CV-INSPECTOR: Towards Automating Detection of Adblock Circumvention

Hieu Le
University of California, Irvine
hieul@uci.edu

Athina Markopoulou
University of California, Irvine
athina@uci.edu

Zubair Shafiq
University of California, Davis
zubair@ucdavis.edu

***Abstract*—The adblocking arms race has escalated over the last few years. An entire new ecosystem of circumvention (CV) services has recently emerged that aims to bypass adblockers by obfuscating site content, making it difficult for adblocking filter lists to distinguish between ads and functional content. In this paper, we investigate recent anti-circumvention efforts by the adblocking community that leverage custom filter lists. In particular, we analyze the anti-circumvention filter list (ACVL), which supports advanced filter rules with enriched syntax and capabilities designed specifically to counter circumvention. We show that keeping ACVL rules up-to-date requires expert list curators to continuously monitor sites known to employ CV services and to discover new such sites in the wild — both tasks require considerable manual effort. To help automate and scale ACVL curation, we develop CV-INSPECTOR, a machine learning approach for automatically detecting adblock circumvention using differential execution analysis. We show that CV-INSPECTOR achieves 93% accuracy in detecting sites that successfully circumvent adblockers. We deploy CV-INSPECTOR on top-20K sites to discover the sites that employ circumvention in the wild. We further apply CV-INSPECTOR to a list of sites that are known to utilize circumvention and are closely monitored by ACVL authors. We demonstrate that CV-INSPECTOR reduces the human labeling effort by 98%, which removes a major bottleneck for ACVL authors. Our work is the first large-scale study of the state of the adblock circumvention arms race, and makes an important step towards automating anti-CV efforts.**

I. INTRODUCTION

The widespread adoption of adblocking has threatened the advertising-based business model of many online publishers [20]. In response, publishers have deployed anti-adblockers that detect adblockers and force users to either disable their adblockers or sign up for paid subscriptions [11], [55], [67]. However, anti-adblocking has not proven very successful: adblockers can often hide anti-adblocking popups [42], [50], [78] or users mostly choose to navigate away [20], [56]. Some publishers have resorted to outright circumvention of adblockers. There are now dedicated third-party *circumvention (CV) services* that help publishers re-insert ads by bypassing adblockers. Examples include AdThrive [12], AdDefend [7], and Publica [59]. These CV services are different, and more advanced, than anti-adblockers. While anti-adblockers generally initiate a dialogue with users [66], CV providers try to sneak ads without giving users any notice or choice [44], [61], [62]. More specifically, CV services re-insert ads by evading filter lists [35], such as the community-driven EasyList (EL) [68], used by adblockers to block ads [13], [22], [26], [53].

The adblocking community has taken notice of the aggressive circumvention tactics used by CV services. Most notably, Adblock Plus (ABP) [5] established a dedicated anti-circumvention (anti-CV) effort that is centered around a new dedicated filter list, the *anti-circumvention list (ACVL)*, to counter these CV services [4], [36], [45]. ACVL supports an extended syntax with advanced capabilities, such as to hide DOM elements based on a combination of CSS styles and text, beyond the simpler rules supported by EL [47]. Concurrently with ABP, other adblockers, such as uBlock Origin [71] and AdGuard [9], also incorporate similar advanced anti-CV filter rules [8], [10], [39], [69]. Similar to other adblocking filter lists [31], [68], anti-CV filter rules are curated manually based on crowdsourced user feedback. However, ACVL is curated primarily by a small set of expert list authors instead of the broader community that supports EL. Thus, a key challenge faced by the ACVL curators is keeping up with the fast paced nature of CV services [30]. Our measurements show that the updates to ACVL are made 8.7 times more frequently as compared to EL. Another challenge is that anti-CV efforts are in the public domain, which gives CV providers the opportunity to monitor anti-CV efforts and adapt their evasive tactics accordingly.

To address these challenges, we introduce CV-INSPECTOR, an automated approach to detect whether a site employs adblock CV services. CV-INSPECTOR includes (i) an automated data collection and differential execution analysis for a list of sites of interest; (ii) an algorithm for prioritizing and expediting ground truth labeling; and (iii) a supervised machine learning classifier using features that capture obfuscation of web requests and HTML DOM by CV services. We evaluate CV-INSPECTOR using two real-world data sets. First, we consider the top-20K sites and show that CV-INSPECTOR is able to accurately detect whether or not a site employs circumvention. In the process, we uncover several new sites (including news publishers, adult sites, and niche lower-ranked sites) that successfully employ third-party CV services. Second, we apply CV-INSPECTOR, with ACVL loaded, on a set of sites that are continuously monitored by ABP, and find that some of them successfully evade anti-CV filters. More importantly, our results show that CV-INSPECTOR can reduce human labeling efforts by 98%, which is a major step in scaling the effort to combat circumvention. To the best of our knowledge,

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24055
www.ndss-symposium.org

this work presents the first large-scale systematic analysis of adblock circumvention on the web. It provides tools [72] that can significantly automate circumvention detection and monitoring, thus helping to prioritize the efforts of expert ACVL curators, which is a major bottleneck in this arms race.

The outline of the rest of the paper is the following. Sec. II provides background of adblock circumvention and related work. Sec. III provides a longitudinal characterization of the anti-CV filter list and highlights pain-points and bottlenecks. Sec. IV presents the design and evaluation of the CV-INSPECTOR methodology, including the description of the automated web crawling, the differential analysis, the machine learning classifier and feature engineering. Sec. V applies CV-INSPECTOR for two different applications: discovering sites that employ CV services in the wild and monitoring sites that are known to employ circumvention to reduce human labeling efforts. Sec. VI concludes with a discussion of potential impact, limitations, and future directions.

II. BACKGROUND & RELATED WORK

Adblockers rely on filter lists to detect and counter ads. Rules in these filter lists are manually curated by volunteers based on crowdsourced user feedback [13], [73]. Filter rules can block network requests to fetch ads using hostname or path information. In addition, they can hide HTML elements of ads using class names or IDs. As adblocking has gone mainstream [20], publishers have undertaken various countermeasures that can be divided into three categories.

A. Whitelisting

Some adblockers allow whitelisting of ads if they conform to certain standards. The Acceptable Ads program [1] whitelists ads if they are not intrusive based on their placement, distinction, and size. ABP and a few other adblockers currently implement the Acceptable Ads based whitelist. The Better Ads Standard [18], by the Coalition for Better Ads, prohibits a narrower set of intrusive ad types such as pop-up ads and large sticky ads. Google's Chrome browser blocks ads on sites that do not comply with the Better Ads Standard, and whitelists ads on the remaining sites [25]. However, whilelisting is not a silver bullet for publishers. First, it is not supported by many popular adblockers such as uBlock Origin and the Brave Browser. Second, some adblockers, such as ABP, require large publishers to pay a fee to be whitelisted. Publishers may also have to pay a fee to ad exchanges, such as the Acceptable Ads Exchange, to serve acceptable ads.

Prior work has investigated the evolution and impact of ad whitelisting. Walls et al. [74] studied the growth of the Acceptable Ads whitelist over the years and showed that it covers a majority of the popular sites. They also reported that the whiltelisting process is flawed because "acceptable" ads are often disliked by users due to their intrusiveness and misleading resemblance to page content. In fact, the whitelisting of deceptive ads from content recommendation networks such as Taboola and Outbrain [17] has been quite controversial [3]. Pujol et al. [60] showed that most ABP users do not opt-out of the Acceptable Ads whitelist despite these issues. Merzdovnik et al. [46] showed that ABP blocked the least amount of ads as compared to other adblocking tools because of whitelisting.

B. Anti-adblocking

Many publishers deploy anti-adblockers that use client-side JavaScript (JS) to detect adblockers based on whether ads are missing. Fig. 1(a) illustrates the workflow of anti-adblocking. The logic is implemented by a client-side JS that first detects whether an ad is missing by measuring the ad's display properties or other dimensions. Then, it displays a popup either warning users to disable their adblockers or a paywall asking them to sign-up for paid subscriptions.

Third-party anti-adblocking services [19], [41], [56] are used by many news publishers such as the Washington Post and Forbes. Nithyanand et al. [52] manually analyzed JS snippets to characterize anti-adblockers. Mughees et al. [50] trained a machine learning classifier to detect anti-adblockers based on HTML DOM changes. These early studies showed that hundreds of sites had started deploying anti-adblockers.

Adblockers counter anti-adblockers using specialized filter lists that use the same syntax as the standard EL. These filter rules either trick the detection logic of anti-adblockers by allowing baits or hiding the warning message shown by anti-adblockers after detection. Iqbal et al. [42] studied the coverage of these filter lists (*e.g.*, Adblock Warning Removal List) against anti-adblocking. They showed that these filter lists are often slow in adding suitable rules by several weeks or sometimes even months. They further trained a machine learning classifier to detect anti-adblocking JS using static analysis. Zhu et al. [78] proposed a dynamic differential analysis approach to detect and disrupt anti-adblockers. The aforementioned countermeasures have proven reasonably successful against anti-adblockers. Moreover, the warning messages shown by anti-adblockers have proven to be of limited benefit [23], [63]. About three-quarters of surveyed users said that they would simply leave the site instead of disabling their adblocker [56].

C. Circumvention

Publishers have recently started to manipulate the delivery of ads on their site to outright circumvent adblockers. Circumvention techniques can be broadly divided into two categories:

Cloaking-based Circumvention. Publishers route ads through channels that adblockers do not have visibility into due to bugs or other limitations. For instance, advertisers used WebSockets to circumvent adblocking extensions in Chrome because of a bug in the WebRequest API that is used by extensions to intercept network requests [16]. More recently, advertisers have used DNS CNAME to disguise HTTP requests to advertising and tracking domains as first-party requests [27], [28]. However, cloaking-based circumvention is not long-lasting because it is neutralized once the bug is fixed. For example, Bashir et al. [16] showed that WebSockets-based cloaking was rendered ineffective when Chrome patched the WebRequest bug [57]. Moreover, cloaking is typically not effective against browsers with built-in adblocking because they are not constrained by the extension API used by adblocking extensions. Thus, we do not focus on cloaking-based circumvention in our work.

Obfuscation-based Circumvention. Publishers obfuscate their web content (e.g., domain, URL path, element ID) to evade filter rules used by adblockers [75]. In contrast to cloaking-based approaches, obfuscation-based circumvention

Filter Types	EL	ACVL	Example	Purpose
Web Request Blocking	✓	✓	||a.com/ads/*/images$script	Blocks web requests matching domain, path and script type
Element Hiding	✓	✓	||a.com##.ad-container	Hides all elements matching class name
Advanced JavaScript Abortion	✗	✓	||a.com#$#abort-on-property-read EX, \|\|a.com#$#abort-on-property-write EX	Stops JS execution from reading or writing to window.EX
Advanced Element Hiding	✗	✓	||a.com#$#hide-if-contains-visible-text /Sponsor/	Hides all elements containing Sponsor text

TABLE I. OVERVIEW OF SIMPLE (USED BY EASYLIST OR "EL") AND ADVANCED (USED BY EL AND ACVL) FILTER RULES. ONLY THE ADVANCED FILTER RULES CAN STOP THE EXECUTION OF JS AND TAKE INTO ACCOUNT THE VISIBILITY OF CONTENT WHEN BLOCKING ELEMENTS.

is powerful because it exploits the inherent weaknesses of filter rules — namely that filter rules must be precise when targeting what to block (i.e., to avoid false positives) and that they are slow to adapt (i.e., filter rule updates). Furthermore, obfuscation-based circumvention can allow publishers to seamlessly continue programmatic advertising that is financially more lucrative for publishers than anti-adblocking.

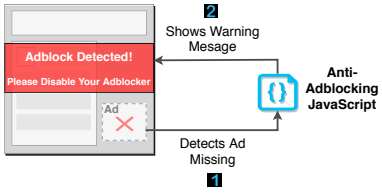
In this work, we focus on obfuscation-based circumvention. Fig. 1(b) illustrates its general workflow: (1) JS detects whether an ad is missing; (2) if an ad is found to be missing, then an obfuscated web request is sent to a CV server; (3) the CV server de-obfuscates the request and relays it to the corresponding third-party ad servers or the publisher's ad server to attain the new ad; (4) the CV server obfuscates the ad content and sends it back to the browser; (5) JS rebuilds the ad content into DOM elements; and (6) it re-injects the new ad at a desired location.

Alrizah et al. [13] anecdotally showed that EL is ineffective at countering obfuscation-based circumvention. More recently, Chen el at. [22] found that about one-third of advertising and tracking scripts are able to evade adblocking filter rules due to URL and other types of obfuscation. To the best of our knowledge, prior work does not provide large-scale characterization of adblock circumvention or automated circumvention detection in the wild.

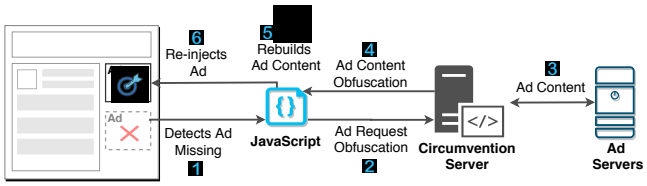
Anti-Adblocking vs. Circumvention. Fig. 1 compares anti-adblocking and adblock circumvention. Both approaches share the first step, detecting whether an ad is missing. Patently, this is necessary for anti-adblocking. However, for circumvention, it is not a required step but rather a choice that publishers select to minimize the cost of using CV services.

After the first step, their subsequent steps differ. As shown in Fig. 1, different from anti-adblocking, circumvention involves a series of additional steps at the server-side to bypass filter rules and re-inject ads in the client-side browser. Thus, circumvention is a more intricate process. It must deal with the process of attaining new ad content and where to place them on the page. Recall that it must do this without disrupting the user experience while also evading filter rules. The complexity of circumvention is further denoted by adblockers implementing new advanced filter rules, such as aborting JS execution, to adequately combat it. This is further explored in Sec. III.

As noted before, anti-adblocking and circumvention both aim to affect adblock users only: thus, making differential analysis a suitable technique to detect them. Intuitively, differential analysis endeavors to capture fundamental characteristics of anti-adblocking or circumvention. For instance, with regards



(a) Anti-Adblocking



(b) Obfuscation-based Circumvention

Fig. 1. **(a)** (1) If JS detects that an ad is missing; (2) it shows a popup window asking the user to disable the adblocker, pay for a subscription, or whitelist the site. **(b)** (1) If JS detects that an ad is missing; (2) it sends an obfuscated ad request through a CV server; (3) the server retrieves the new ad from an ad server; (4) the server obfuscates it before sending it back to the browser; (5) JS rebuilds the ad content into DOM elements; and (6) re-injects the ad back onto the page.

to Fig. 1(a), prior work [50], [78] sought to detect the action of step 1 and whether the popup of step 2 was displayed to the user. Note that the outcome of anti-adblocking does not involve ads. On the other hand, our work identifies characteristics of circumvention, described in Fig. 1(b), within actions of steps 2 and 4, and whether ads were displayed as a result of step 6.

However, the differential analysis method proposed in prior work to detect anti-adblockers cannot be directly used to detect adblock circumvention. For example, Zhu et al. [78] conducted differential analysis of JS execution to find branch divergences due to anti-adblocking. This technique, if used as is, would incur false positives when a site is able to re-insert ads but *unsuccessfully* displays them due to filter rules hiding the ad element. More specifically, the circumvention approach illustrated in Fig. 1(b) would exhibit a branch divergence at the first step of detecting missing ads, which would be incorrectly considered a positive label (successful circumvention). While CV-INSPECTOR also uses a differential analysis approach that involves loading a page with and without adblocker, it does not aim to capture branch divergences due to anti-adblocking. As we discuss later, CV-INSPECTOR conducts differential analysis of web requests, DOM mutations, and other features to be able to distinguish between *successful* and *unsuccessful* circumvention of adblockers.

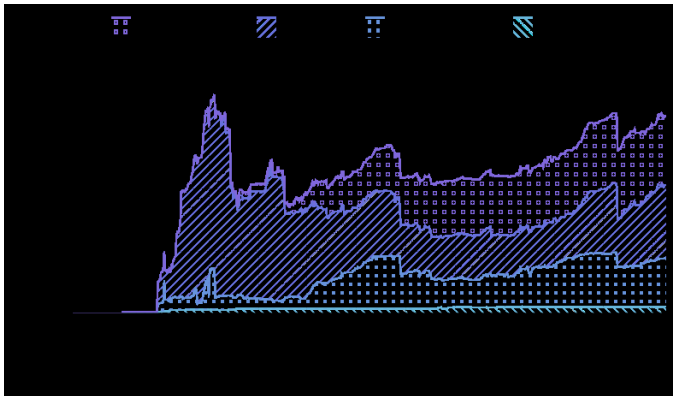


Fig. 2. **Anti-circumvention List Over Time.** This shows how filter rules from ABP's ACVL have evolved from May 2018 to May 2020 and categorizes them by filter types.

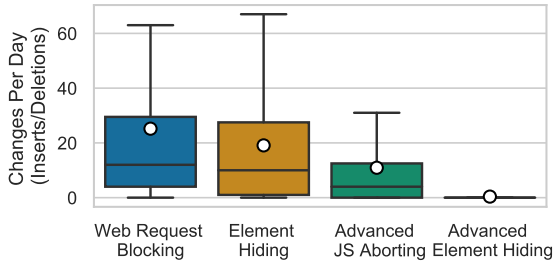


Fig. 3. **Commits by Filter Type.** A boxplot of commit changes from 2018 to 2020 and categorized by filter types for ACVL. The horizontal lines within the boxes represent the median, while the white circles represent the mean.

III. State of Anti-Circumvention

The adblocking community is increasingly wary of circumvention. Most notably, ABP recently started a dedicated filter list, ACVL, to combat circumvention [43]. The filter list is enabled by default in ABP to help block "circumvention ads." This anti-CV list has two key advantages over the standard EL. First, it allows ABP to have full control over filter rule design and management, including pushing the updated rules at a higher frequency (*e.g.*, every hour as opposed to every four days for EL) and without community consensus. Second, it supports advanced filter rules with enriched syntax and capabilities, which are not supported by the standard EL, specifically to counter CV services [38].

A. Filter Rules Overview

Filter rules can be either simple or advanced. Table I provides examples and their compatibility with EL and ACVL. We refer to EL types of rules as *simple filter rules*: they can block web requests by matching domains and paths or hide DOM elements by targeting CSS styles or content.

ACVL deploys additional *advanced* rules to combat circumvention: these can abort the execution of JS or hide DOM elements based on computed styles and visibility of content [33]. For example, if "EX" is an JS object that holds circumvention code, then "||a.com#$#abort-on-property-read EX" can block any JS that accesses it. Creating the rule often involves reverse engineering the code to identify that "EX" holds circumvention related code. Furthermore, a filter rule like "||a.com#$#hide-if-contains-visible-text /Sponsor/" can hide any element containing the visible text "Sponsor." Notably, this differs from simple element hiding because the simple rule only takes into account the existence of text content and not whether it is displayed to the user.

B. Analysis of the Anti-circumvention List (ACVL)

Evolution of Anti-circumvention Rules. We consider the commit history of ACVL by using its GitHub repository and rebuild the list's filter rules for each day from May 2018 to May 2020 [4]. Fig. 2 shows the evolution of the list since its inception in May 2018. The list grows rapidly near the end of September 2018 and peaks at 700 filter rules in November 2018. We see the overwhelming usage of element hiding over other filter types such as web request blocking and advanced element hiding. This can be attributed to the fact that advanced element hiding has a large performance cost (with the use of "window.getComputedStyles"), causing filter list authors to use it sparingly. Also, element hiding may have been more effective in 2018 because JS aborting was not introduced until mid-November of that year [40]. Due to the over dependency on element hiding, we see that until February 2019, ABP could not prevent the loading of circumvented ads but rather only hide them from the user. Moreover, we see a large drop in element hiding rules (~300 filter rules removed) from November to December 2018. When inspecting the commit changes of that drop, they appear to be cleaning up old filter rules for Czech and German sites [64], [76]. In particular, we find that many element hiding rules are used to target only 13 sites (*e.g.*, novinky.cz and super.cz). This is a downside to element hiding: it must target specific elements resulting in a large number of rules to cover ads even for one site.

Next, we observe that the introduction of JS aborting rules in mid-November 2018 triggers a change in the filter type usage within ACVL. First, the popularity of JS aborting rules denotes its effectiveness against circumvention. Second, it reduces the ACVL's dependency on element hiding because JS aborting prevents ad reinsertion, which results in fewer ad elements to hide. Consequently, this also increases the popularity of web request blocking. This can be due to two factors: (1) once filter list authors understand which JS employs circumvention, they can better find a way to block the script entirely; and (2) CV services rely more on web request obfuscation during that period. Thus, from late 2018 to 2020, we see that the three filter types were used almost equally.

Frequency of Updates. For 2019, which denotes ACVL's first complete year, Fig. 2 shows that the number of filter rules has stabilized within the range of 400 to 500 rules. This contrasts with EL's constant growth, which increases at approximately 8K rules per year [73]. However, the daily modifications to ACVL remains high. To explore this notion, we review the changes of all commits within a day by using "git diff" and parse each change to categorize then into filter types. Fig. 3 reveals the spread of changes per day (defined as number of inserts and deletions) for each filter type within ACVL. We find that the medians of changes are 12, 10, and 5 for web request blocking, element hiding, and advanced JS aborting,

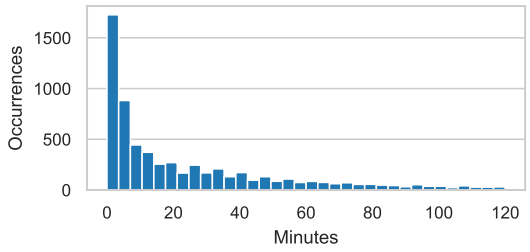


Fig. 4. **Time between Commits.** The time between commits for ACVL is most frequently within 4 minutes while the average is 2.3 hours.

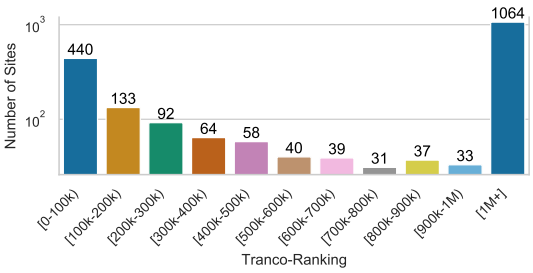


Fig. 5. **Tranco-ranking of ACVL.** Sites extracted from ACVL and their corresponding Tranco-ranking. We see that there is low coverage of sites for circumvention from ranking 100k to one million. Note that about half of the sites do not appear in the Tranco top one million list (labeled as 1M+).

respectively. The median for advanced element hiding remains at zero due to its infrequent changes. Moreover, the frequency of commits persists at a high rate, as indicated by the time between commits, reported in Fig 4: it is commonly within 4 minutes. The average time is 2.3 hours, which is about 8.7 times more frequent that EL's 20 hours [73]. This highlights the accelerated arms race between publishers and adblockers within the circumvention space.

Publishers that Employ Circumvention. Upon further inspection, we find that commits are generally modifications to existing rules. For web request blocking, curators typically change URL components (*e.g.*, subdomains and paths) or resource type within the rule. For both simple and advanced element hiding, they often modify class names, IDs, styles, and DOM structure. For advanced JS aborting, they change the name of the JS object that the rule is targeting.

To evaluate publishers that cause these frequent commits, we identify domains that appear in both insert and deletion lines within a commit. We discover that the top two sites, reuters.com and quoka.de, have triggered 671 changes to rules over a period of 17 months and 269 changes within 18 months. Overall, the top-10 websites that give filter list authors the most trouble have an average ranking of about 10K [58]. However, when considering the top-30, the average ranking is 189K, which can be explained by the fact that many of the sites are from Germany, where most of the ACVL authors reside.

Coverage of ACVL. Next, we investigate the coverage of ACVL on the web, which has not been previously explored. We extract the sites that are specified in the filter rules and map them to their corresponding Tranco-ranking [58] in Fig. 5. First, we note that there are about 927 sites that employ circumvention within the Tranco top one million sites, which denotes the low prevalence of circumvention; perhaps, due to the cost of CV providers. Second, surprisingly, we see that ACVL covers about 1064 sites that are beyond the one million (1M) Tranco-ranking, more than twice the amount of coverage when compared to the top-100K.

Furthermore, we see low coverage numbers for the range of rankings between 100K to 1M. We can deduce that the ACVL may lack coverage in two ways. First, advanced rules must specify which sites to target while simple rules can be website agnostic. Second, we previously saw that ACVL's number of rules has stabilized — showing that ABP is more focused on combating circumvention from a few known sites rather than discovering new sites that employ circumvention. In addition, while EL authors receive help from the community through forums that have up to 23K reports over a span of nine years [13], ACVL authors rely on submitted GitHub issues, with a current total of 379 issues over a span of two years [4]. Thus, significant manual work (*e.g.*, updating rules and discovering new circumvention sites) falls onto the filter list authors.

Takeaways. The number of ACVL filter rules has stabilized in contrast to EL. This can be attributed to two factors: (1) ABP's focus on a few known CV providers; and (2) changes within ACVL primarily being modifications to existing filter rules. Thus, the coverage of ACVL is limited due to the focus on modifying rules rather than discovering new circumvention sites. Moreover, the effort to combat circumvention requires significant effort from filter list authors. ACVL has only 14 contributors with three main contributors: wizmak, arsykan, and Milene [6], who commit five, four, and three times on average per day, respectively. These few filter list authors must undertake a huge effort in keeping rules up-to-date.

This motivates our methodology in the next section, which aims at assisting and prioritizing this effort by providing ways to automate detecting successful circumvention in the wild, to monitor the changes in publishers, and to be notified when a site has successfully circumvented the adblocker.

IV. CV-Inspector: Design and Implementation

In this section, we present CV-Inspector for detecting whether a site employs circumvention or not. Fig. 6 presents an overview of our methodology. In Sec. IV-A, we present our instrumentation and automated data collection. In Sec. IV-B, we apply differential analysis to identify data that is indicative of circumvention. Then, in Sec. IV-C to IV-F, we extract features, train, and evaluate our CV-Inspector classifier.

A. Instrumentation and Data Collection

1) How we collect data: Our crawling script takes as input a list of websites for which we collect data. For each page load of a site, we wait for 25 seconds: we denote this as a "page visit." Page load times are commonly less than a minute as they affect the search ranking of sites. As shown in Fig. 6, we visit each site for a total of eight times. As a result, we select 25 seconds to not significantly slow down CV-Inspector, which is inline with prior work [78].

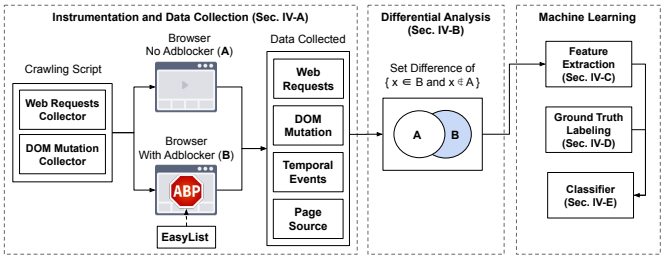


Fig. 6. **CV-INSPECTOR Workflow.** Given a list of URLs, our crawling script will visit each site four times for: (A) "No Adblocker" and (B) "With Adblocker." With each visit, we collect web requests, DOM mutation events, temporal events (*e.g.*, timestamps and blocked events by the adblocker), and the page source. We take the set difference between the data collected in the two cases, (B)-(A), since websites commonly employ circumvention when adblocker is on. We use the (B)-(A) data to extract most features, train and evaluate our classifier.

No Adblocker vs. With Adblocker. Since websites typically employ circumvention only when an adblocker is present, we utilize differential analysis to obtain insights into circumvention "signals." For each website, we collect data for two different cases: (A) "No Adblocker" and (B) "With Adblocker." For "No Adblocker," we load each site four times and take the union of the collected data in order to capture the dynamic nature of a site because a it can retrieve ads from different ad servers. This is a heuristic but justified choice: we experimented with loading the same page for a varying number of times and found that the number of contacted domains plateaus at four. We will refer to these "four page visits" per case, throughout the paper. We repeat the same process for "With Adblocker." In addition, we use ABP and configure it to use EL. We deselect the "Allow Acceptable Ads" option as we want to make sure ads are shown due to circumvention and not because it was whitelisted. Furthermore, this gives sites the best chance to circumvent the adblocker and the best opportunity for us to capture it.

Landing Page and Sub-pages. Our crawling considers both landing pages and sub-pages. This is critical because sites may not employ circumvention in their landing pages but rather wait until the user clicks into a sub-page to show circumvented ads (*e.g.*, maxpark.com). To find a sub-page, we inject JS into the landing page to retrieve all URLs from hyperlink tags. We select the first-party link with the longest number of path segments. We use the intuition that the deeper the user explores the site, the more interested the user is in the content, thus increasing the chance that the site would serve ads. We find that this methodology works well for sites that have articles. To further ensure that we find a sub-page with ads, we ignore informational pages using keywords (*e.g.*, "contact," "login") within the path. To only consider pages with content, we further ignore first-party links that have extensions (*e.g.*, ".tar.gz," ".exe"), to prevent downloading external files.

Automatic Collection. We use Selenium [51], a framework to automate testing of websites, to implement the crawling process. We select Chrome (version 78) [24] as the browser due to its popularity. As depicted in Fig. 6, we create two Chrome profiles. One profile is for the "No Adblocker" case, where we include the web request extension and DOM mutation extension. The second profile is for the "With Adblocker" case, where we also include the custom ABP extension that only loads EL. In order to have a consistent behavior with ABP, we only use one version of EL and the ACVL from March 13, 2020. Then, we configure Selenium to disable caching and clear cookies to have a stateless crawl. For scalability purposes, we utilize Amazon's Elastic Compute Cloud (EC2) and select the "m5.2xlarge" instance that allows CPU usage without throttling [14]. We create a snapshot out of the setup using Amazon Machine Image (AMI) [15], which allows us to spawn many instances of EC2 for data collection.

2) What data we collect for each page: Next, we describe the types of information we collect for each site. We are interested in how the site changes from "No Adblocker" to "With Adblocker," at four vantage points:

1) Web requests: HTTP incoming and outgoing requests.
2) DOM mutation for nodes, attributes, and text.
3) Time stamps of all events like web requests, DOM mutations, and blocked events caused by ABP (*i.e.*, when a filter rule is matched, see Table I).
4) Page source code of the site (*e.g.*, HTML, text, inline CSS, and inline JS).

We also collect screenshots, which are capped at 1925x3000 to deal with websites that can infinitely scroll. Screenshots are useful as we use them to verify our ground truth in Sec. IV-D. Next, we explain how this collected data can reveal obfuscation-based circumvention employed by the site.

1. Collecting Web Requests. Circumvention providers often randomize subdomains and paths as an obfuscation technique to retrieve new ad content for reinsertion, going beyond simply rotating domains [13], [42], as illustrated in Fig. 1(b). Capturing web requests can help identify this behavior. Examples are provided in Sec. IV-C. We implement a Chrome extension to collect web requests by hooking into the Chrome Web Request API [37]. This API streamlines the flow of web requests into various life-cycle stages that developers can easily subscribe to. Specifically, we hook into "onSendHeaders" to collect outgoing HTTP request headers and "onCompleted" to collect incoming HTTP response headers of successful requests. To collect web requests blocked by ABP, we hook into "onErrorOccurred" and look for status code "ERR_BLOCKED_BY_CLIENT."

2. Collecting DOM Mutation. Fig. 1(b) shows that re-injected ads are often reconstructed in step 5 and may not have the same DOM structure as the originally blocked ads. Capturing how the DOM changes as the page loads can help uncover these particular actions. We build a Chrome extension that uses DOM Mutation Observers [49] to collect DOM changes. The extension compiles events such as new nodes added (*e.g.*, an ad image being added), nodes removed (*e.g.*, a script being removed), attribute changes (*e.g.*, an ad element from height 0 to 280px), and text changes (*e.g.*, anti-adblocker popup text). Furthermore, recall from Table I that an adblocker can do element hiding. We capture this by instrumenting the ABP extension (version 3.7) and hook into methods that hide elements when a filter rule is matched to label the elements with a custom HTML attribute "abp-blocked-element," shown in Listing 1. Since this causes a DOM attribute change, we consider this as part of the DOM Mutation information.

Listing 1. **Page Source Annotations.** Highlighted in blue, attribute "abp-blocked-element" denotes that the adblocker has blocked the element. While attribute "anticv-hidden" means that the img is not visible (not related to the adblocker). All visible images and iframes are labeled with their offsetwidth and offsetheight to give a more accurate representation of the page.

```
<div abp-blocked-element="true">
    <img src="https://ads.com/banner.jpg"/>
</div>
<div class="mobile">
    <img src="https://b.com/logo.jpg"
    anticv-hidden="true"/>
</div>
<iframe src="https://b.com/ad" height="90"
    anticv-offsetwidth="728"
    anticv-offsetheight="90">
</iframe>
```

3. Collecting Temporal Information. Since circumvention is typically a reaction to ads being blocked, timestamps of changes on the page can reveal how adblockers and circumvention code interact with each other. Thus, we record and consider timestamps for web requests, DOM mutation, and blocked events. For completeness, when we consider the ACVL in Sec. V-B, we hook into methods that abort the execution of JS to capture JS blocked events as well.

4. Collecting Page Source with Annotations. We use Selenium to save the page source of the site at the end of the page load time. It gives us information about the state of the site such as the HTML and text, inline CSS, and inline JS. In addition, it contains the annotated elements that are hidden by the adblocker, as shown in Listing 1. Furthermore, since the page source does not provide the actual visibility state of images and iframes, we inject JS to annotate these elements with a custom attribute "anticv-hidden" detailed in Listing 1. We extract all images and iframes and consider the following cases. First, if the element's "offsetParent" is null and its "offsetWidth" and "offsetHeight" are zero: this denotes that the element is hidden due to its parent being hidden. Second, otherwise, we use "window.getComputedStyles," which provides us the final styles that are applied on the element. We consider styles such as "display: none" and "opacity $<=$ 0.1" to see if the element is hidden. Third, we treat elements with width and height of less than or equal to two as hidden. This filters out pixel elements used for tracking. We further use these annotations for feature extraction, as described in Sec IV-C.

3) Tools and Limitations: Using Amazon's EC2 and AMI, our methodology is scalable (*e.g.*, multiple instances can be initiated to fit the problem) and configurable (*e.g.*, number of sub-pages to find, which filter list to load). However, it also has its limitations. First, some sites utilize Cloudflare's protection against web-crawlers where it shows a captcha, which prohibits CV-INSPECTOR from accessing the page. Second, Selenium may not properly produce screenshots, which depends on how body styles are applied. We address this limitation by first checking whether the height of the body is zero. If so, then we check the next immediate child element of the body to see if it has a height to capture, and so on. Third, when discovering sub-pages, we do not consider links from non-hyperlink tags or if the site is utilizing JS to redirect users upon a click. Finally, recall that we wait for 25 seconds during each page visit, which might miss some behavior on sites that need longer to load. This is a parameter to tune: longer crawling times is possible at the expense of slowing down CV-INSPECTOR.

4) Data sets: We apply our methodology and collect different data sets, summarized in Table II, which we then use for different purposes throughout the paper. For each of these data sets, we start from a list of URLs, apply the methodology described earlier in this section, and we collect the four types of information, referred to as "collected data" in Fig. 6: web requests, DOM changes, temporal information, and page source with annotations. The top three data sets in Table II are collected using our methodology based on a given list of sites: ACVL sites, Tranco's most popular sites and Adblock Plus Monitoring. The first two are publicly available.

ACVL has been extensively discussed in Sec. III and includes sites that currently employ, or had employed in the past, CV services; we use this list to find positive samples. We use Tranco ranked sites in two ways. First, since circumvention is hard to find, we use the Tranco top-2K sites within our ground truth data set (GT) to ensure that it covers popular sites. Second, we use the Tranco-20K data set (which excludes the top-2K) to test our classifier on popular sites that matter to users. The third data set, internally maintained by ABP, contains sites that employed circumvention at some point and ABP continuously monitors them to see if ACVL is still effective on them. We refer to sites that are closely monitored by adblockers as "sites of interest." Generally, this means that the sites affect a large portion of adblock users (*i.e.*, in terms of popularity) or that the sites have caused users to submit feedback about them.

The bottom part of Table II summarizes our three original crawled data sets that we use for training and evaluating our classifiers in Sections IV-D, IV-E, and V.

B. Differential Analysis

1) Set Difference: Our intuition is that behavior observed when an adblocker is used ("With Adblocker") is different from the behavior observed when there is no adblocker ("No Adblocker"). This is likely due to CV services being triggered. Recall from "No Adblocker vs. With Adblocker" of Sec.IV-A1, that we need to account for the dynamic nature of websites. Therefore, first, we take the union of the data sets collected across all four page visits in each case. Then, we take the difference of the two union sets ("With Adblocker" minus "No Adblocker"). Next, we elaborate on what differences we examine for each of the four types of data collected.

First, for web requests, circumvention services can serve content behind first-party domains. Therefore, we cannot simply do a set difference on the domain level for web requests, which would eliminate the presence of the circumvented ads. Instead, we do a set difference based on the fully qualified domain and its path while disregarding the query parameters. Second, for for DOM mutations, we create a signature for each event based on the element's attribute names, tag name, parent tag name, and sibling count. We do not depend on the value of attributes because they can be randomized [13], which would introduce more unrelated events to circumvention. Instead, we rely on the length of the value within our event signature. For a

Data Set Name	List of Sites Crawled	# Pages & Sub-pages
Obtained by crawling a given list of sites		
ACVL sites	Sites extracted from ACVL (public [4])	3K
Tranco	Most popular sites (top-20K) at tranco-list.eu (public [58])	32K
Adblock Plus Monitoring	Sites that ABP monitors (maintained and provided by ABP)	360
Derived from ACVL & Tranco, used for ML training & testing		
Candidate for labeling (CL)	ACVL ∪ Tranco top-2K	6.2K
Ground Truth (GT)	Subset of sites from CL that are inspected and labeled (positive or negative) for circumvention	2.3K
Tranco-20K	Tranco top 2K-20K (excluding the top-2K used in CL)	29.3K
Ground Truth Positives (GTP)	Subset of GT with only positive labels	700

TABLE II. DATA SET SUMMARIES AND TERMINOLOGY USED THROUGHOUT THE PAPER. EACH OF THE ORIGINAL DATA SETS IS OBTAINED BY CRAWLING THE CORRESPONDING LIST OF SITES (AND SUB-PAGE) AND COLLECTING ALL 4 TYPES OF DATA (WEB REQUESTS, DOM CHANGES, TEMPORAL, AND PAGE SOURCE).

simple example, if the element is " ," we would consider it as "div_class9." Third, for temporal information, we first extract features per visit then average them within their respective cases, then we apply the set difference. Fourth, for page source, we do a set difference based on words for text differences. For example, a text change event with an old value of "Please subscribe to our content" and a new value of "Please disable your adblocker to view our content," will result in a set difference of "subscribe, disable, your, adblocker, view."

2) Cleaning the Data: Recall that we load each site four times to capture its dynamic content. A side effect is that we end up with data (*e.g.*, web requests and DOM mutations) that is not necessarily related to circumvention, and can be due to tracking, discernible non-ad resources, dynamic content, etc. We filter these out before extracting features for circumvention. First, for web requests, we identify tracking, social, and anti-adblocking requests by applying EasyPrivacy [31], Adblock Warning Removal List [2], Disconnect.Me [29], and uBlock Origin's GetAdmiral [70] filter lists. To filter out the requests, we use Brave's Adblock engine [21], a filter list parser that supports EL-compatible rules. Second, we keep third-party ad resources by looking at ones that have content-length larger than 2 KB and have a max-age (within cache-control headers) shorter than 40 days. We conclude on these numbers by inspecting resources that were blocked by ABP. This gives us a profile about what content-length and max-age ad resources should have. Third, we only consider successful web requests (*e.g.*, HTTP status code 200) and discard the ones that involve redirection, errors, or no content (*e.g.*, HTTP status codes 304, 400, 204). This is because circumvention related web requests should have content such as JSON (that may define ad content) and JS (code to re-inject ads).

Web Request Features	Top
Number of content-types	✓
Entropy of subdomains, paths, query parameters (by content-types and first/third-party)	✓
Number of Mismatches of URL extension and content-type	
Number of Mismatches of loaded resources	
DOM Mutation Features	**Top**
Number of DOM attribute changes (display, class, etc)	✓
Number of DOM nodes removed (iframes, etc)	✓
Number of elements blocked by EL (imgs, iframes, etc)	✓
Number of DOM nodes added (a, imgs, etc)	
Temporal Features	**Top**
Number of blocked events (in first 12sec of page visit)	✓
Number of blocked events (in second 12sec of page visit)	
Average cluster size of DOM mutations over time	
Page Source Features	**Top**
Number of iframes and images in ad positions	✓
Number of distinct words, characters, and newlines	
Entropy of subdomains, paths, query parameters of visible iframes and images contained in hyperlinks (with target or rel attributes)	

TABLE III. SOME OF THE FEATURES USED IN CV-INSPECTOR. THERE WERE 93 FEATURES TOTAL IN THESE 4 CATEGORIES. THOSE MARKED AS "TOP" WERE IN THE TOP-10 MOST IMPORTANT FEATURES IN SEC. IV-E.

Listing 2. **Obfuscated URL Example.** Taken from psychologyjunkie.com, we compare a normal URL with an obfuscated one where subdomains & paths are randomized. Although truncated, the path can reach up to 6K in length. The entropy of the subdomains for the regular and obfuscated URLs are 1.58 and 2.25, respectively. Their first path segments would have entropy of 1.79 and 4.56. As expected, the obfuscated strings have higher entropy.

```
0
1 /* Regular URL */
2 https://cdn.convertkit.com/assets/CKJS4.js
3 /* Obfuscated URL */
4 https://h239rh.lmyiwaakn.com/qO8HqaNP1NUGrt
5 d4qtgA1agJ2JAHpqoDo9QDqqYAptl4qaoF1dZ0...
```

C. Feature Extraction

Next, we describe the features that we extract from the cleaned set difference to capture circumvention. Not all features involve set differences, *e.g.*, blocked events only appear in the "With Adblocker" case. Table III lists the features that we explored and highlights those that ended up being the top-10 most important features. Then, we evaluate those features and explain our intuition of why they can capture the presence of CV services.

1. Web Request Features. One widely used obfuscation technique is to randomize URL components and other features extracted from web requests, resulting in noticeable differences between "No Adblocker" and "With Adblocker" cases. Listing 2 shows a comparison between a regular URL and an obfuscated one by circumvention. To capture this randomization, we treat URL components, such as subdomains and paths, as strings, and we calculate their Shannon entropy, based on the frequency of each character occurring in the string. The idea is that randomized strings will have higher entropy. An illustrative example is shown in Listing 2. As expected, the obfuscated strings have higher entropy for both

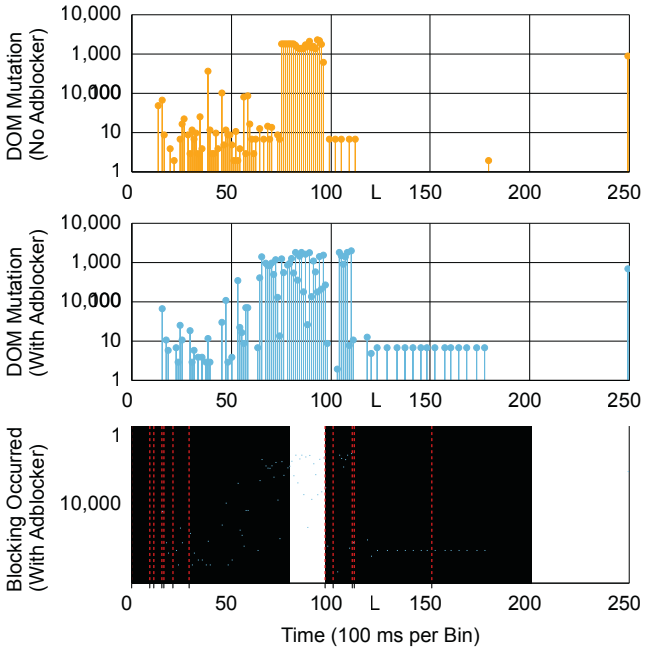


Fig. 7. **Example of Temporal Features.** We show the number of DOM mutations (spikes) over time for "No Adblocker" and "With Adblocker" (with the corresponding blocked events). We define a cluster of activity as consecutive spikes (no more than one bin apart) and the cluster size as the number of bins that it spans. The top figure shows the "No Adblocker" case, which has 9 clusters with an average cluster size of 8.33. In the middle figure, we show the "With Adblocker" case, which has 22 clusters with an average size of 3.86. In the bottom figure, the dashed vertical lines represent whether blocking events occurred. The majority of blocking happened within the first 12 seconds when compared to the remaining time (*e.g.*, 11 events *vs.* 1 event).

subdomains and paths. We further split web requests up into first-party and third-party sets. In addition, we count the the number of different content-types extracted from their response headers. Furthermore, we look at mismatch cases like when a web request ends with a ".jpg" extension but its content-type is "application/javascript." Also, we look at whether a particular path loads different numbers of resources. For instance, when a path "a.com/images/" loads 10 images with the "No Adblocker" case but then loads 15 images for the "With Adblocker" case.

2. DOM Mutation Features. DOM mutation features can uncover behavior such as when new ad-related elements are added. For nodes being added and removed, we focus on element types that can be associated with ads such as "<a>," "<imgs>," and "<iframes>." For attribute changes, we focus on changes such as the class attribute, visibility styles like display and position, and the height of the element. Moreover, we count the number of DOM attribute changes that involve "abp-blocked-element," which denotes the number of elements blocked by EL.

3. Temporal Features. We expect that a site would exhibit different behavior (events) over time when employing circumvention, as depicted in Fig. 1(b). Therefore, we examine the timing of events to extract temporal features. Fig. 7 details how we capture differences in DOM mutations over time by utilizing spikes, clusters, and cluster sizes. By considering the cluster size, we can identify bursts of DOM mutations and how prolonged they are. For "With Adblocker," we see fewer DOM mutations within the first five seconds, perhaps due to many blocked events in the beginning. However, after that, we see more bursts of DOM activity; notably, within the 12–18 seconds that are not present in the "No Adblocker" case. This is captured by the smaller average cluster size. Interestingly, this turned out not to be a top feature. We deduce that this is because not all circumvention techniques cause large DOM mutation changes. For instance, a site can load in a static ad and use a simple ad structure, as shown in Listing 3. We further discuss circumvention techniques in Sec. V-A2 and Table VI.

Listing 3. **Simple Ad Structure.** An example of a simple ad structure that can be used during ad re-insertion instead of an iframe.

```
0
1  <a href="https://www.512xiaojin.com"
2      target="_blank" rel="nofollow">
3      <img src="https://1.bp.blogspot.com/
4      -YTj2YjT6ODw.gif" />
5  </a>
```

Since blocked events (*i.e.*, any matching of filter rules in Table I) can happen for sites that do not employ circumvention, we want to investigate whether the timing of blocked events can signal circumvention. Recall that we visit each page for 25 seconds, a parameter value chosen for reasons explained in Sec. IV-A1. We compute the number of blocked events in the first or second 12 seconds of the page visit. We initially thought the second half would be a differentiating feature, as the page would exhibit the action of re-injecting ads and the adblocker would then once again block those ads. However, we observed that the first half was more important, as shown in Fig. 7. This may be because loading ads is a priority, leading to the blocked events happening in the beginning of the page load. Also, filter rules often aim to stop circumvention at the earliest possible point. Ultimately, adblockers are more aggressive against sites with circumvention, and therefore, cause more blocked events.

4. Page Source Features. Page source features characterize the state of the site at the end of our page visit time. These features convey whether circumvention was successful by identifying possible ads that are still visible on the page. We discover that circumvention exhibits behavior such as altering the DOM structure of the ad to circumvent adblockers, while re-injecting the ads back to specific, and often the same, locations.

First, we target specific DOM structures that hold ads such as images or iframes. For images, we select those that are contained by hyperlink elements ("<a>") with attributes "target" and "rel," as shown in Listing 3. The "target" attribute defines how the browser behaves after a user clicks on the link such as opening up in a new window or tab. The "rel" attribute defines the relationship between the current page and the outgoing link. We can use this to infer that if the outgoing link is also third-party, then it is likely to be an ad.

Second, we identify possible ad locations that can be utilized for re-injection. We use the "No Adblocker" page source and extract all iframes. We then dynamically create CSS selectors for the iframes, specifying at least three levels of ancestors to make sure the selector is specific enough. We then use these selectors on the page source of the "With Adblocker"

side and count the number of images or iframes that remain. To deal with sites that randomly alter their element attributes, we do a second search (when the first search does not match any elements) with more generic selectors by looking at the existence of attributes and not the values of them. For instance, a selector of "div > div[opacity='1'] > div[class='rerejhf']" will turn into "div > div[opacity] > div[class]."

For both of these cases, we make sure that iframes and images are visible and not hidden by the adblocker or pixel-size used for tracking. This is possible by using our annotations from Listing 1 to ignore elements that are invisible to the user.

D. Ground Truth Labeling

Let us revisit Sec. IV-A4 and discuss how we use the original data sets, shown in the top two rows of Table II, to create our GT data set, for training our classifier.

Why Positive Labels are Important. A major challenge for our GT data set is that positive samples (*i.e.*, sites that successfully employ circumvention) are rare and hard to find. First, there are simply not many sites that employ circumvention today. For example, in Fig. 5, only 927, out of the top one million Tranco sites, utilize circumvention. Second, we define positive labels as not only attempting circumvention, but also successfully circumventing adblockers, which further reduces their number. Conversely, negative labels are easy to discover because they correspond to sites that do not attempt circumvention and to sites that do, but are unsuccessful at evading the adblocker. For instance, see the imbalance in Table V. Furthermore, human inspection and labeling of sites is a labor-intensive process. To resolve these challenges, we devise a methodology that reduces human labeling efforts while finding many positive labels.

Candidates for Labeling (CL). We start from a list of URLs that we consider candidates for labeling: this includes 2K domains extracted from the ACVL, as described in Sec. III-B, and popular Tranco top-2K sites. Domains extracted from ACVL are not guaranteed to have positives, because compatible rules from ACVL can be transferred to EL, thus EL can deal with circumvention for some sites. Furthermore, since Fig. 5 reveals that many ACVL domains are beyond the one million ranking, we also consider the Tranco top-2K sites as candidates for labeling, to include more popular sites of interest. We then crawl the sites using our data collection methodology, depicted in Fig. 6, and end up with approximately 6.2k sites (including sub-pages) for our CL data set.

Labeling Each Site. We label each site, in our CL data set, as either successful circumvention (positive label) or not (negative label). We capture a screenshot each time we visit a page and depend on them to label our sites. Our labeling methodology is as follows. First, we open up screenshots from "No Adblocker" case and identify where ads are shown. Then we open up screenshots from "With Adblocker" and compare to see if the ads are removed. If an ad is still visible, we label the site as positive; otherwise, we label it as negative. Second, there may be "suspicious content." For instance, ads can look similar to page content rather than common ads, either because they lack transparency (*e.g.*, not annotated by "Advertisement" or "Sponsored"), or they may be closely related to the site content. Fig. 8 illustrates an example of such "suspicious content": gaming ads are displayed for a gaming site, "gamer.com.tw," which makes it difficult to tell whether they are ads or first-party content. To settle these cases, we visit the site on our Chrome browser and set up ABP with the same configuration (settings and filter lists) as our data collection. This allows us to further verify whether the content was an ad by looking at the outgoing link or test it out by clicking on it. If the content is indeed an ad that goes to a third-party site, we label it as positive. In our GT data set, we encountered "suspicious content" only 69 out of 2321 times, thus making it a corner case.

(a) gamer.com.tw: "No Adblocker" case

(b) gamer.com.tw: "With Adblocker" case

Fig. 8. **Example of "Suspicious Content."** This gaming website shows suspicious content on the right sidebar outlined in red. Note that the three small images change between the (a) "No Adblocker" and (b) "With Adblocker" sub-figures. Although the content may look like ads, it could also be benign content related to gaming. Using a browser, we looked at their outgoing URLs and observed that the two smaller images for Tera Awaken and EOS are ads, while the third image links to a first-party page. Since there are still ads displayed in (b) "With Adblocker," we label this example as a positive label.

As described, our labeling methodology relies on using screenshots. Recall from Section IV-A1 that for a given site, we visit it four times for the "No Adblocker" and "With Adblocker" cases, which corresponds to four screenshots for each case. An alternative approach to labeling would be to use a browser to check the site, which can produce higher quality labels. For instance, the browser allows us to view the entire site as opposed to the limited height of the screenshots, which is capped at 3000px to deal with infinitely scrolling sites. However, the browser approach increases the human labeling efforts. Screenshots offer an attractive compromise: they allow us to quickly compare the four page visits of "No Adblocker" and "With Adblocker" with each other, without setting up our browser and loading the sites four times per case.

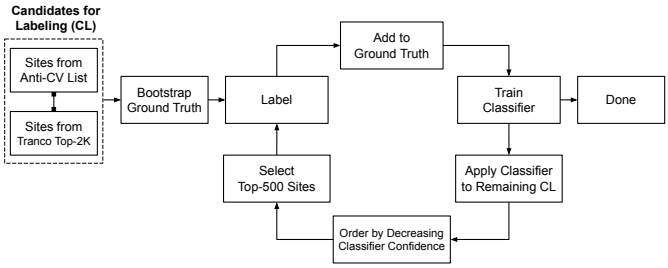


Fig. 9. **Labeling Methodology:** We start with a list of sites from both ACVL and Tranco top-2K, as Candidates for Labeling (CL). We develop an iterative process for prioritizing which (500 in a batch) sites to inspect and label next, then add them to ground truth. We bootstrap a classifier by using outlier detection to find positive labels. In each iteration, we apply the classifier on the remaining sites in CL, sort the sites by decreasing classifier confidence, and inspect and label the 500 sites where the classifier is most confident. Compared to picking randomly 500 sites to label, this heuristic prioritization discovers more positive labels. For example, see Fig. 10 between "Iteration Zero" and "Iteration Zero Random." We add the new labeled samples into our ground truth, retrain our classifier, and repeat the process for two more iterations and declare "Done" when the performance converges, as shown in Fig. 10. We combine all labeled data into our Ground Truth (GT) data set.

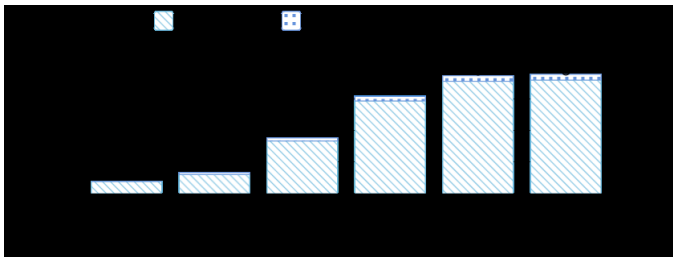


Fig. 10. **Positive Labels and F1 (per Iteration):** For our ground truth, we show how many positive labels (sites with successful circumvention) were discovered within each iteration. When we compare iteration zero and the randomly chosen iteration zero, we find that our methodology discovers twice as many more positive labels. We see that by the end of iteration two, we receive diminishing returns on our classifier performance based on its F1-score. Note that we only find 55 positive labels from the Tranco top-2K overall.

Prioritizing Which Sites to Label. Labeling is time-consuming and is a well-known bottleneck in all communities that maintain filter lists, including EL and ACVL. We develop a heuristic for prioritizing which sites from CL to inspect and label first to rapidly discover positive labels and minimize the overall effort. We employ an iterative process shown in Fig 9.

Bootstrapping. We start from CL and perform outlier detection using Isolation Forest [65]; our intuition is that sites that utilize circumvention are drastically different from those that do not. However, not all outliers have circumvention, as there can be other reasons why a site behaves differently, such as displaying more page content when ads are not displayed. Therefore, we still need to inspect and label this initial (108) outliers, and we find 56 positive labels. Next, we order the remaining sites extracted from ACVL by Tranco ranking, and pick the top-400 sites. Our intuition comes from Fig. 5, where there are around 400 sites in the Tranco top-100k sites. We balance our ground truth with the most popular sites in the ACVL so that our classifier can generalize well in the wild. We merge the labeled outliers with the top-400 sites in the ACVL to obtain our first batch of ground truth with ∼500 sites. We train our classifiers on this GT.

Label	Precision	Recall	Accuracy	F1-score
CV	0.94	0.84	0.93	0.89
No CV	0.92	0.97	0.93	0.94

TABLE IV. **CV-INSPECTOR Cross-validation Results.** USING A RANDOM FOREST CLASSIFIER, 93 FEATURES, AND 5-FOLD VALIDATION. THE LABEL "CV" MEANS SUCCESSFUL CIRCUMVENTION AND "NO CV" MEANS THAT SITES HAVE NO CV ACTIVITY OR FAILED AT CV.

Iteratively enhancing the ground truth. We apply the classifier on the remaining sites of CL, sort the sites by decreasing classifier confidence, and inspect and label the 500 sites where the classifier is most confident. We add the new labeled samples into our ground truth, we retrain our classifier, and repeat the process. In each iteration, we choose and label 500 sites and add them to the ground truth, until the performance converges. Fig. 10 shows diminishing returns after iteration 1, thus we stop at 2 iterations. The main advantage of prioritizing which sites to label is that it discovers more positive labels in each iteration, compared to *e.g.* choosing 500 random sites to label. This saves human effort, which is the main bottleneck. Fig. 10 compares Iteration Zero (with our choice of 500 sites in decreasing confidence) vs. Iteration Zero (Random choice of 500 sites) and shows that we discover more than twice the positive labels, and we achieve a higher F1.

Ground Truth Data Set (GT). We combine all labeled data (from all iterations, including the randomly selected Iteration Zero) into one data set, which we refer to as GT. It contains 755 positive labels and 1566 negative labels.

E. The CV-INSPECTOR *Classifier*

Training the Classifier. We train a classifier that can detect successful circumvention, using all 93 features extracted in Sec. IV-C, and the ground truth obtained in Sec. IV-D. We considered different classifiers and observed that Random Forest performs best. We split the GT data into 70/30 for training and testings, respectively, and we perform 5-fold cross-validation. We consider our contribution to lie not in the ML technique itself but in the domain-knowledge that guided the design of differential analysis, feature selection, and ground truth labeling.

Cross-Validation Results. We display the results in Table IV. Detecting positive labels (*i.e.* sites succeeding in circumventing adblockers) is of interest for filter list authors such as ABP. Here, we achieve an F1-score of 0.89 and precision of 0.94. Detecting negatives labels is also important because authors want to be confident when disregarding sites without circumvention accurately: we see an F1-score of 0.94 and a precision of 0.92; this becomes invaluable in the monitoring approach in Sec. V-B as it reduces human effort.

Important Features. Not all 93 features from Sec. IV-C are equally important. In Table III, we denote some of the features that end up being in the top-10 most important ones. Fig. 11 also shows the empirical CDFs (ECDF) of four top-features and illustrates that they can discriminate between sites that employ successful circumvention or not. For example, consider the circumvention technique that randomizes the JS first-party path. We see that the path has much more randomness than sites that did not circumvent the adblocker; see example in

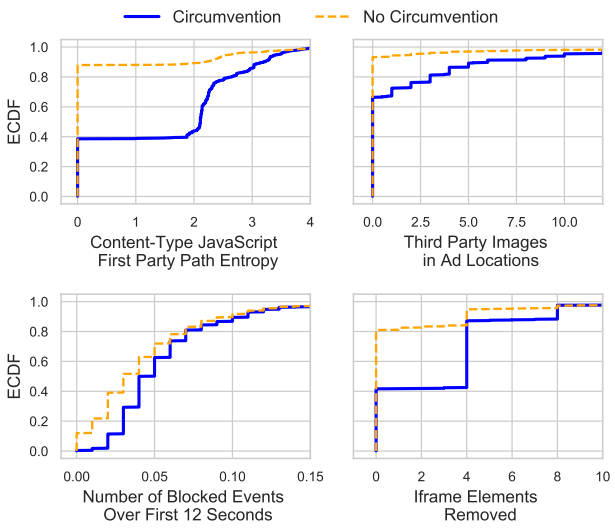


Fig. 11. **Top-Features ECDF.** We show empirical CDFs of some of the top features for our classifier. JS path entropy is the most discriminatory feature.

Listing 2. Specifically, 40% of sites with circumvention have path entropy of two or less, while it is more than 80% of sites with no circumvention. This captures the fact that publishers can use first-party resources that contains circumvention code to initialize the circumvention process. Thus, randomizing the path can make it difficult for the adblocker to block it. The corresponding ECDF is the most discriminatory, compared to ECDFs of other features, uncovering the fact that randomizing the path is a more effective technique against adblockers. Fortunately, our usage of entropy as a feature captures this difference and can detect the presence of circumvention.

Iframe elements removed and third-party images in ad locations are also direct mechanisms of circumvention. The former depicts when sites generally clean up iframes that are being hidden or blocked by the adblocker. The latter details the subsequent actions of re-injecting ad images into previously known ad locations during circumvention. We can infer that if a site completes more ad re-injection actions, then it has a higher chance of circumventing the adblocker. The ECDF of number of blocked events indicates circumvention, where the adblocker generally blocks more for sites that successfully circumvent the adblocker. This highlights that adblockers do not need to block aggressively for sites where they can easily target the root cause of ads. However, when obfuscation techniques are employed, the adblocker must try harder and has a higher chance of not blocking all ads.

Analysis of Mistakes. Next, we discuss the mistakes made by CV-INSPECTOR and we explain the root causes of false negatives (FN) and false positives (FP).

1) False Negatives (FN): FN occur when the site circumvented the adblocker but CV-INSPECTOR predicted that it did not. We find that CV-INSPECTOR does not perform well for sites that employ excessive DOM obfuscation. For example, argumentiru.com displays Yandex [77] ads using nested custom HTML tags named <yatag>, while separating the ad image and the ad link in different parts of the ad DOM structure. This makes it hard to identify whether it is an ad or not and to evaluate the ad link for entropy. In addition, strip2.xxx uses MobiAds [48] to display ads with a small square image and the rest is text outside of the image. This differs from regular ads where it is entirely an image with text encapsulated in the image. As a result, CV-INSPECTOR cannot help notify filter list authors when they should update filter rules for these particular cases. However, we argue that CV-INSPECTOR can be extended to cover corner cases to capture CV activity, if the sites are of interest to the adblocker.

Another reason for FN is the logic of triggering circumvented ads for a user. We find that even when a site is capable of circumventing the adblocker, it may choose not to. Though more future work is necessary to infer the business logic of circumvention, we find that for a few cases where the site only triggers circumvention once out of the four times we load the page, CV-INSPECTOR would predict there is no circumvention. However, the classifier confidence is generally higher (~0.40), which is close to a positive label when compared to when a site displays no ads at all within the four page loads.

Lastly, some FNs are due to the limitations of screenshots not conveying whether an ad is first-party or not. Thus, when investigating these sites, we manually go to the sites and found that they were first-party ads and should be labeled as a negative. Here, we see that the classifier was able to determine the correct label when it comes to first-party ads.

2) False Positives (FP): CV-INSPECTOR can mistake sites that heavily rely on affiliation links or third-party links as their own web content. For example, home-made-videos.com comprises completely of links to third-parties with image dimensions that can be considered as ad dimensions. Furthermore, some mistakes by CV-INSPECTOR can be attributed to a site's code mistakes. For instance, when investigating empflix.com, we find that CV-INSPECTOR accurately identifies web requests that correspond to circumvented ad content. However, during re-insertion, the JS errors out because it expects the existence of an element with ID "mewTives" but the container is actually not there. We note that this error does not happen on the site's sub pages where the container does exists, and CV-INSPECTOR correctly predicts that circumvention happens.

We find some false positives were actual true positives but were mislabeled due to the height cap of screenshots. Recall that we limit the height of the screenshots to be 3000px to be compatible with sites that would infinitely scroll. We discover that many adult content sites using ExoClick [32] would re-inject ads back near the bottom of the page. We see this as a strength of CV-INSPECTOR that establishes that it can detect circumvention beyond just the top part of the site (*i.e.*, above the fold section [54]).

F. Feature Robustness

CV-INSPECTOR extracts a diverse range of features that capture different fundamental characteristics of circumvention. In this section, we discuss approaches that CV providers could utilize to attempt to evade each type of features, along with the approaches' effectiveness and trade-offs involved. We argue that it is challenging for CV providers to evade the features used by the CV-INSPECTOR, while still achieving

their objectives, which are: (1) to evade adblocking filter rules, to display ads, and to obtain publisher ad revenue; (2) to not degrade the user experience on the publisher's site; and (3) to minimize the cost and overhead incurred by the provider when integrating the CV service.

1. Web Request Features. Randomizing URL components, such as subdomains and paths, is a typical obfuscation technique that CV providers use to evade filter rules. However, our entropy features capture not the exact randomized string (which would be easy to evade) but the fact that randomization is used at all (which is robust). An example was shown in Listing 2. To bypass these features, a CV provider would have to stop obfuscating URL components all together, *i.e.* abandon this circumvention technique.

2. DOM Mutation Features. A CV provider could try to manipulate DOM mutation features. For instance, instead of removing DOM nodes, the provider can hide the nodes. However, circumvention would still be detected by our features relating to "DOM attribute changes," such as display and class. CV providers could also try to add noise by causing dummy DOM mutations. However, unless the provider can affect the "No Adblocker" case as well, it will make circumvention activity even easier to detect via differential analysis. Furthermore, adding too many dummy mutations can make the site slow since the browser must refresh how the page is displayed, which affects the user experience.

3. Temporal Features. The CV provider can try to change the number of blocked elements by making the advertising DOM structure simpler, as shown in Listing 3, or more complex by using unnecessary DOM elements. This effectively reduces the number of blocked elements. However, page source features can still detect circumvention by analyzing ad positions rather than the DOM structure. Another possible exploit is to delay the triggering of circumvention (*e.g.* after the 12 second period) so that CV-INSPECTOR does not detect the number of blocked events. However, this goes against the main objective of ads, which is to quickly display ads to the user before the user leaves the page. This delaying approach would negatively affect the revenue that the publisher wants to recover by employing circumvention in the first place.

4. Page Source Features. To evade the features related to the number of iframes and images in ad positions, a CV provider can change the location of ads when circumvention is employed. For example, if ads were original shown on the right side bar for the "No Adblocker" case, then the ads can be moved to left side bar. However, this increases the overhead for the publisher to integrate with CV providers, as the new ad locations must be seamlessly incorporated into individualized templates of different sites. In the above example, the left side bar must make sense within the publisher's template to be a feasible ad location. Also recall from Fig. 1(b) that the publisher must still fetch for new ad content. Thus, CV-INSPECTOR can still capture this circumvention characteristic through our web requests features.

Takeaways. Overall, CV-INSPECTOR raises the bar in the arms race with CV providers, by extracting diverse features that collectively capture the fundamental behavior of CV providers through differential analysis. In order to evade differential analysis, CV providers would have to make the site's behavior "With Adblocker" similar to that of "No Adblocker." However, this either limits ad re-injection to simple static ads (often not profitable for publishers) or requires that CV services are triggered for all users (using adblockers or otherwise) resulting in higher cost for the publisher.

Detection on Tranco-20K Data Set				
Sampling	**Label**	**Predicted**	**Correct**	**Precision**
No	CV	91	79 / 91	87%
Yes	No CV	29,248	345 / 380	91%

TABLE V. WE APPLY CV-INSPECTOR TO THE TRANCO-20K. FOR "NO CV" INSTANCES, WE SAMPLE FROM THAT PREDICTED SET TO HAVE A CONFIDENCE LEVEL OF 95% WITH 5% MARGIN OF ERROR.

G. Summary

In this section, we presented the design and implementation of CV-INSPECTOR. Specifically, it collects data from web requests, DOM mutations, temporal information (including blocked events caused by ABP), page source, and screenshots. Then, it employs differential analysis designed uniquely to capture circumvention activity, and we extract intuitive features specifically designed for capturing circumvention. We also provide an iterative methodology for obtaining ground truth, that can speed up the process while discovering more positive labels. We trained and evaluated a Random Forest classifier using this GT data set, and demonstrated that it achieves an accuracy of 93% in detecting sites that employ CV providers. We further find that web request features relating to path entropy is the most effective feature. By capturing the essential characteristics of circumvention, we conclude that it would be difficult for CV providers to evade both CV-INSPECTOR and filter rules without incurring costs, *i.e.*, not being able to show profitable ads to the users and overhead of activating circumvention for all users. Next, we apply and evaluate CV-INSPECTOR in real world settings.

V. CV-INSPECTOR: IN THE WILD DEPLOYMENT

We employ CV-INSPECTOR in two real world scenarios. First, in Sec. V-A, we employ CV-INSPECTOR on the popular Tranco-20K sites to *discover* sites that circumvent adblockers, and are possibly unknown. Second, in Sec. V-B, we use CV-INSPECTOR to *monitor* the effectiveness of ACVL on sites that are well-known to circumvent adblockers, and which are continuously monitored by filter list curators. For the evaluation of monitoring, we use two data sets: our own GTP data set and Adblock Plus Monitoring data set provided by ABP. More details are provided in the respective sections and the data sets are detailed in Sec. IV-A4 and Table II.

A. Discovering Circumvention in the Wild

1) In the Wild Performance: We first conduct a large-scale analysis of deploying CV-INSPECTOR in the wild. Our goal is to facilitate the crowdsourcing effort by the adblocking community to discover sites that successfully circumvent adblockers. To that end, we apply CV-INSPECTOR on the popular Tranco-20K sites, which contains 29.3K pages with sub-pages. Recall that the Tranco top-2K sites were used as candidates for

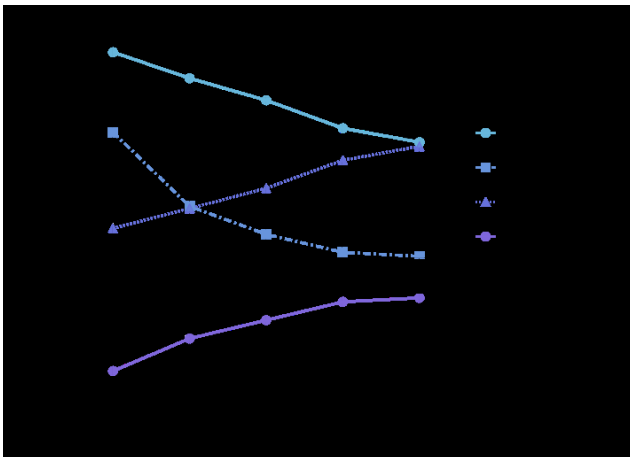
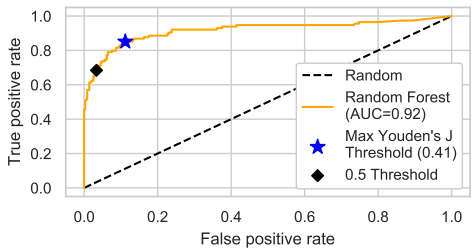
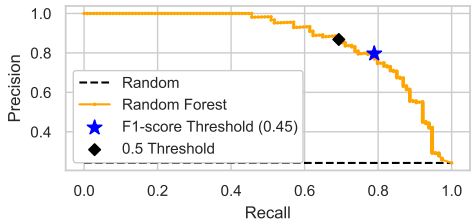


Fig. 12. **Discovery vs. Precision.** The trade-off between discovering more circumvention sites (positive instances) within our Tranco-20K (in the wild) data set *vs.* being correct in the prediction.



(a) ROC Curve



(b) Precision vs. Recall

Fig. 13. **Trade-offs on the Tranco-20K Data Set. (a)** Within our ROC curve, we can maximize Youden's J index if we value high true positive rate (TPR) with low false positive rate (FPR) for the purpose of discovering sites with successful circumvention. The threshold following this criteria is 0.41, which corresponds to a TPR of 0.85 and FPR of 0.11. **(b)** Within our precision-recall curve, we can find the threshold that corresponds to the optimal F1-score for positive labels. The threshold following this criteria is 0.45, which achieves an F1-score of 0.79.

labeling (CL), which eventually affected the training set (GT) for our CV-INSPECTOR's classifier. Therefore, we exclude it from the in the wild evaluation because we want to keep the Tranco sites used for training (top-2K) and testing (top 2k-20K) disjoint. We follow our earlier data collection approach, described in Fig. 6, to crawl these URLs. As shown in Table V, CV-INSPECTOR detects 91 sites as "CV" and the remaining 29,248 sites as "No CV." We validate the 91 "CV" sites and a random sample (380) of "No CV" sites. CV-INSPECTOR achieves 87% precision when identifying sites with successful CV and 91% for the opposite case. Our evaluation in Table V shows that CV-INSPECTOR generalizes well in the wild, with similar precision when compared to Table IV.

The Random Forests classifier picks the likeliest class, which in binary classification is by default the class with probability above 0.5. This is the case in the results presented in Tables V, VII, and VIII in this section. CV-INSPECTOR can be applied to different use cases (*e.g.* discovery or monitoring of sites employing circumvention) that value different metrics (*e.g.* recall vs. precision, respectively). Since there is no universally applicable operating point, instead of tuning parameters to overfit a particular use case and data set, we discuss the trade-offs involved and leave it up to the users of CV-INSPECTOR to decide upon the operating point that matches their goals.

Trade-offs. Fig. 12 reports how CV-INSPECTOR navigates the trade-off between discovering more sites that successfully circumvent adblockers and precision, when applied to the Tranco-20K data set. For instance, a confidence level threshold of 0.6 achieves a precision of 98% with only one FP. This would be an attractive option to minimize human supervision for monitoring sites of interest. However, if discovering sites that use circumvention is more important, then lowering the threshold below 0.5 would find more sites at the expense of increasing human efforts to deal with FPs. The operating point can be tweaked to optimize various objectives of interest.

As a concrete example, Fig. 13(a) depicts how CV-INSPECTOR can navigate the trade-off between true positive rate (TPR) and false positive rate (FPR). The suitable classifier threshold depends on the use case. For example, if one wants to optimize for TPR (*i.e.*, recall) while keeping the FPR low, one metric to maximize is Youden's J index, which leads to a threshold of 0.41 with corresponding TPR of 0.85 and FPR of 0.11. This is the right objective when we are interested in discovering more sites that employ circumvention at the risk of some additional false positives.

Another even more relevant trade-off in our case is precision vs. recall, depicted on Fig. 13(b). We find that a threshold of 0.45 maximizes the F1-score for positive labels, achieving a F1-score of 0.79. It is not surprising that this value is close but below 0.5, because our Tranco-20K data set is imbalanced as shown in Table V. As discussed in Sec. IV-D, positive labels are rare when compared to negative labels, which makes the classifier less sensitive to the minority class. To compensate for this, one would decrease the threshold to improve recall for positive labels at the expense of precision.

2) Circumvention Providers: We now analyze the breakdown of different circumvention providers. Note that CV-INSPECTOR is not designed to distinguish between different circumvention providers. Therefore, we rely on other heuristics to detect specific CV providers.

Unique Keywords for CV Providers. We curate keywords that are indicative of specific CV providers through a careful manual inspection of known circumvention sites in our GT data set. Intuitively, to discover keywords, we first search using the name of the providers within our collected data of web requests (*e.g.*, URLs and HTTP request/response headers) and page source files (*e.g.*, HTML files consisting of HTML, inline CSS, and inline JS). Notably, we discover that some CV providers, like ExoClick, AdDefend, and Adthrive, do not attempt to hide

CV Providers	Count	WR	DOM	CV-INSPECTOR precision
AdThrive [12]	154	●	◐	98%
Publica [59]	77	●	◐	95%
ExoClick [32]	76	◐	◐	100%
Yandex [77]	434	◐	●	100%
AdDefend [7]	38	◐	●	N/A
MobiAds [48]	17	◐	●	N/A

TABLE VI. **CIRCUMVENTION PROVIDERS & APPROACHES**. WE SHOW THE PRESENCE OF CIRCUMVENTION PROVIDERS WITHIN THE TRANCO-20K. WE USE ● TO MEAN FULL OBFUSCATION, WHICH MEANS RANDOMIZED URL COMPONENTS (WR) OR DEEPLY NESTED NONSTANDARD DOM STRUCTURES FOR AD (DOM). ◐ DENOTES PARTIAL OBFUSCATION, WHICH MEANS AD RESOURCES MAY BE HIDDEN WITH FIRST-PARTY DOMAIN (WR) AND AD REINSERTION USES SIMPLER DOM STRUCTURES (DOM). WR = WEBREQUESTS, DOM = DOM CHANGES.

Ground Truth Positives (GTP) Data Set				
Sampling	**Label**	**Predicted**	**Correct**	**Precision**
No	CV	244	223 / 244	91%
Yes	No CV	465	187 / 211	89%

TABLE VII. WE SHOW THE RESULTS OF APPLYING OUR CLASSIFIER ON ∼700 SITES FROM OUR GROUND TRUTH THAT ALSO ORIGINATED FROM ACVL (TABLE II). HOWEVER, THIS TIME WE COLLECT THE DATA BY TURNING ON ACVL AS WELL WITHIN OUR CUSTOM ABP EXTENSION. FOR "NO CV" INSTANCES, WE SAMPLE FROM THAT PREDICTED SET TO HAVE A CONFIDENCE LEVEL OF 95% WITH 5% MARGIN OF ERROR.

Adblock Plus Monitoring Data Set				
Sampling	**Label**	**Predicted**	**Correct**	**Precision**
No	CV	5	4 / 5	80%
Yes	No CV	355	184 / 185	99%

TABLE VIII. FROM A REAL WORLD DATA SET USED BY ABP TO MONITOR CIRCUMVENTION, WE APPLY OUR CLASSIFIER AND SHOW THE RESULTS. FOR "NO CV" INSTANCES, WE SAMPLE FROM THAT PREDICTED SET TO HAVE A CONFIDENCE LEVEL OF 95% WITH 5% MARGIN OF ERROR.

their presence, as the name of the provider was sufficient to be used as keywords. For example, we found that ExoClick can be detected by the keywords "exoclick" and "exoloader" in the page source. For AdDefend, AdThrive and MobiAds, we look at keywords "addefend," "adthrive," and "mobiads" in the page source, respectively. Similarly, Publica can be detected by looking for the key "publica_user_id" in the HTTP response Set-Cookie header. When the name of the provider was not enough, we inspected the DOM structure of the ad using the page source to see if there was any unique identifier that we could use. In this case, we find that Yandex can be detected by looking for the custom DOM tag "<yatag>." In total, we utilized seven keywords to identify the presence of six different CV providers listed in Table VI.

We note that the presence of these keywords does not necessarily always indicate that circumvention was successful. It could also mean that circumvention attempt failed or that circumvention was not even attempted (e.g., dormant code). Therefore, we cannot simply use these heuristics in place of CV-INSPECTOR to detect sites that circumvent adblockers. Table VI summarizes the application of the aforementioned heuristics on Tranco-20K sites. We identify many instances of different CV providers, including ad networks such as Yandex and dedicated CV providers such as AdThrive and AdDefend.

Taxonomy of Circumvention Approaches. Next, we characterize the obfuscation approaches used by different CV providers by defining whether the obfuscation is full (●) or partial (◐). For web request obfuscation, full obfuscation refers to the use of randomized URLs including subdomains and paths as shown in Listing 2 while partial obfuscation refers to the use of first-party subdomains. For DOM obfuscation, full obfuscation refers to the use of non-standard DOM structures such as deeply nested elements or randomized tag attributes while partial obfuscation refers to only randomized tag attributes with simple DOM structures, such as Listing 3.

Using this taxonomy, we compare the full *vs.* partial obfuscation techniques of different CV providers. First, ExoClick and AdDefend simply leverage inlined JS, which is difficult to block without hurting other page functionality [22], to implement their circumvention logic. AdThrive redirects through several domains (*e.g.*, cloudfront.net → edvfwlacluo.com → lmyiwaakn.com) before fetching the JS that implements their circumvention logic. Second, ExoClick and AdDefend do not obfuscate URLs but rather serve their ad resources under first-party domains that are difficult to distinguish from legitimate content. AdThrive fetches ads in iframes using rotating third-party domains, subdomains, and randomized IDs. Third, ExoClick and AdDefend differ in their DOM obfuscation techniques. ExoClick uses a simpler ad structure (a hyperlink with two div children) while obfuscating the ad image by serving it with CSS background-image instead of a regular image tag. On the other hand, AdDefend employs a nested DOM structure with obfuscated IDs, while Yandex uses nested non-standard tags with obfuscated class names.

Finally, we analyze CV-INSPECTOR's performance in detecting different CV providers. We match each detected CV provider instance in Table VI to our CV-INSPECTOR deployment results on Tranco-20K sites from Table V. We see that CV-INSPECTOR achieves good precision in detecting different popular CV providers. For AdDefend and MobiAds, we use "N/A" to denote that we lack sufficient data.

B. Monitoring Circumvention for Sites of Interest

As discussed in Sec. III-B, ACVL is updated very frequently to combat the back and forth between adblockers and circumvention providers. In addition, filter list authors generally focus their attention on "sites of interest," as discussed in Sec. IV-A4. Curators must continuously monitor them to see if the filter list (ACVL) continues to be effective, or if circumvention has evolved, and the filter rules need updating. Consequently, much human labor goes to this continuous monitoring of sites in the ACVL. To that end, we show how CV-INSPECTOR can automatically monitor whether ACVL is effective in countering circumvention on a site. We use the same approach as laid out in Fig. 6 but with one change. We use the ACVL, in addition to EL, when crawling a site with an adblocker. We use two data sets from Table II for evaluation: (1) the GTP data set, which contains all sites that circumvent the adblocker in our GT; and (2) Adblock Plus Monitoring data set, which contains 360 sites that ABP continuously monitor for circumvention activity to update filter rules.

1) Monitoring sites in GTP: We use CV-INSPECTOR to classify sites within our GTP data set, which comprises of 700 sites from the GT data set that originated from ACVL and were successful at circumventing the adblocker. If CV-INSPECTOR again detects a site as "CV," it shows that the site is able to successfully circumvent even the ACVL. We manually validate CV-INSPECTOR's classifications. Table VII summarizes the results. We note that CV-INSPECTOR again detects 244 sites as "CV" with 91% precision and 465 sites as "No CV" with 89% precision. The results show that more than one-third of sites with relevant filter rules in the ACVL are still able to successfully circumvent adblockers. This demonstrates that the sites addressed by the ACVL need to be continuously monitored. We suggest that CV-INSPECTOR should be periodically used (e.g., every hour) to monitor the sites on the ACVL. The sites that are detected by CV-INSPECTOR would need to be reviewed by ACVL curators to update the filter rules and the rest can be safely ignored.

2) Monitoring sites from ABP: To further demonstrate CV-INSPECTOR's usefulness, we obtain a list of 360 sites from ABP that are manually monitored by the ABP team due to the sites' fast-paced adaptation to changes in the ACVL. Table VIII summarizes the results of applying CV-INSPECTOR (with ACVL) on these sites. Out of these sites, we note that 5 sites are detected as "CV" and the remaining 355 as "No CV," again with high precision. This finding shows that even the sites that are closely monitored to be addressed by the ACVL team can successfully circumvent the adblocker. Notably, if we consider only the 190 sites that we labeled as human labeling effort, then CV-INSPECTOR was able to save up to 98% of the work for ACVL curators by predicting 188 sites correctly. Thus, CV-INSPECTOR can help with continuously monitoring these sites at a high frequency.

VI. DISCUSSION AND FUTURE DIRECTIONS

Summary. In this paper, we studied an emerging threat in the advertising ecosystem: circumvention (CV) services that help publishers bypass adblockers and re-injects ads. CV services are sophisticated, opaque for the user, and exploit fundamental weaknesses of adblockers' design and the open-source nature of anti-CV community efforts (exemplified by the anti-CV list). Although there has been increasing anecdotal evidence about adoption of circumvention in the wild, to the best of our knowledge, ours is the first large-scale study of the state of circumvention arms race. We develop CV-INSPECTOR: a methodology for automatically crawling sites of interest and a classifier that can accurately detect whether a site successfully circumvent the adblocker or not. We envision that CV-INSPECTOR will serve as an automation tool for filter list curators to help them focus their inspection efforts in discovering new sites that employ circumvention in the wild and in monitoring sites of interest continuously in the arms race between circumvention and anti-CV filter rules.

Open Source Tools. We plan to make CV-INSPECTOR available to the community at [72]. This will include the data sets (including our labeled dataset of top-20K crawled sites), crawling instrumentation (shareable as Amazon Machine Images [15]), and the trained classifier.

Limitations. There are limitations in our design and implementation. First, CV-INSPECTOR uses differential analysis that relies on differences between the "No Adblocker" and "With Adblocker" cases. If sites exhibit no actual differences in the two cases, then CV-INSPECTOR will not be able to detect circumvention. For example, searchenginereports.net already includes circumvented ads in the DOM structure of the "No Adblocker" case but only hidden. When it detects an adblocker affecting its ads, it will simply show the backup ads that were already there. Second, CV-INSPECTOR only considers circumvention that appears without user interaction. For instance, shahid4u.cam displays no visual ads to the user, but when the user clicks on a link, it will redirect the user to an ad before showing the real content. More details on implementation choices and limitations are provided in Sec. IV-A3.

Future Directions. We plan to further *automate filter rule generation* and help anti-CV authors, by building on two opportunities already provided by CV-INSPECTOR. First, our differential analysis already uncovers web requests that are related to circumvention. Consider the spring.org.uk example: CV-INSPECTOR already pinpoints all randomized paths and subdomains of podfdch.com. Using that information, a filter list author can simply create a filter rule such as "*.podfdch.comˆ" or any variations of its subdomains and paths if there are common prefixes and suffixes like "||podfdch.com/erej*". Second, our feature extraction already dynamically generates CSS selectors of ad locations where re-injection can happen. Filter list authors can translate them into DOM element hiding rules, as described in Table I. They can infer the effectiveness of the selectors — the more elements that match, the more ads the selectors will affect.

It also remains to be seen how robust CV-INSPECTOR is in the presence of ever-changing circumvention obfuscation techniques. Our intuition is that the features used by CV-INSPECTOR (*e.g.*, randomness in an obfuscated path) are inherently more long-lived and harder to evade than the exact rules used by filter lists (*e.g.*, the actual randomized string in the path). It would be interesting to characterize the *time scales* of this arms race.

Feature engineering can also be improved. We can consider new features (*e.g.* extracted from JS) and improve existing features (*e.g.* the way we capture DOM mutation, by taking into account the DOM graph structure in the differential analysis). With respect to JS in particular, the current version of CV-INSPECTOR does not take into account JS features on purpose, because CV providers heavily obfuscate JS, which makes differential analysis challenging. As shown in Fig. 1(b), this involves retrieving new ad content (web requests) and displaying the ad to the user at the end (DOM structure). The technique that JS utilizes to re-inject ads back upon the page does not matter: as long as CV-INSPECTOR can recognize the final DOM structure, it can still detect circumvention.

Overall, we consider CV-INSPECTOR to be the first significant step towards automating aspects of the defense (ad blockers, filter list authors' effort) against circumvention by showing that it can reduce human labeling efforts by 98%. The longer term goal is to fully automate the defense against circumvention through detection and filter list generation.

ACKNOWLEDGEMENTS

This work is supported in part by NSF Awards 1815666 and 1715152, 1815131, 1954224. We would like to thank our NDSS shepherd, Sooel Son, and the anonymous NDSS reviewers, for their constructive and detailed feedback. We would also like to thank UCI undergraduate students Qingchuan Yang and Yiyu Qian, who helped inspect and label the GT data set. Last but not least, the authors would like to thank eyeo [34], for providing the Adblock Plus Monitoring data set; special thanks to Oleksandr Paraska, Uwe Bernitt, and Shwetank Dixit, for sharing their insights on circumvention.

REFERENCES

[1] "The Acceptable Ads Standard," https://acceptableads.com/standard, Acceptable Ads.

[2] "Adblock warning removal list," https://easylist-downloads.adblockplus.org/antiadblockfilters.txt, Adblock Plus, (Accessed on 07/10/2020).

[3] "Taboola whitelisting too annoying, turned off acceptable ads," https://adblockplus.org/forum/viewtopic.php?f=17&t=50287, Adblock Plus, January 2017, (Accessed on 05/04/2020).

[4] "ABP anti-circumvention filter list ," https://github.com/abp-filters/abp-filters-anti-cv, Adblock Plus, 2019, (Accessed on 05/09/2019).

[5] "Adblock plus — the world's # 1 free ad blocker," https://adblockplus.org/, Adblock Plus, 2020, (Accessed on 07/22/2020).

[6] "Contributors to abp-filters/abp-filters-anti-cv," https://github.com/abp-filters/abp-filters-anti-cv/graphs/contributors, Adblock Plus, May 2020, (Accessed on 05/21/2020).

[7] "Anti-adblock platform - addefend.com," https://www.addefend.com/en/platform/##why-anti-adblock-inventory, AdDefend, 2020, (Accessed on 03/18/2020).

[8] "AdGuard Scriptlets and Resources ," https://github.com/AdguardTeam/Scriptlets, AdGuard, 2019, (Accessed on 05/09/2019).

[9] "Adguard — world's most advanced adblocker!" https://adguard.com/en/welcome.html, AdGuard, 2020, (Accessed on 07/22/2020).

[10] "Adguardextra: Adguard extra is designed to solve complicated cases when regular ad blocking rules aren't enough." https://github.com/AdguardTeam/AdGuardExtra, AdGuard, 2020, (Accessed on 03/18/2020).

[11] "Admiral launches one-click subscriptions and donations for publishers to help grow alternative revenue post-gdpr," https://blog.getadmiral.com/admiral-launches-subscriptions-donations-transact-publishers, Admiral, 2020, (Accessed on 03/18/2020).

[12] "Ad management and optimization for the world's best content creators," https://www.adthrive.com/, AdThrive, 2020, (Accessed on 07/20/2020).

[13] M. Alzirah, S. Zhu, Z. Xing, and G. Wang, "Errors, misunderstandings, and attacks: Analyzing the crowdsourcing process of ad-blocking systems," in *Proceedings of the Internet Measurement Conference 2019*, ser. IMC '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3355369.3355588

[14] "Amazon ec2 instance types - amazon web services," https://aws.amazon.com/ec2/instance-types/, Amazon, 2020, (Accessed on 05/09/2020).

[15] "Amazon machine images (ami) - amazon elastic compute cloud," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html, Amazon, 2020, (Accessed on 05/09/2020).

[16] M. A. Bashir, S. Arshad, E. Kirda, W. Robertson, and C. Wilson, "How tracking companies circumvented ad blockers using websockets," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: ACM, 2018, pp. 471–477. [Online]. Available: http://doi.acm.org/10.1145/3278532.3278573

[17] M. A. Bashir, S. Arshad, and C. Wilson, ""recommended for you": A first look at content recommendation networks," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 17–24. [Online]. Available: https://doi.org/10.1145/2987443.2987469

[18] "The Better Ads Standards," https://www.betterads.org/standards, Better Ads.

[19] "Blockadblock — stop losing ad revenue," https://blockadblock.com/, BlockAdBlock, (Accessed on 05/04/2020).

[20] "2020 adblock report – blockthrough," https://blockthrough.com/2020/02/06/2020-adblock-report-3/, Blockthrough, February 2020, (Accessed on 03/18/2020).

[21] "brave/ad-block: Ad block engine used in the brave browser for abp filter syntax based lists like easylist." https://github.com/brave/ad-block, Brave, May 2020, (Accessed on 06/21/2020).

[22] Q. Chen, P. Snyder, B. Livshits, and A. Kapravelos, "Improving web content blocking with event-loop-turn granularity javascript signatures," 2020.

[23] Y. Chen, "Tough sell: Why publisher 'turn-off-your-ad-blocker' messages are so polite - digiday," https://digiday.com/media/tough-sell-publisher-turn-off-ad-blocker-messages-polite/, April 2016, (Accessed on 05/04/2020).

[24] "Chromedriver - webdriver for chrome," https://sites.google.com/a/chromium.org/chromedriver/, Chromium, (Accessed on 05/09/2020).

[25] "Chromium blog: Under the hood: How chrome's ad filtering works," https://blog.chromium.org/2018/02/how-chromes-ad-filtering-works.html, Chromium, Febrary 2018, (Accessed on 05/04/2020).

[26] T. Claburn, "Revealed: The naughty tricks used by web ads to bypass blockers," https://www.theregister.co.uk/2017/08/11/ad_blocker_bypass_code/, 2017, (Accessed on 05/09/2019).

[27] R. Cointepas, "Cname cloaking, the dangerous disguise of third-party trackers," https://medium.com/nextdns/cname-cloaking-the-dangerous-disguise-of-third-party-trackers-195205dc522a, November 2019, (Accessed on 05/04/2020).

[28] H. Dao, J. Mazel, and K. Fukuda, "Characterizing cname cloaking-based tracking on the web," *IEEE/IFIP TMA'20*, pp. 1–9, 2020.

[29] "Disconnectme tracking services," https://raw.githubusercontent.com/disconnectme/disconnect-tracking-protection/master/services.json, DisconnectMe, May 2020, (Accessed on 06/21/2020).

[30] S. Dixit, "Block, unblock, block! How ad blockers are being circumvented ," https://www.youtube.com/watch?v=Vk9bPDaZELQ, 2019, (Accessed on 05/09/2019).

[31] "Easyprivacy," https://easylist.to/easylist/easyprivacy.txt, EasyList, June 2020, (Accessed on 06/21/2020).

[32] "Exoclick the innovative ad company," https://www.exoclick.com/, ExoClick, 2020, (Accessed on 07/26/2020).

[33] "Snippet filters tutorial — adblock plus help center," https://help.eyeo.com/adblockplus/snippet-filters-tutorial, eyeo, (Accessed on 06/10/2020).

[34] "eyeo gmbh – putting you in charge of a fair, profitable web." https://eyeo.com/, eyeo, 2020, (Accessed on 07/30/2020).

[35] "Filterlists — subscriptions for ublock origin, adblock plus, adguard, ..." https://filterlists.com/, FilterLists, 2020, (Accessed on 07/30/2020).

[36] M. Garcia, "Circumvention of ad blockers? not on our watch. – eyeo gmbh," https://eyeo.com/circumvention-of-ad-blockers-not-on-our-watch/, September 2018, (Accessed on 05/04/2020).

[37] "chrome.webrequest - google chrome," https://developer.chrome.com/extensions/webRequest, Google Chrome, 2020, (Accessed on 05/06/2020).

[38] greiner, "Adblock plus • view topic - why anti-circumvention filter list not operated by easylist?" https://adblockplus.org/forum/viewtopic.php?f=4&t=59473, Adblock Plus, September 2019, (Accessed on 05/23/2020).

[39] gwarser, "Resources Library ," https://github.com/gorhill/uBlock/wiki/Resources-Library, 2019, (Accessed on 05/09/2019).

[40] hfiguiere, "#6969 (implement abort-on-property-read snippet) – adblock plus issue tracker," https://issues.adblockplus.org/ticket/6969, March 2019, (Accessed on 06/11/2020).

[41] "Deal - iab tech lab," https://iabtechlab.com/standards/ad-blocking/deal/, IAB Tech Lab, (Accessed on 05/04/2020).

[42] U. Iqbal, Z. Shafiq, and Z. Qian, "The ad wars: Retrospective measurement and analysis of anti-adblock filter lists," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New

York, NY, USA: ACM, 2017, pp. 171–183. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131387

[43] M. Jethani, "Adblock plus and (a little) more: Adblock plus 3.3 for chrome, firefox and opera released," https://adblockplus.org/releases/adblock-plus-33-for-chrome-firefox-and-opera-released, August 2018, (Accessed on 05/04/2020).

[44] L. Kudryavtseva, "New ad-tech terms: "ad reinsertion", "ad recovery", "ad replacement"," https://adguard.com/en/blog/ad-reinsertion.html, AdGuard, March 2017, (Accessed on 03/18/2020).

[45] N. Lomas, "Adblock Plus maker has a new taskforce to fight publisher efforts to reinject ads," https://techcrunch.com/2018/09/19/adblock-plus-maker-has-a-new-taskforce-to-fight-publisher-efforts-to-reinject-ads/, 2018, (Accessed on 05/09/2019).

[46] 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 *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 319–333.

[47] mjethani, "Implement basic support for snippet filters ," https://issues.adblockplus.org/ticket/6781, 2018, (Accessed on 05/09/2019).

[48] "mobiad – home — mobiad home," http://mobiadhome.com/, Mobiad, 2020, (Accessed on 07/26/2020).

[49] "Mutationobserver - web apis — mdn," https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver, Mozilla, January 2020, (Accessed on 05/06/2020).

[50] M. H. Mughees, Z. Qian, and Z. Shafiq, "Detecting anti ad-blockers in the wild," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 130–146, 2017.

[51] B. Muthukadan, "Selenium with python — selenium python bindings 2 documentation," https://selenium-python.readthedocs.io/, 2018, (Accessed on 05/09/2020).

[52] R. Nithyanand, S. Khattak, M. Javed, N. Vallina-Rodriguez, M. Falahrastegar, J. E. Powles, E. De Cristofaro, H. Haddadi, and S. J. Murdoch, "Adblocking and counter blocking: A slice of the arms race," in *6th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 16)*, 2016.

[53] A. Oehler, "How the platform works – help center," https://support.instart.com/hc/en-us/articles/220929867, October 2019, (Accessed on 03/18/2020).

[54] "Above the fold," https://www.optimizely.com/optimization-glossary/above-the-fold/, Optimizely, (Accessed on 07/14/2020).

[55] "Oriel," https://oriel.io/index.html##howitworks, Oriel, 2020, (Accessed on 03/18/2020).

[56] Page Fair, "The State of the Blocked Web," https://pagefair.com/downloads/2017/01/PageFair-2017-Adblock-Report.pdf, 2017, (Accessed on 05/09/2019).

[57] pkalinnikov, "Issue 2449913002: Support websocket in webrequest api. - code review," https://codereview.chromium.org/2449913002/, 2017, (Accessed on 05/04/2020).

[58] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," *arXiv preprint arXiv:1806.01156*, 2018.

[59] "Publica," https://dev.getpublica.com/products/, Publica, 2020, (Accessed on 07/27/2020).

[60] E. Pujol, O. Hohlfeld, and A. Feldmann, "Annoyed users: Ads and ad-block usage in the wild," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 93–106. [Online]. Available: https://doi.org/10.1145/2815675.2815705

[61] "Adblock circumvention strategies: Ad reinsertion, ad replacement, ad recovery," http://news.reviveads.com/adblock-circumvention-strategies/, ReviveAds, (Accessed on 03/18/2020).

[62] "Reviveads and ad reinsertion: An overview," http://news.reviveads.com/white-paper-reviveads-ad-reinsertion/, ReviveAds, (Accessed on 03/18/2020).

[63] K. Rogers, "Why doesn't my ad blocker block 'please turn off your ad blocker' popups? - vice," https://www.vice.com/en_us/article/j5zk8y/why-your-ad-blocker-doesnt-block-those-please-turn-off-your-ad-blocker-popups, December 2018, (Accessed on 05/04/2020).

[64] sashachu, "D: #8471 · abp-filters/abp-filters-anti-cv@d36effc," https://github.com/abp-filters/abp-filters-anti-cv/commit/d36effc62ec5207f5a6730127372a6cd3ebd1717, December 2018, (Accessed on 06/11/2020).

[65] "sklearn.ensemble.isolationforest — scikit-learn 0.23.1 documentation," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html, scikit-learn, September 2016, (Accessed on 07/07/2020).

[66] S. Singh, "LEAN - IAB Tech Lab," https://iabtechlab.com/standards/ad-blocking/lean/, 2019, (Accessed on 05/09/2019).

[67] SourcePoint, "Homepage - sourcepoint," https://www.sourcepoint.com/, 2020, (Accessed on 03/18/2020).

[68] The EasyList authors, "EasyList," https://easylist.to/, 2005, [Online; accessed 2019-05-11].

[69] "Resources for uBlock Origin, uMatrix: static filter lists, ready-to-use rulesets, etc. ," https://github.com/uBlockOrigin/uAssets, uBlock Origin, 2019, (Accessed on 05/09/2019).

[70] "Getadmiral-domains," https://raw.githubusercontent.com/LanikSJ/ubo-filters/master/filters/getadmiral-domains.txt, uBlock Origin, March 2020, (Accessed on 06/21/2020).

[71] "gorhill/ublock: ublock origin - an efficient blocker for chromium and firefox. fast and lean." https://github.com/gorhill/uBlock, uBlock Origin, July 2020, (Accessed on 07/22/2020).

[72] "CV-Inspector: Towards Automating Detection of Adblock Circumvention: Project Overview," https://athinagroup.eng.uci.edu/projects/cv-inspector/, UCI Networking Group, January 2021, (Accessed on 01/04/2021).

[73] A. Vastel, P. Snyder, and B. Livshits, "Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking," *arXiv preprint arXiv:1810.09160*, 2018.

[74] R. J. Walls, E. D. Kilmer, N. Lageman, and P. D. McDaniel, "Measuring the impact and perception of acceptable advertisements," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 107–120. [Online]. Available: https://doi.org/10.1145/2815675.2815703

[75] W. Wang, Y. Zheng, X. Xing, Y. Kwon, X. Zhang, and P. Eugster, "Webranz: Web page randomization for better advertisement delivery and web-bot prevention," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 205–216. [Online]. Available: https://doi.org/10.1145/2950290.2950352

[76] wizmak, "D: #9056 · abp-filters/abp-filters-anti-cv@ddd0c3d," https://github.com/abp-filters/abp-filters-anti-cv/commit/ddd0c3d9cd729d589519c57ba9aaa07229bdf10c, November 2018, (Accessed on 06/11/2020).

[77] "Yandex advertising network and ad exchanges - yandex.direct. help," https://yandex.com/support/direct/general/yan.html, Yandex, 2020, (Accessed on 07/26/2020).

[78] S. Zhu, X. Hu, Z. Qian, Z. Shafiq, and H. Yin, "Measuring and disrupting anti-adblockers using differential execution analysis," in *The Network and Distributed System Security Symposium (NDSS)*, 2018.