads to monetize are spread across top 100K websites. Figure 8(b) shows the top 15 categories of the websites that employ anti ad-blockers. The overall categorization trend is similar to what has been observed for top 5K websites in [17]. The top five categories are blogs, news, entertainment, games, and pornography. The remaining 25% websites (not shown in the figure) are grouped in the other category.

**Limitations.** Our proposed methodology cannot identify all websites that employ anti ad-blocking. There are several reasons. First, we only visit the homepages of Alexa top-100K websites. Some websites may only employ anti ad-blocking on specific subpages. Second, some websites may detect ad-blockers but may not react to them. Since our methodology relies on detecting changes in the HTML content, we will not detect these websites. Third, the default EasyList is primarily constructed based on ads in English websites, and may not have the best coverage in blocking ads in other websites (although we do observe ads blocked in many non-English websites). Including additional filtering lists (such as those that target other languages) may improve the detection rate of anti ad-blocking [36]. Finally, websites can employ anti ad-blockers non-deterministically, *e.g.* once every 10 site visits, or after a long delay. Our measurements will likely miss these websites as well. In summary, our results represent a lower bound on the websites that employ anti ad-blockers, and we plan to address the limitations as future work.

# 4 Analyzing Anti Ad-block Scripts

In this section, we characterize the functionality of anti ad-blockers. More specifically, we analyze anti ad-block JavaScript code snippets to study different ad-block detection strategies. For systematic analysis, we first automatically cluster anti ad-blockers based on their JavaScript code similarity and then manually analyze different anti ad-block clusters. Through this analysis, we are able to identify several third-party anti ad-block services that are used by multiple publishers.

## 4.1 Collecting Anti Ad-block JavaScript Snippets

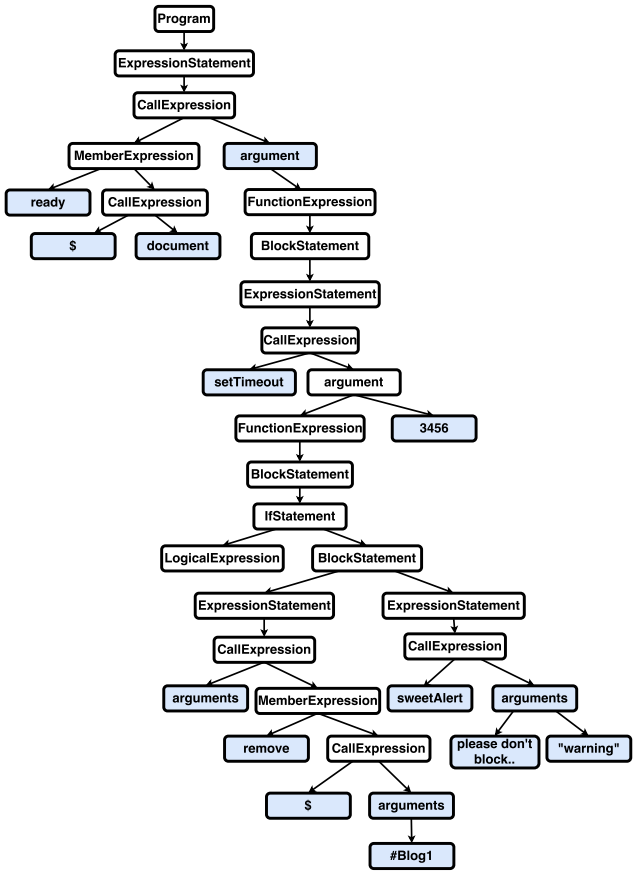
As a first step, we collect the JavaScript snippets on all websites that employ anti ad-blockers. Analyzing the functionality of JavaScript code is non-trivial because the code can be obfuscated and packed inside functions such as `eval`. To overcome these issues, we leverage the fact that the packed code needs to unpack itself before execution. We attach a debugger between the Chrome V8 JavaScript engine [37] and the web pages. Specifically, we observe `script.parsed` function, which is invoked when `eval` is called or new code is added with `<iframe>` or `<script>` tags. We implement the debugger as a Chrome extension and collect all JavaScript snippets parsed on a webpage and manually identify the snippet responsible for ad-block detection.

## 4.2 Clustering

We want to cluster anti ad-block JavaScript snippets into a few groups. To this end, we map JavaScript snippets using a tree representation and then group the ones with similar structure.

To identify similar anti ad-block JavaScript snippets, we first parse them to produce abstract syntax trees (ASTs). ASTs have been used in prior literature for JavaScript malware detection [38, 39]. ASTs allow us to retain the structural and logical properties of the code while ignoring fine-grained details, such as variable names, which are not useful for our analysis. Figures 9 and 10 show two anti ad-block JavaScript snippets and their corresponding AST visualizations. We use the Esprima JavaScript parser [40] to visualize the ASTs. Note that although these JavaScript snippets look fairly different due to different variable names and values, but their ASTs have similar logical structure except for minor differences near the leaf nodes.

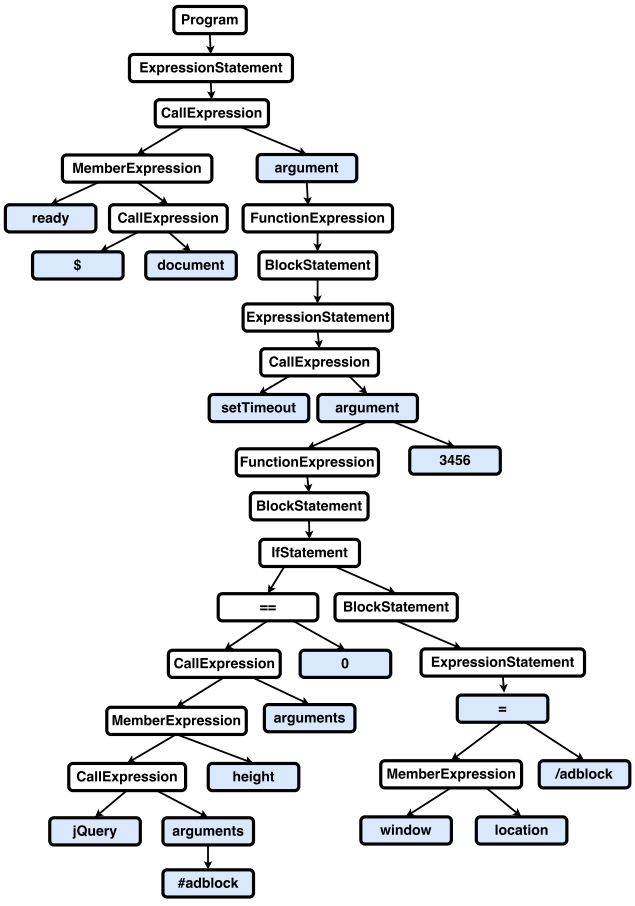
We transform ASTs of all anti ad-block JavaScript snippets to normalized node sequences by performing the pre-order traversal. These sequences are composed of different AST node types such as IfStatement, WhileStatement, and AssignmentExpression. Note that these sequences are of different lengths. We identify 88 distinct node types in the set of anti ad-block JavaScript snippets. To transform variable length normalized node sequences to a fixed number of dimensions, we convert each sequence into a 88-dimensional summary vector. Each anti ad-block JavaScript snippet is represented as an 88-dimensional point, where each dimension corresponds to a node type. The value of each dimension is the occurrence frequency of the node type in the sequence.




```
1 $(document).ready(function() {
2   setTimeout(function() {
3     if (localStorage.noad === undefined &&
4       (16 >= $("#gAds").height() ||
5       16 >= $("#gAd2".height()))) {
6             $("#Blog1").remove();
7             sweetAlert("Oops.. please don't
8             block my ADs","warning");
9         }
10      }, 3456)
11 });
```

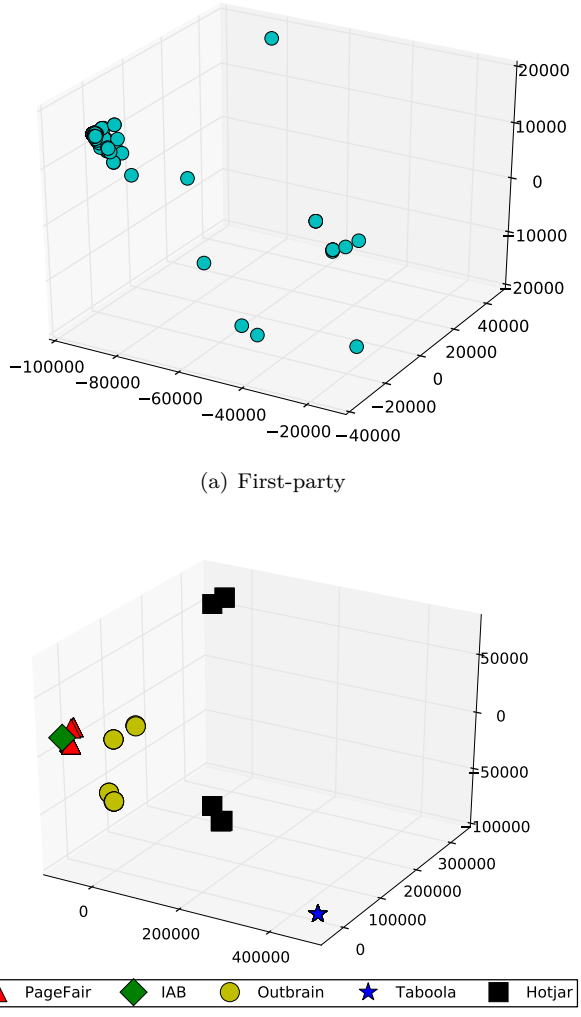
**Fig. 9.** Example 1: AST of anti ad-block JavaScript snippet.

We use Principal Component Analysis (PCA) to reduce the dimensionality of the summary vector for visualization. Figure 11 separately plots 3-dimensional visualizations of first- and third-party anti ad-block scripts. In Figure 11(a), we note that most first-party anti ad-block scripts are in a dense cluster. The dense cluster of first-party scripts indicates that they use a similar approach for ad-block detection. In Figure 11(b), third-party anti ad-block scripts are spread out from each other. Multiple clusters of third-party anti ad-block scripts indicate that they use different approaches




```
1 jQuery(document).ready(function() {
2   setTimeout( if (jQuery("#adblock").height() == 0){
3         window.location = "/adblock"
4 }, 3456);
5 });
```

**Fig. 10.** Example 2: AST of anti ad-block JavaScript snippet.




(a) First-party

(b) Third-party

**Fig. 11.** Visualization of anti ad-block clusters using PCA. Most first-party anti ad-blockers are in one dense cluster. In contrast, third-party anti ad-blockers are generally different from each other.

which are potentially more sophisticated than first-party scripts.

## 4.3 Analysis and Discussions

Next, we analyze ad-block detection strategies used by different anti ad-block clusters. We first study the anti ad-blockers in the dense central cluster of Figure 11(a). Most of these are customized anti ad-block scripts, which are served as first-party by publishers. Our inspection of different clusters in Figure 11(b) revealed that they are mostly third-party anti ad-block scripts. Below, we provide an in-depth analysis of both first-party and third-party anti ad-block scripts.

### 4.3.1 First-Party Anti Ad-Block Scripts

First-party anti ad-block scripts are fairly simple. Most of them are 5-10 lines of code, yet they can successfully detect state-of-the-art ad-blockers. Ad-blockers block ads by removing ad frames and they do not really try to hide their operation. Therefore, such simple first-party anti ad-blockers can easily detect ad-blockers.

**Timing.** First-party anti ad-blockers typically execute their logic at the beginning of the page load process. Since it may take a few seconds before an ad-blocker removes ads, some websites delay the execution of their logic using `setTimeout()` or `setTimeIntervel()`. In

```
//http://knowlet3389.blogspot.hk
if (localStorage.noad === undefined &&
(16 >= $("#gAds").height()
|| 16 >= $("#gAd2").height()))
//http://www.elahmad.com
if(jQuery("#adblock").height()==0)
//http://urlchecker.org
if ($("#adchecker").height() < 10)
//http://www.hentai.to
if(document.getElementById("tester")!=undefined)
//http://forum.pac-rom.com
if(!ad || ad.innerHTML.length == 0 ||
ad.clientHeight === 0)
```

**Fig. 12.** Examples of detection logic used by first-party anti ad-blockers

```
//http://knowlet3389.blogspot.hk
$("#Blog1").remove();
sweetAlert("Oops.. please don't block my ADs",
"warning");
//http://www.elahmad.com
window.location="/adblock"
//http://urlchecker.org
$("#ads_notify").fadeIn();
$("#getlinks").hide();
$("#adchecker_btn").fadeIn();
//http://www.hentai.to
document.write(
  '<p class="no">Please <u>disable</u> your
  adblocking software on hentai.to to keep
  our community <u>FREE</u>! ^.^</p>');
//http://forum.pac-rom.com
 alert("We've detected an ad blocker running
 on your browser...");
```

**Fig. 13.** Examples of responses by first-party anti ad-blockers

Figures 9 and 10, we show two example anti ad-block scripts, one inserting delay and another without delay. Since their ad-block detection logic is a one-time check (*i.e.* it is not invoked periodically), adding a delay helps to ensure that ad-blockers remove ads before anti ad-blockers try to detect ad-blockers.

**Detection logic.** To detect ad-blockers, first-party anti ad-blockers typically check different HTML elements to detect ad-blockers. In Figure 12, we illustrate the detection logic implemented by several websites. We note that the detection checks are fairly simple and intuitive. For example, consider `urlchecker.org`, which checks whether the *height* of *adcheker* `div` is less then 10pxs. Our further analysis revealed that Adblock Plus blocks *adsbygoogle.js* script due to which the *adcheker* `div` is empty and its height is equal to 1px. Other publishers check CSS properties of different `div` elements. Since the filter lists used by ad-blockers (*e.g.* EasyList [20]) are publicly available, anti ad-blockers can successfully setup such high-precision detection rules.

**Response.** Although the detection logic used by first-party anti ad-blockers is fairly similar across the board, their response to ad-blockers vary widely. Figure 13 lists a few of the responses. For *hentai.to*, a <p> element requests users to disable the ad-blocker. Since the original content is preserved, this response is not aggressive. However, for *knowlet3389.blogspot.hk*, the *#Blog1* `div` is removed upon ad-block detection, which indicates that the website hides its content from ad-block users. *elahmad.com* also aggressively responds by redirecting ad-block users to a warning page. Overall, as discussed earlier in Section 3.5, we find a wide spectrum of responses to ad-block detection, ranging from gentle requests to remove ad-blockers to more aggressive redirection.

Note that some first-party anti ad-block scripts in Figure 11(a) are separated from the dense cluster. Our analysis reveals that these scripts are generally more sophisticated than simple first-party anti ad-blockers. One such example is forbes.com, where the timing, detection logic, and response of its anti ad-blocker is much more sophisticated. First, instead of waiting for a fixed time before running the one-time check, it continuously checks for ad-blockers after periodic intervals. Second, it uses random ad divs as baits and checks their height or display properties for detection. Another unique aspect is that it uses cookies (*e.g.* forbes_ab) to keep track of users' detection status across multiple visits.

### 4.3.2 Third-Party Anti Ad-Block Scripts

Some publishers rely on third-party anti ad-block services. These third-party anti ad-block services not only enable publishers to detect ad-block users but also provide different ways to respond. For example, in addition to quantify ad-block usage, some third-party anti ad-block services "recoup lost advertising revenue" for publishers. Below we analyze PageFair, which is a popular third-party anti ad-block service that enables publishers to detect ad-block usage and display tailored non-intrusive ads to ad-block users.

**Timing.** PageFair performs multiple periodic checks at various stages of the web page load process to detect ad-blockers. This approach is much more sophisticated than most first-party anti ad-blockers and makes it harder for ad-blockers to evade detection by simply delaying their activity.

**Detection logic.** PageFair's detection logic attempts to actively trap ad-blockers by injecting different baits on web pages. In addition, PageFair attempts to check whether any ad-block plug-in is installed by looking for various browser resources exposed by ad-blocking extensions. The use of these methods makes PageFair's

```
//DIV bait
var b = document.createElement("DIV");
// d = "influads_block"
b.id = d;
//c=''
b.className = c;
//1x1 div
b.style.width = "1px";
b.style.height = "1px";
//div not located in visible frame
b.style.top ="-1000px";
b.style.left ="-1000px";
document.body.appendChild(b);
//jquery selector of created div
c = jQuery("#" + d);
// check if this div is hidden by ad-blocker
d = c.is(":hidden") ? 1 : 0;
//removing the created div
c.remove();

//SCRIPT bait
b = document.createElement("SCRIPT");
//d="295f89b1"
b.id = d;
b.type = "text/javascript";
document.getElementsByTagName("head")[0]
.appendChild(b);
//a="http://asset.pagefair.com/adimages/adsense.js"
b.src = a
```

**Fig. 14.** PageFair uses different baits to detect ad-blockers.

```
var c = {
    adblock: "chrome-extension://gighmmpiobklfepjoc
    namgkkbiglidom/img/icon24.png",

    adblock_plus: "chrome-extension://cfhdojbkjhnkl
    bpkdaibdccddilifddb/block.html",

    adblock_pro: "chrome-extension://ocifcklkibdehe
    kfnmflempfgjhbedch/components/block/block.html",

    adblock_premium: "chrome-extension://fndlhnanhe
    doklpdaacidomdnplcjcpj/img/icon24.png",

    adblock_super: "chrome-extension://knebimhccknd
    higlamoabbnifdkijidd/widgets/block/block.html",

    adguard: "chrome-extension://bgnkhhnnamicmpeena
    elnjfhikgbkllg/elemhidehit.png",

    adremover: "chrome-extension://mcefmojpghnacead
    nghednjhbmphipkb/img/icon24.png",

    ublock: "chrome-extension://epcnnfbjfcgphgdmggk
    amkmgojdagdnn/document-blocked.html"
}
```

**Fig. 15.** PageFair tries to access public extension resources to detect ad-blockers.

detection logic difficult to evade. We separately discuss both of these methods below.

• *Baiting:* Figure 14 shows two different types of baits used by PageFair. PageFair injects these baits with words like "ad" in the element name or URL to trigger ad-blockers. The first example shows an injected `div` element whose identifier is set to *influads_block* and it is not visible on the page. The second example shows script bait whose source is set to *adsense.js*, which is a common script used by Google AdSense. Note that several other anti ad-blockers also use script baits to detect ad-blockers which alleviates the need to check CSS properties of HTML elements. To identify such anti ad-blockers, we rerun our experiments described in Section 3 and remove all page element removal rules (that block ads based on the div names etc.) from EasyList and leave unchanged the remaining rules such as the request blocking rules (that block specific javascripts from being downloaded for example). We find that 149 out of 686 websites can still detect ad-blockers. Thus, anti ad-blockers on these 149 websites use non-CSS based approaches to detect ad-block users (likely by detecting the removal of bait javascripts).

• *Extension resources:* PageFair also attempts to detect the presence of ad-blockers by accessing extension resources exposed by various ad-blockers at `chrome-extension://`. Figure 15 shows how PageFair accesses extension resources to identify 8 popular ad-blockers including AdBlock, Adblock Plus, AdBlock Pro, AdBlock Premium, Adblock Super, Adguard, Ad Remover, and uBlock. For each type of ad-blocker, it includes a unique extension identifier, *e.g. gighmmpiobklfepjocnamgkkbiglidom* for AdBlock, and the resource file path. Note that Chrome generally does not allow web pages to directly access extension resources unless an extension specifies resources as `web_accessible_resources` in the manifest file and makes them publicly accessible. Our investigation showed that some ad-block extension resources, such as whitelisted pages shown in Figure 15, are indeed publicly accessible. For example, Adblock Plus exposes *block.html*, which allows websites to get the list of blocked URLs. Thus, ad-blockers are susceptible to leaking the proof of their presence to anti ad-blockers.

**Response.** PageFair provides a whitelist ad service under the acceptable ads manifesto [12]. To understand PageFair's service, we installed PageFair scripts on a test website that uses Google Ads. With ad-blocker, as expected, we find that Google ads are not served. Instead, PageFair shows a replacement ad which is hosted on PageFair's domain. It is noteworthy that PageFair's domain is whitelisted as part of the acceptable ads program. PageFair's response is reflective of the growing adoption of acceptable ads by many publishers and ad-blockers [27].

In addition to third-party anti ad-block services such as PageFair, several community scripts such as IAB [22] and BlockAdblock [41] are freely available for publishers to detect ad-block users. Publishers can serve

```
function attachOrFire() {
  var fireNow = false;
  var fn;

  //function triggers only when document is loaded
      completely
  if (document.readyState) {
    if (document.readyState == 'complete') {
      fireNow = true;
    }
  }

  fn = function() {
    if (_options.useLocalBait) {
      beginTest(quickBait, false);
    } else {
      log('ignoreing local bait - download remote');
      fetchRemoteLists();
    }
  }
}

function beginTest(bait, isRemote) {
  log('start beginTest');
  if (findResult == true) {
    return; // found bait
  }
  testExecuting = true;
  //function to create a bait
  castBait(bait);

  if (!isRemote) {
    exeResult.quick = 'testing';
  } else {
    exeResult.remote = 'testing';
  }

  timerIds.test = setTimeout(
    function() {
    //function to test bait
      reelIn(bait, 1);
    },
    5);
}
```

**Fig. 16.** Key functions in IAB's anti ad-block script

these scripts from first-party, third-party content delivery networks, or they can be integrated into third-party analytics scripts. Below we analyze IAB's community anti ad-block script to detect and respond to ad-block users. Note that this script is available both as an independent script and as a Google Analytics wrapper.

**Timing.** IAB's script makes multiple attempts to detect ad-blockers until the maximum_loop limit is reached. Instead of waiting for an arbitrary time, as shown in Figure 16, the script waits for the page to finish loading and then sets the fireNow flag to begin testing.

**Detection logic.** IAB's script injects realistic baits to detect ad-blockers. As shown in Figure 16, the script includes an option to specify an external ad-block filter list. If an external ad-block filter list is specified, the script parses the list and then creates baits based on the rules in the list. Otherwise, the default baits are used. The script checks different CSS properties of these baits to detect ad-block users.

**Response.** The script allows publishers to implement custom responses to ad-block detection. For example, publishers can show custom notifications, set cookies to track ad-block users, or load ads from alternate servers.

# 5 Concluding Remarks

We presented a machine learning based approach to study anti ad-blockers in the wild. Our main observation is that at least 686 websites in the Alexa top-100K currently deploy anti ad-blockers at their home page to detect ad-block users and respond with visible notifications. The notifications ask users to either disable ad-blockers, consider donation, or pay a subscription fee. Almost a quarter of these notifications cannot be dismissed unless users disable ad-blockers or pay up. Such attempts to undermine ad-blockers could harm their utility.

The arms race between ad-blockers and anti ad-blockers is rapidly escalating. To counter anti ad-blockers, ad-blockers have started to use filter lists to remove anti ad-block scripts and ad-block detection warnings. EasyList [20], which is used by ad-blockers to block ads, now contains rules that specifically target anti ad-blockers. There are also some dedicated filter lists to counter anti ad-blockers. Adblock Warning Removal List [33] mostly contains page element removal rules to remove warning messages. Anti-Adblock Killer list [34] mostly contains web request blocking rules to remove anti ad-block scripts. The rules in these crowdsourced lists are tailored to specific anti ad-blockers. Since anti ad-blockers continuously adjust their behavior, these rules need to be constantly updated to keep up with the changes. We envision the cat-and-mouse game between ad-blockers and anti ad-blockers to continue in future. We expect our findings and analysis to spur future research along this direction.

# Acknowledgements

We thank our shepherd, Vern Paxson, and the anonymous reviewers for their valuable feedback that considerably improved the paper. This work is supported in part by a grant from the Data Transparency Lab.

## References


[1] IAB Internet Advertising Revenue Report. http://www.iab.com/wp-content/uploads/2015/05/IAB_Internet_Advertising_Revenue_FY_2014.pdf, April 2015.

[2] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical privacy in online advertising. In *NSDI*, 2011.

[3] Micah Lee Morgan Marquis-Boire, Glenn Greenwald. NSA's Google for the World's Private Communications. https://theintercept.com/2015/07/01/nsas-google-worlds-private-communications/.

[4] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2016.

[5] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The Dark Alleys of Madison Avenue: Understanding Malicious Advertisements. In *ACM Internet Measurement Conference (IMC)*, 2014.

[6] Daniel G Goldstein, R Preston McAfee, and Siddharth Suri. The Cost of Annoying Ads. In *WWW*, 2013.

[7] AdBlock Plus. https://adblockplus.org.

[8] Matthew Malloy, Mark McNamara, Aaron Cahn, and Paul Barford. Ad blockers: Global prevalence and impact. In *ACM Internet Measurement Conference (IMC)*, 2016.

[9] Enric Pujol, Oliver Hohlfeld, and Anja Feldmann. Annoyed Users: Ads and Ad-Block Usage in the Wild. In *ACM Internet Measurement Conference (IMC)*, 2015.

[10] 2015 Adblocking Report. The PageFair Team. https://blog.pagefair.com/2015/ad-blocking-report.

[11] 2016 Mobile Adblocking Report. The PageFair Team. https://blog.pagefair.com/2015/ad-blocking-report, 2016.

[12] Allowing acceptable ads in adblock plus. https://adblockplus.org/acceptable-ads, Nov 2015.

[13] Jared Newman. Yahoo Mail begins blocking users with ad block enabled. http://www.pcworld.com/article/3006981/data-center-cloud/yahoo-mail-begins-blocking-users-with-ad-block-enabled.html.

[14] Jeremy Barr. Checking in on Wired's Ad-Blocking Experiment, Including an Ad-Free Version. http://adage.com/article/media/checking-wired-magazine-s-ad-blocking-experiment/303795/, 2016.

[15] Martin Anderson. Sites that block adblockers seem to be suffering. https://thestack.com/world/2016/04/21/sites-that-block-adblockers-seem-to-be-suffering/, 2016.

[16] M Zubair Rafique, Tom Van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS)*, 2016.

[17] 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. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2016.

[18] Leo Kelion. Apple brings ad-blocker extensions to Safari on iPhones. http://www.bbc.com/news/technology-34173732, September 2015.

[19] Ghostery. https://www.ghostery.com.

[20] EasyList. https://easylist-downloads.adblockplus.org/easylist.txt.

[21] EasyPrivacy. https://easylist-downloads.adblockplus.org/easyprivacy.txt.

[22] IAB. Ad Block Detection Code Access Request. https://www.iab.com/ad-block-detection-code-access-request/, 2016.

[23] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[24] Hassan Metwalley, Stefano Traverso, Marco Mellia, Stanislav Miskovic, and Mario Baldi. The Online Tracking Horde: A View from Passive Measurements. In *Traffic Monitoring and Analysis*. 2015.

[25] 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. In *Proceedings of USENIX Security*, 2016.

[26] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising. In *ACM CCS*, 2012.

[27] Robert J Walls, Eric D Kilmer, Nathaniel Lageman, and Patrick D McDaniel. Measuring the impact and perception of acceptable advertisements. In *ACM Internet Measurement Conference (IMC)*, 2015.

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

[29] S. Afroz and R. Greenstadt. Phishzoo: Detecting phishing websites by looking at them. In *2011 IEEE Fifth International Conference on Semantic Computing*, 2011.

[30] M. Dunlop, S. Groat, and D. Shelly. Goldphish: Using images for content-based phishing analysis. In *2010 Fifth International Conference on Internet Monitoring and Protection*, 2010.

[31] Selenium WebDriver. http://www.seleniumhq.org/projects/webdriver.

[32] W3C DOM4. W3C Recommendation 19 November 2015. http://www.w3.org/TR/dom/#mutationobserver.

[33] Adblock Warning Removal List. https://easylist-downloads.adblockplus.org/antiadblockfilters.txt.

[34] Anti-Adblock Killer List. https://github.com/reek/anti-adblock-killer/blob/master/anti-adblock-killer-filters.txt.

[35] T. Mitchell. *Machine Learning*. Mc-Graw-Hill, 1997.

[36] Other Supplementary Filter Lists and EasyList Variants. https://easylist.to/pages/other-supplementary-filter-lists-and-easylist-variants.html.

[37] V8 JavaScript Engine. https://code.google.com/p/v8/.

[38] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *USENIX Security Symposium*, 2011.

[39] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In *USENIX Security Symposium*, 2013.
[40] Esprima Javascript Parser. http://esprima.org.
[41] BlockAdBlock. https://github.com/sitexw/BlockAdBlock.