TrackerSift: Untangling Mixed Tracking and Functional Web Resources

Abdul Haddi Amjad
hadiamjad@vt.edu
Virginia Tech
USA

Danial Saleem
l174115@lhr.nu.edu
FAST-NUCES
Pakistan

Muhammad Ali Gulzar
gulzar@cs.vt.edu
Virginia Tech
USA

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

Fareed Zaffar
fareed.zaffar@lums.edu.pk
LUMS
Pakistan

ABSTRACT

Trackers have recently started to mix tracking and functional resources to circumvent privacy-enhancing content blocking tools. Such mixed web resources put content blockers in a bind: risk breaking legitimate functionality if they act and risk missing privacy-invasive advertising and tracking if they do not. In this paper, we propose TrackerSift to progressively classify and untangle mixed web resources (that combine tracking and legitimate functionality) at multiple granularities of analysis (domain, hostname, script, and method). Using TrackerSift, we conduct a large-scale measurement study of such mixed resources on 100K websites. We find that more than 17% domains, 48% hostnames, 6% scripts, and 9% methods observed in our crawls combine tracking and legitimate functionality. While mixed web resources are prevalent across all granularities, TrackerSift is able to attribute 98% of the script-initiated network requests to either tracking or functional resources at the finest method-level granularity. Our analysis shows that mixed resources at different granularities are typically served from CDNs or as inlined and bundled scripts, and that blocking them indeed results in breakage of legitimate functionality. Our results highlight opportunities for finer-grained content blocking to remove mixed resources without breaking legitimate functionality.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; **Browser security**; • **Software and its engineering** → *Software defect analysis*.

ACM Reference Format:
Abdul Haddi Amjad, Danial Saleem, Muhammad Ali Gulzar, Zubair Shafiq, and Fareed Zaffar. 2021. TrackerSift: Untangling Mixed Tracking and Functional Web Resources. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3487552.3487855

1 INTRODUCTION

Background & Motivation. Privacy-enhancing content blocking tools such as AdBlock Plus [2], uBlock Origin [1], and Brave [4] are widely used to block online advertising and/or tracking [26, 37, 39]. Trackers have engaged in the arms race with content blockers via counter-blocking [40, 42] and circumvention [17, 36]. In the counter-blocking arms race, trackers attempt to detect users of content blocking tools and give them an ultimatum to disable content blocking. In the circumvention arms race, trackers attempt to evade filter lists (e.g., EasyList [6], EasyPrivacy [7]) used to block ads and trackers, thus rendering content blocking ineffective. While both arms races persist to date, trackers are increasingly employing circumvention because counter-blocking efforts have not successfully persuaded users to disable content blocking tools [21, 44, 47].

Limitations of Prior Work. Trackers have been using increasingly sophisticated techniques to circumvent content blocking [17, 19, 36]. At a high level, circumvention techniques can be classified into two categories. One type of circumvention is achieved by frequently changing the network location (e.g., domain or URL) of advertising and tracking resources. Content blocking tools attempt to address this type of circumvention by updating filter lists promptly and more frequently [28, 29, 48, 49, 52]. The second type of circumvention is achieved by mixing up tracking resources with functional resources, such as serving both from the same network endpoint (e.g., first-party or Content Delivery Network (CDN)) [17, 20, 23]. Content blocking tools have struggled against this type of circumvention because they are in a no-win situation: they risk breaking legitimate functionality as collateral damage if they act and risk missing privacy-invasive advertising and tracking if they do not. While there is anecdotal evidence, the prevalence and modus operandi of this type of circumvention has not been studied in prior literature.

Measurement & Analysis. This paper aims to study the prevalence of mixed resources, which combine tracking and functionality, on the web. We present TrackerSift to conduct a large-scale measurement study of mixed resources at different granularities starting from network-level (e.g., domain and hostname) to code-level (e.g., script and method). TrackerSift's hierarchical analysis sheds light on how tracking and functional web resources can be progressively untangled at increasing levels of finer granularity. It uses a localization approach to untangle mixed resources beyond the script-level granularity of state-of-the-art content blocking tools. We show how

IMC '21, November 2–4, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487855

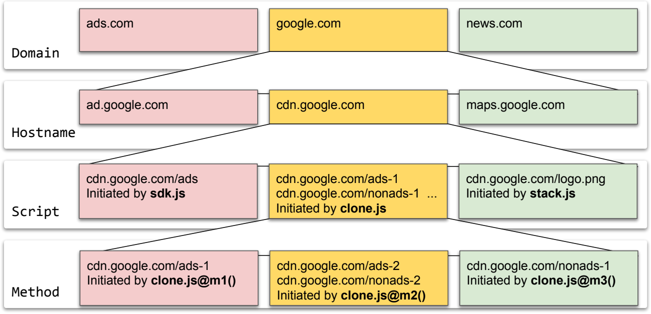


Figure 1: TrackerSift progressively classifies tracking (red) and functional (green) resources. For mixed resources (yellow), it proceeds to a finer granularity for further classification.

to classify methods in mixed scripts, which combine tracking and functionality, to localize the code responsible for tracking behavior. A key challenge in adapting software fault localization approaches to our problem is to find a rigorous suite of test cases (i.e., inputs labeled with their expected outputs) [32]. We address this challenge by using filter lists [6, 7] to label tracking and functional behaviors during a web page load. By pinpointing the genesis of a tracking behavior even when it is mixed with functional behavior (e.g., method in a bundled script), TrackerSift paves the way towards finer-grained content blocking that is more resilient against circumvention than state-of-the-art content blocking tools.

Results. Using TrackerSift, our measurements of 100K websites show that 17% of the 69.3K observed domains are classified as mixed. The requests belonging to mixed domains are served from a total of 26.0K hostnames. TrackerSift classifies 48% of these hostnames as mixed. The requests belonging to mixed hostnames are served from a total of 350.1K (initiator) scripts. TrackerSift classifies 6% of these scripts as mixed. The requests belonging to mixed scripts are initiated from a total of 64.0K script methods. TrackerSift classifies 9% of these script methods as mixed. Our analysis shows that the web resources classified as mixed by TrackerSift are typically served from CDNs or as inlined and bundled scripts, and that blocking them indeed results in breakage of legitimate functionality. While mixed web resources are prevalent across all granularities, TrackerSift is able to attribute 98% of the script-initiated network requests to either tracking or functional resources at the finest method-level granularity.

Our key contributions include:

- a **large-scale measurement and analysis** of the prevalence of mixed web resources; and
- a **hierarchical localization approach** to untangle mixed web resources.

2 TRACKERSIFT

In this section, we describe the design of TrackerSift to untangle mixed web resources. TrackerSift conducts a hierarchical analysis of web resources to progressively localize tracking resources at increasingly finer granularities if they cannot be separated as functional or tracking at a given granularity. TrackerSift needs a test oracle capable of identifying whether a web page's behavior (e.g., network requests) is tracking or functional. TrackerSift relies on filter lists, EasyList [6] and EasyPrivacy [7], to distinguish between tracking and functional behavior. As also illustrated in Figure 1, we next describe TrackerSift's hierarchical analysis at increasingly finer granularities of domain, hostname, script, and method.

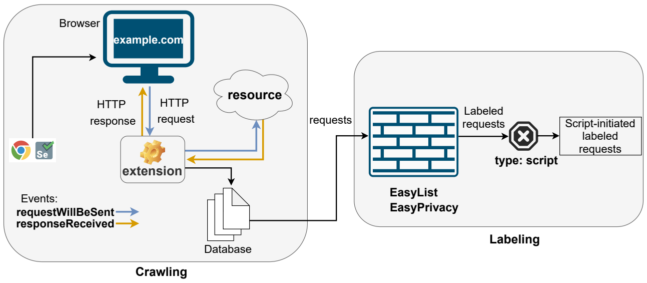


Figure 2: TrackerSift's web crawling and labeling

Domain classification. As a webpage loads, multiple network requests are typically initiated by scripts on the page to gather content from various network locations addressed by their URLs. We capture such script-initiated requests' URLs and apply filter lists to label them as tracking or functional. We then extract the domain names from request URLs and pass the label from URLs to domain names. For each domain, we maintain a tracking count and functional count. All the domains that are classified as tracking or functional are set aside at this level. The rest representing mixed domains serving both tracking and functional requests are further examined at a finer granularity. For instance, in Figure 1, the domain `ads.com` and `news.com` serve solely tracking and solely functional content, respectively. The domain `google.com` serves both and thus needs analysis at a finer granularity.

Hostname classification. At the domain level, we find the requests served by mixed domains and extract their hostnames. We increment the tracking and functional count for each hostname within mixed domains based on the corresponding request's label. The hostnames serving both tracking and functional requests are further analyzed at a finer granularity, while the rest are classified as either tracking or functional. In Figure 1, `google.com` was previously classified as mixed and therefore, all hostnames belonging to `google.com` need to be examined. We classify `ad.google.com` and `maps.google.com` as tracking and functional, respectively. In contrast, `cdn.google.com` is mixed and thus needs analysis at a finer granularity.

Script classification. We locate the script initiating the request to a mixed hostname and label it as either functional or tracking, reflecting the type of request they initiate. Like other levels, we measure the count of tracking and functional requests launched from each script and redistribute those into functional, tracking, and mixed scripts, where mixed scripts will be further analyzed at a finer granularity. In Figure 1, `sdk.js`, `clone.js`, and `stack.js` all initiate requests to the mixed hostname `cdn.google.com`. We classify `sdk.js` and `stack.js` as tracking and functional, respectively. Since `clone.js` requests both tracking and functional resources, it needs analysis at a finer granularity.

Method classification. We analyze the corresponding requests for each mixed script and locate the initiator JavaScript methods of each request. We then measure the number of tracking and functional requests initiated by each of the isolated methods. In the final step, we classify the methods into functional, tracking,

Table 1: Classification of requests at different granularities

Granularity	Tracking (Count)	Functional (Count)	Mixed (Count)	Separation Factor (%)	Cumulative Separation Factor (%)
Domain	755,784	566,810	1,129,109	54%	54%
Hostname	161,604	106,542	860,963	24%	65%
Script	235,157	490,295	135,511	84%	94%
Method	23,819	74,223	37,469	72%	98%

Table 2: Classification of resources at different granularities

Granularity	Tracking (Count)	Functional (Count)	Mixed (Count)	Separation Factor (%)
Domain	6,493	50,938	11,861	83%
Hostname	4,429	9,248	12,383	52%
Script	194,156	134,726	21,168	94%
Method	17,940	40,500	5,579	91%

and mixed. In Figure 1, for the mixed script `clone.js`, we classify `m1()` as tracking and `m3()` as functional. Since `m2()` requests both tracking and functional resources, it is classified as mixed.

3 DATA

In this section, we describe TrackerSift's browser instrumentation that crawls websites and labels the collected data. Note that TrackerSift's hierarchical analysis is post hoc and offline. Thus, it does not incur any significant overhead during page load other than the browser instrumentation and bookkeeping for labeling.

Crawling. We used Selenium [12] with Chrome 79.0.3945.79 to automatically crawl the landing pages of 100K websites that are randomly sampled from the Tranco top-million list [46] in April 2021. Our crawling infrastructure, based on a campus network in North America, comprised of a 13-node cluster with 112 cores at 3.10GHz, 52TB storage, and 832GB memory. Each node uses a Docker container to crawl a subset of 100K webpages. The average page load time (until `onLoad` event is fired) for a web page was about 10 seconds. Our crawler waits an additional 10 seconds before moving on to the next website. Note that the crawling is stateless, i.e., we clear all cookies and other local browser states between consecutive crawls.

As shown in Figure 2, our crawler was implemented as a purpose-built Chrome extension that used DevTools [8] API to collect the data during crawling. Specifically, it relies on two `network` events: `requestWillBeSent` and `responseReceived` for capturing relevant information for script-initiated network requests during the page load. The former event provides detailed information for each HTTP request such as a unique identifier for the request (`request_id`), the web page's URL (`top_level_url`), the URL of the document this request is loaded for (`frame_url`), requested resource type (`resource_type`), request header, request timestamp, and a `call_stack` object containing the initiator information and the stack trace for script-initiated HTTP requests. The latter event provides detailed information for each HTTP response, such as response headers and response body containing the payload.

Labeling. We gather authoritative source labels by applying filter lists to the crawled websites. Filter lists are not perfect (e.g., they are slow to update [49] and are prone to mistakes [17]) but they are the best available source of labels. We use two widely used filter lists that target advertising (EasyList [6]) and tracking (EasyPrivacy [7]). These filter lists mainly build of regular expressions that match advertising and/or tracking network requests. As shown in Figure 2, network requests that match EasyList or EasyPrivacy are classified as tracking, otherwise they are classified as functional. Note that we maintain the call stack that contains the ancestral scripts that in turn triggered a script-initiated network request (e.g., `XMLHTTPRequest` fetches). For asynchronous JavaScript, the stack track that preceded the request is prepended in the stack. Thus, for script-initiated network requests, we ensure that if a request is classified as tracking or functional, its ancestral scripts in the stack are also classified as such. Since network requests that are not script-initiated can not be trivially classified as tracking or functional, we exclude them from our analysis.

4 RESULTS

Classifying Mixed Resources. We compute the logarithmic ratio of the number of tracking to functional network requests to quantify the mixing of tracking and functional resources.

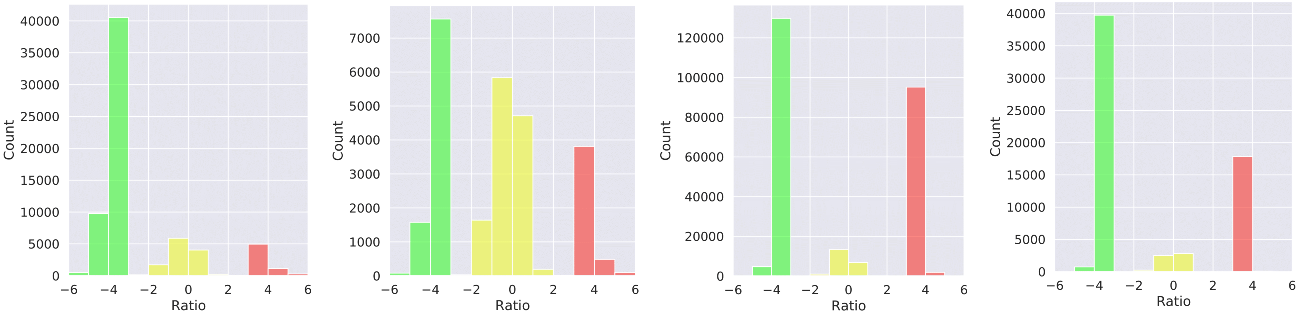
$$ratio = \log\left(\frac{\#\ of\ tracking\ requests}{\#\ of\ functional\ requests}\right) \qquad (1)$$

At each granularity, we classify resources with the common logarithmic ratio less than -2 as functional because they triggered 100× more functional requests than tracking requests. Similarly, we classify resources with the common logarithmic ratio of more than 2 as tracking because they triggered 100× more tracking requests than functional requests. The resources with the common logarithmic ratio between -2 and 2 are classified as mixed. We analyze the suitability of the selected classification threshold using sensitivity analysis later in Section 5.

Results Summary. Table 1 summarizes the results of our crawls of the landing pages of 100K websites. Using the aforementioned classification, we are able to attribute 54% of the 2.43 million script-initiated network requests to tracking or functional domains. The remaining 46% (1129K) of the 2.43 million requests attribute to mixed domains that are further analyzed at the hostname-level. We are able to attribute 24% of the requests from mixed domains to tracking or functional hostnames. The remaining 76% (860K) of the requests attribute to mixed hostnames that are further analyzed at the script URL-level. We are able to attribute 84% of the requests from mixed hostnames to tracking or functional script URLs. The remaining 16% (135K) of the requests attribute to mixed script URLs that are further analyzed at the script method-level. We are able to attribute 72% of the requests from mixed script URLs to tracking or functional script methods. This leaves us with less than 2% (37K) requests that cannot be attributed by TrackerSift to tracking or functional web resources and require further analysis.

Next, we analyze the distribution of the ratio of tracking to functional requests by web resources at different granularities of domain, hostname, script URL, and script method in Figure 3. Table 2 shows the breakdown of web resources classified as tracking, functional, and mixed at different granularities.

Domain classification. 2451K requests in our dataset are served from a total of 69,292 domains (eTLD+1). Figure 3a shows three distinct peaks: [2, ∞) serve tracking requests, (-∞, -2] serve functional



(a) domain **(b) hostname** **(c) script URL** **(d) script method**

Figure 3: Distribution of resources at increasingly finer granularities. Y-axis shows the count of unique (a) domains, (b) hostnames, (c) scripts, and (d) script methods. X-axis represents the common logarithmic ratio of the number of tracking to functional requests. Interval (-∞,-2] is classified as functional (green), (-2,2) is classified as mixed (yellow), and [2,∞) is classified as tracking (red).

requests, and (-2, 2) serve both tracking and functional requests. We can filter 31% of the requests by classifying 6,493 domains that lie in the [2, ∞) interval as tracking. Notable tracking domains include google-analytics.com, doubleclick.net, and googleadservices.com, bing.com. We can filter 23% of the requests by classifying 50,938 domains that lie in the (-∞, -2] interval as functional. Notable functional domains include CDN and other content hosting domains twimg.com, zychr.com, fbcdn.ne, w.org, and parastorage.com. However, 46% of requests are served by 11,861 mixed domains that lie in the (-2, 2) interval. These mixed domains cannot be safely filtered due to the risk of breaking legitimate functionality, and not filtering them results in allowing tracking. Notable mixed domains include gstatic.com, google.com, facebook.com, facebook.net, and wp.com.

Hostname classification. 1129K requests belonging to mixed domains are served from a total of 26,060 hostnames. Figure 3b shows three distinct peaks representing hostnames that serve tracking, functional, or both tracking and functional requests. We can filter 14% of the requests by classifying 4,429 hostnames that lie in the [2, ∞) interval as tracking. We can filter 9% of the requests by classifying 9,248 hostnames that lie in the (-∞, -2] interval as functional. However, 76% of the requests are served by 12,383 hostnames that lie in the (-2, 2) interval are classified as mixed. Again, these mixed hostnames cannot be safely filtered due to the risk of breaking legitimate functionality, and not filtering them results in allowing tracking. Take the example of hostnames of a popular mixed domain wp.com. The requests from wp.com are served from tracking hostnames such as pixel.wp.com and stats.wp.com, functional hostnames such as widgets.wp.com and c0.wp.com, and mixed hostnames such as i0.wp.com and i1.wp.com.

Script classification. 860K requests belonging to mixed hostnames are served from a total of 350,050 initiator scripts. Figure 3c again shows three distinct peaks representing scripts that serve tracking, functional, or both tracking and functional requests. We can filter 27% of the requests by classifying 194,156 scripts that lie in the [2, ∞) interval as tracking. We can filter 57% of the requests by classifying 134,726 scripts that lie in the (-∞, -2] interval as functional. However, 16% of the requests are served by 21,168 scripts that lie in the (-2, 2) interval are classified as mixed. These mixed scripts cannot be safely filtered due to the risk of breaking legitimate functionality, and not filtering them results in allowing tracking. For example, let’s analyze the initiator scripts of a mixed hostname i1.wp.com. The requests to this hostname are the result of different initiator scripts on the webpage www.ibn24.tv. Specifically, a tracking request to i1.wp.com is initiated by the script show_ads_impl_fy2019.js and a functional request to i1.wp.com is initiated by the script jquery.min.js. As another example, on the webpage somosinvictos.com, both tracking and functional requests to i1.wp.com are initiated by the mixed script lazysizes.min.js. Note that the scripts classified as tracking initiate requests to well-known advertising and tracking domains. For example, script uc.js served by consent.cookiebot.com initiated requests to googleadservices.com, doubleclick.net, and amazon-adsystem.com.

Method classification. 135K requests belonging to mixed scripts are served from a total of 64,019 script methods. Figure 3d again shows three distinct peaks representing methods that serve tracking, functional, or both tracking and functional requests. We can filter 17% of the requests by classifying 17,940 methods that lie in the [2, ∞) interval as tracking. We can filter 55% of the requests by classifying 40,500 methods that lie in the (-∞, -2] interval as functional. However, 28% of the requests are served by 5,579 methods that lie in the (-2, 2) interval are classified as mixed. These mixed methods cannot be safely filtered due to the risk of breaking legitimate functionality, and not filtering them results in allowing tracking. For example, let’s analyze script methods for a mixed script tfa.js on the webpage hubblecontacts.com. While both tracking and functional requests are initiated by the script, the tracking request was initiated by *get* method, and the functional request was initiated by *X* method. As another example, let’s analyze script methods for a mixed script app.js on the webpage radioshack.com.mx. In this case, both tracking and functional requests are initiated by the mixed script method *Pa.xhrRequest*.

5 DISCUSSION

In this section, we discuss some case studies, opportunities for future work, and limitations.

Circumvention strategies. There are two common techniques for mixing tracking and functional resources.

(1) *Script inlining:* Despite potential security risks, publishers are willing to inline external JavaScript code snippets (as opposed to including external scripts using the `src` attribute) for performance

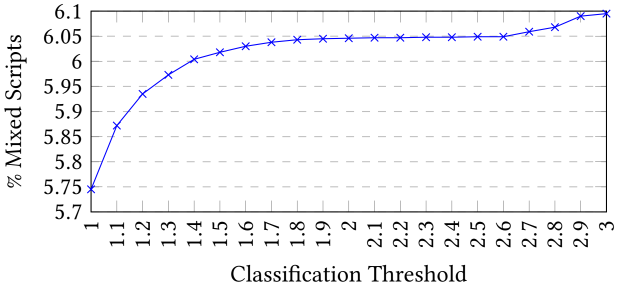


Figure 4: Sensitivity analysis of the classification threshold (default is -2 and 2) by studying the proportion of mixed scripts as a function of varying thresholds. The X-axis represents the threshold buckets. For example, 1.5 represents (-1.5, 1.5).

reasons as well as for circumvention [35, 41]. For example, we find that the Facebook pixel [9] is inlined on a large number of websites to assist with targeting Facebook ad campaigns and conversion tracking.

(2) *Script Bundling:* Publishers also bundle multiple external scripts from different organizations with intertwined dependencies for simplicity and performance reasons. JavaScript bundlers, such as webpack [14] and browserify [5], use dependency analysis to bundle multiple scripts into one or a handful of bundled scripts. For example, pressl.co serves a script app.9115af433836fd824ec7.js that is bundled using the webpack [14]. This bundled script includes the aforementioned Facebook pixel and code to load functional resources from a first-party hostname. Existing content blocking tools struggle to block inlined and bundled tracking scripts without the risk of breaking legitimate site functionality. Finer-grained detection by TrackerSift presents an opportunity to handle such scripts by localizing the methods that implement tracking.

Threshold sensitivity analysis. We set the classification threshold to a symmetric value of (-2,2) for classifying mixed resources in Equation 1. To assess our choice of the threshold, we analyze the sensitivity of script classification results in Figure 4. Similar trends are observed for domain, hostname, and method classification. The plot shows the percentage of scripts classified as mixed as we vary the threshold from 1 to 3 in increments of 0.1 Note that the curve plateaus around our selected threshold of 2. Thus, we conclude that our choice of the threshold is stable and reasonably separates mixed resources from tracking and functional resources.

Breakage analysis. We conducted manual analysis to assess whether blocking mixed resources results in breakage of legitimate functionality. To assess functionality breakage, we load a random sample of websites with (treatment) and without (control) blocking mixed scripts as classified by TrackerSift. We label breakage as: *major* if the core functionality such as search bar, menu, images, and page navigation is broken in treatment but not in control; *minor:* if the secondary functionality such as comment/review sections, media widgets, video player, and icons is broken in treatment but not in control; and *none:* if the core and secondary functionalities of the website are same in treatment and control. Note that we consider missing ads as no breakage. Table 3 shows our breakage analysis on a representative sample of 10 websites. We note major or minor breakage in all except one case. Thus, we conclude that mixed web resources indeed cannot be safely blocked by existing content blocking tools.

Table 3: Manual analysis of breakage caused by blocking mixed scripts on randomly selected 10 websites.

Website	Mixed Script	Breakage	Comment
caremanagementmatters.co.uk	jquery.min.js	Minor	scroll bar and two widgets missing
gratis.com	main.js	Major	page did not load
forevernew.com.au	require.js	Major	multiple page banners missing
flamesnation.ca	player.js	Minor	video pop missing
biba.in	MJ_Static-Built.js	Major	page did not load
ecomarket.ru	2.0c9c64b2.chunk.js	Major	page did not load
peachjohn.co.jp	jquery-1.11.2.min.js	Major	navigation and scroll bar missing
shoobs.com	widgets.js	None	no visible functionality breakage
editorajuspodivm.com.br	jquery.js	Major	navigation and scroll bar missing
resourceworld.com	jquery.min.js	Major	navigation bar and images missing

Blocking mixed scripts. When TrackerSift classifies a mixed script with different tracking and functional methods, we can simply remove tracking methods to generate a surrogate script that can then be used to shim the mixed script at runtime. Existing content blockers such as NoScript, uBlock Origin, AdGuard, and Firefox SmartBlock use surrogate scripts to block tracking by mixed scripts while avoiding breakage [3, 10, 13, 38]. However, these surrogate scripts are currently manually designed [11]. TrackerSift can help scale up the process of generating surrogate scripts by automatically detecting and removing tracking methods in mixed scripts. Note that removing tracking methods is tricky because simply removing them risks functionality breakage due to potential coverage issues of dynamic analysis. To mitigate this concern, we plan to explore a more conservative approach using a guard—a predicate that blocks tracking execution but allows functional execution. Such a predicate has a similar structure to that of an `assertion`. We envision using classic invariant inference techniques [25, 43] on a tracking method's calling context, scope, and arguments to generate a program invariant that holds across all tracking invocations. If an online invocation satisfies the invariant, the guard will block the execution. A key challenge in this approach is collecting the context information, e.g., program scope, method arguments, and stack trace, for each request initiated by the mixed method at runtime. We plan to address these challenges in leveraging TrackerSift for generating safe surrogate scripts in our future work.

Blocking mixed methods. Our analysis shows that TrackerSift's separation factor is 91% even at the finest granularity. This leaves 5.6K mixed methods that cannot be safely blocked. One possible direction is to apply TrackerSift in the *context* of a mixed method initiating a request. We can define *context* as calling context, program scope, or parameters to the mixed method. In the case of calling context, we can perform a call stack analysis that takes a snapshot of a mixed method's stack trace when the method initiates a tracking or functional request. We hope to see distinct stack traces from tracking and functional requests by a mixed method. We can consolidate the stack traces of a mixed method and locate the point of divergence, i.e., a method in the stack trace that only participates in tracking requests. We hypothesize that removing such a method will break the chain of methods needed to invoke a tracking behavior, thus removing the tracking behavior.

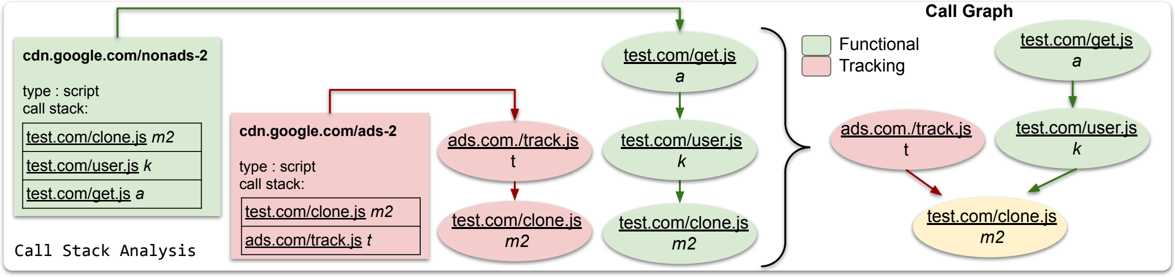


Figure 5: Call stack analysis for the requests *ads-2* and *nonads-2* that can not be separated at method level i.e. *m2*. Call stack is analyzed to identify the first point of divergence i.e *track.js t* and it could be removed to block the tracking request.

Figure 5 illustrates our proposed call stack analysis. It shows the snapshot of stack traces of requests `nonads-2` and `ads-2`. These requests are initiated by a mixed method `m2()` on the webpage. The two stack traces are merged to form a call graph where each node represents a unique script and method, and an edge represents a caller-callee relationship. The yellow color indicates that a node participates in invoking both tracking and functional requests. `t` in track.js is the point of divergence since it only participates in the tracking trace. Therefore, `t` is most likely to originate a tracking behavior which makes it a good candidate for removal.

Limitations. We briefly acknowledge a few limitations our measurement and analysis. First, our web crawls do not provide full coverage of the events triggered by user interactions (e.g., scroll, click). This is a general limitation of dynamic analysis and can be mitigated by using a forced execution framework to execute other possible paths [33]. Second, our method-level analysis does not distinguish between different anonymous functions in a script and treats them as part of the same method. This limitation can be addressed by using the line and column number information available for each method invocation in the call stack. Finally, our web crawls are limited to the landing pages and the results might vary for internal pages [18]. As part of our future work, we plan to deploy TrackerSift on internal pages as well.

6 RELATED WORK

We summarize closely related work documenting anecdotal evidence of circumvention by mixing up tracking and functional resources. Most notably, Alrizah et al. [17] and Chen et al. [20] showed how first-party hosting and script inlining or bundling is being used by trackers to circumvent filter lists used by content blockers. Alrizah et al. [17] documented a variety of attacks on content blocking tools, including both counter-blocking and circumvention attacks. Among other things, they showed that some websites circumvent filter lists by mixing tracking and functional resources through techniques such as script inlining. These websites essentially have a "self-defacement" strategy, where content blockers risk breaking legitimate functionality as collateral damage if they act and risk missing privacy-invasive advertising and tracking if they do not. Chen et al. [20] leveraged their JavaScript signature approach to document about 500 false negative cases where tracking scripts were inlined or bundled for successful circumvention. Relatedly, trackers have started to exploit techniques such as CDN proxies (i.e., serve functional and tracking resources from the same CDN server) [36] and CNAME cloaking (i.e., masquerade third-party tracking resources from first-party using a minor change in DNS records) [22, 23] to assist with implementing these circumvention techniques.

The problem of localizing tracking-inducing code shares similarities with prior research on fault-inducing code localization. For example, spectra-based fault localization (SBFL) [16, 24, 31, 32, 45, 50] collect statement coverage profiles of each test, passing or failing, to localize the lines of code that are most likely to induce a test failure. Bela et al. [51] and Laghari et al. [34] presented a call frequency-based SBFL technique. Instead of coverage information, they use the frequency of method occurrence in the call stack of failing test cases. A method that appears more in the failing call stack of failing test cases is more likely to be faulty. In TrackerSift, methods responsible for more frequently initiating tracking requests than functional requests is classified as tracking. Abreu et al. [15] studied how accurate these SBFL techniques are, and their accuracy is independent of the quality of test design. Jiang et al. [30] used call stack to localize the null pointer exception, and Gong et al. [27] generated call stack traces to successfully identify 65% of the root cause of the crashing faults. One common limitation across most fault-localization approaches is that they require an extensive test suite capable of exercising faulty behavior, along with an instrumented runtime to collect statement-level coverage. TrackerSift overcomes these limitations by using filter lists as test oracle during page load time and uses an instrumented browser to capture fine-grained coverage.

7 CONCLUSION

We presented TrackerSift, a hierarchical approach to progressively untangle mixed resources at increasing levels of finer granularity from network-level (e.g., domain and hostname) to code-level (e.g., script and method). We deployed TrackerSift on 100K websites to study the prevalence of mixed web resources across different granularities. TrackerSift classified more than 17% domains, 48% hostnames, 6% scripts, and 9% methods as mixed. Overall, TrackerSift was able to attribute 98% of all requests to tracking or functional resources by the finest level of granularity. Our results highlighted opportunities for finer-grained content blocking to remove mixed resources without breaking legitimate site functionality. TrackerSift can be used to automatically generate surrogate scripts to shim mixed web resources.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under grant numbers 2051592, 2102347, 2103038, 2103439, and 2106420. We would like to thank our shepherd, Paul Barford, and the anonymous IMC reviewers, for their constructive feedback. We would also like to thank Haris Amjad for his valuable input to help improve the quality of visualizations in the paper.

REFERENCES

[1] 2020. gorhill/uBlock: uBlock Origin - An efficient blocker for Chromium and Firefox. Fast and lean. https://github.com/gorhill/uBlock.

[2] 2021. Adblock Plus. https://adblockplus.org/.

[3] 2021. AdGuard Scriptlets and Redirect resources. https://github.com/AdguardTeam/Scriptlets.

[4] 2021. Brave Browser. https://brave.com/.

[5] 2021. Browserify. https://browserify.org/

[6] 2021. EasyList. https://easylist.to/easylist/easylist.txt.

[7] 2021. EasyPrivacy. https://easylist.to/easylist/easyprivacy.txt.

[8] 2021. Extending DevTools. https://developer.chrome.com/docs/extensions/mv3/devtools/

[9] 2021. Facebook Pixel: Implementation. https://developers.facebook.com/docs/facebook-pixel/implementation/

[10] 2021. Firefox 87 introduces SmartBlock for Private Browsing. https://blog.mozilla.org/security/2021/03/23/introducing-smartblock/

[11] 2021. Security/TrackingProtectionBreakage. https://wiki.mozilla.org/Security/TrackingProtectionBreakage#Trivial_shim_needed_to_avoid_breakage.3B_no_yellowlisting_required

[12] 2021. Selenium. http://docs.seleniumhq.org/. http://docs.seleniumhq.org/

[13] 2021. uBO-Scriptlets: A custom arsenal of scriptlets to be used for injecting userscripts via uBlock Origin. https://github.com/uBlock-user/uBO-Scriptlets,.

[14] 2021. webpack. https://webpack.js.org/.

[15] Rui Abreu, Peter Zoeteweij, and Arjan J.C. van Gemund. 2007. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*.

[16] Pragya Agarwal and Arun Prakash Agrawal. 2014. Fault-Localization Techniques for Software Systems: A Literature Review. *SIGSOFT Softw. Eng. Notes* (2014).

[17] Mshabab Alrizah, Sencun Zhu, Xinyu Xing, and Gang Wang. 2019. Errors, Misunderstandings, and Attacks: Analyzing the Crowdsourcing Process of Adblocking Systems. In *ACM Internet Measurement Conference (IMC)*.

[18] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M. Maggs. 2020. On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. In *Proceedings of the ACM Internet Measurement Conference*.

[19] Muhammad Ahmad Bashir, Sajjad Arshad, Engin Kirda, William Robertson, and Christo Wilson. 2018. How Tracking Companies Circumvented Ad Blockers Using WebSockets. In *Proceedings of the Internet Measurement Conference (IMC)*.

[20] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. 2021. Detecting Filter List Evasion With Event-Loop-Turn Granularity JavaScript Signatures. In *IEEE Symposium on Security and Privacy*.

[21] Yuyu Chen. 2016. 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/.

[22] Romain Cointepas. 2019. CNAME Cloaking, the dangerous disguise of third-party trackers. https://medium.com/nextdns/cname-cloaking-the-dangerous-disguise-of-third-party-trackers-195205dc522a.

[23] Ha Dao, Johan Mazel, and Kensuke Fukuda. 2020. Characterizing CNAME Cloaking-Based Tracking on the Web. *IEEE/IFIP TMA'20* (2020), 1–9.

[24] Marwa El-Wahab, Amal Aboutabl, and Wessam El-Behaidy. 2018. Graph Mining for Software Fault Localization: An Edge Ranking based Approach. *Journal of Communications Software and Systems* 13 (01 2018), 178–188. https://doi.org/10.24138/jcomss.v13i4.402

[25] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. 1999. Dynamically Discovering Likely Program Invariants to Support Program Evolution. In *Proceedings of the 21st International Conference on Software Engineering* (Los Angeles, California, USA) *(ICSE '99)*. Association for Computing Machinery, New York, NY, USA, 213–224. https://doi.org/10.1145/302405.302467

[26] Kiran Garimella, Orestis Kostakis, and Michael Mathioudakis. 2017. Ad-Blocking: A Study on Performance, Privacy and Counter-Measures. In *Proceedings of the 2017 ACM on Web Science Conference*.

[27] Liang Gong, Hongyu Zhang, Hyunmin Seo, and Sunghun Kim. 2014. Locating Crashing Faults based on Crash Stack Traces. In *arXiv:1404.4100*.

[28] Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. 2017. The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists. In *IMC*.

[29] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. 2020. AdGraph: A Graph-Based Approach to Ad and Tracker Blocking. In *Proceedings of the IEEE Symposium on Security & Privacy*.

[30] Shujuan Jiang, Wei Li, Haiyang Li, Yanmei Zhang, Hongchang Zhang, and Yingqi Liu. 2012. Fault Localization for Null Pointer Exception Based on Stack Trace and Program Slicing. In *2012 12th International Conference on Quality Software*.

[31] James A. Jones and Mary Jean Harrold. 2005. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering* (Long Beach, CA, USA) *(ASE '05)*. Association for Computing Machinery, New York, NY, USA, 273–282. https://doi.org/10.1145/1101908.1101949

[32] James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of Test Information to Assist Fault Localization. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 467–477. https://doi.org/10.1145/581339.581397

[33] Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu. 2017. J-force: Forced execution on javascript. In *Proceedings of the 26th international conference on World Wide Web*. 897–906.

[34] Gulsher Laghari, Alessandro Murgia, and Serge Demeyer. 2015. Localising Faults in Test Execution Traces. In *Proceedings of the 14th International Workshop on Principles of Software Evolution* (Bergamo, Italy) *(IWPSE 2015)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/2804360.2804361

[35] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *Network and Distributed System Security Symposium (NDSS)*.

[36] Hieu Le, Athina Markopoulou, and Zubair Shafiq. 2021. CV-Inspector: Towards Automating Detection of Adblock Circumvention. In *Network and Distributed System Security Symposium (NDSS)*.

[37] Matthew Malloy, Mark McNamara, Aaron Cahn, and Paul Barford. 2016. Ad Blockers: Global Prevalence and Impact. In *ACM Internet Measurement Conference (IMC)*.

[38] Giorgio Maone. [n.d.]. Surrogate Scripts vs Google Analytics. https://hackademix.net/2009/01/25/surrogate-scripts-vs-google-analytics/.

[39] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar R. Weippl. 2017. Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools. In *IEEE European Symposium on Security and Privacy*.

[40] Muhammad Haris Mughees, Zhiyun Qian, and Zubair Shafiq. 2017. Detecting Anti Ad-blockers in the Wild . In *Privacy Enhancing Technologies Symposium (PETS)*.

[41] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions. In *ACM Conference on Computer and Communications Security (CCS)*.

[42] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E. Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J. Murdoch. 2016. Adblocking and Counter-Blocking: A Slice of the Arms Race. In *USENIX Workshop on Free and Open Communications on the Internet*.

[43] Saswat Padhi, Rahul Sharma, and Todd Millstein. 2016. Data-Driven Precondition Inference with Learned Features. *SIGPLAN Not.* 51, 6 (June 2016), 42–56. https://doi.org/10.1145/2980983.2908099

[44] Page Fair. 2017. The State of the Blocked Web. https://pagefair.com/downloads/2017/01/PageFair-2017-Adblock-Report.pdf.

[45] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and Improving Fault Localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 609–620. https://doi.org/10.1109/ICSE.2017.62

[46] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2018. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156* (2018).

[47] Kaleigh Rogers. 2018. 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.

[48] Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. 2021. WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking. *arXiv preprint arXiv:2107.11309* (2021).

[49] Alexander Sjosten, Peter Snyder, Antonio Pastor, Panagiotis Papadopoulos, and Benjamin Livshits. 2020. Filter List Generation for Underserved Regions. In *The Web Conference*.

[50] H. A. D. Souza, M. L. Chaim, and Fabio Kon. 2016. Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges. *ArXiv* abs/1607.04347 (2016).

[51] Béla Vancsics, Ferenc Horváth, Attila Szatmári, and Arpád Beszédes. [n.d.]. Call Frequency-Based Fault Localization. ([n. d.]).

[52] Antoine Vastel, Peter Snyder, and Benjamin Livshits. 2020. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced AdBlocking. In *ACM SIGMETRICS/Performance*.