KHALEESI: Breaker of Advertising and Tracking Request Chains

Umar Iqbal
University of Washington

Charlie Wolfe
University of Iowa

Charles Nguyen
UC Davis

Steven Englehardt[‡]
DuckDuckGo

Zubair Shafiq
UC Davis

Abstract

Request chains are being used by advertisers and trackers for information sharing and circumventing recently introduced privacy protections in web browsers. There is little prior work on mitigating the increasing exploitation of request chains by advertisers and trackers. The state-of-the-art ad and tracker blocking approaches lack the necessary context to effectively detect advertising and tracking request chains. We propose KHALEESI, a machine learning approach that captures the essential sequential context needed to effectively detect advertising and tracking request chains. We show that KHALEESI achieves high accuracy, that holds well over time, is generally robust against evasion attempts, and outperforms existing approaches. We also show that KHALEESI is suitable for online deployment and it improves page load performance.

1 Introduction

Request chains, most commonly implemented using HTTP redirects, enable several important web functionalities such as URL shortening [23], protocol upgrades [31], CDN request routing [27], etc. They have also been used by advertisers and trackers to implement cookie syncing as part of programmatic online advertising [21,26,38,39,57,58]. As mainstream browsers have recently implemented countermeasures against third-party cookies [10, 12], advertisers and trackers have started to use request chains to circumvent these privacy protections. For example, request chains are being used to implement bounce tracking [17,69,71] — a tracking technique that advertisers and trackers use to circumvent third-party cookie blocking by forcing users to visit them in the first-party context. Request chains have also been used to generate HTTP Strict Transport Security (HSTS) super cookies [16,70].

While the malicious use of request chains for drive-by malware download, spam, and phishing has been extensively studied [30,34,41,46,52,54,56], the research community has only recently started to look into their recent increased use by advertisers and trackers and effective countermeasures are still lacking [51]. The state-of-the-art approaches to detecting and blocking advertising and tracking resources are generally limited to analyzing requests individually. A slew of heuristic and machine learning (ML) approaches have been proposed to analyze information in HTTP request headers and payloads to detect advertising and tracking requests [28,45,62]. These approaches cannot effectively detect advertising and tracking request chains since they lack the necessary context to do so. Several approaches target cookie syncing enabled by request chains using information in HTTP request and response headers and payloads [26,38,39,58]. These approaches narrowly target request chains implementing a specific behavior such as cookie syncing and cannot detect a variety of other advertising and tracking behaviors implemented using request chains such as bounce tracking. On a similar note, some browsers such as Safari [68] defend against specific abuses of request chains [40, 55, 75]. However, Safari's Intelligent Tracking Prevention (ITP) considers the appearance of a request in a chain but not what it actually does. Thus, since ITP does not capture the available sequential context, it is prone to misclassify benign request chains, limiting its utility to storage restrictions, e.g., cookie clearing [68], rather than outright request blocking.

We propose KHALEESI, a machine learning approach that focuses on sequential context to detect advertising and tracking request chains. Specifically, we design a light-weight representation of request chains to capture the sequence of requests and responses. This representation allows us to effectively capture the interactions between chains of interrelated requests triggered by HTTP redirects or embedded scripts. We leverage this purpose-built representation to extract features that capture this context and train our classifier. KHALEESI's classifier makes a new classification decision in an online fashion as a new request in a chain is loaded. This capitalization of sequential context enables KHALEESI to more accurately and efficiently detect advertising and tracking request chains than prior approaches.

We evaluate KHALEESI across various browser configura-

[‡]The majority of this work was completed while Steven was at Mozilla.

tions covering landing and internal pages on top-10K websites. The results show that KHALEESI classifies request chains, which account for one-third of all requests in our crawls, with an accuracy ranging from 98.63–99.95%. KHALEESI outperforms prior approaches by 3.51–40.07 percentage points in terms of accuracy. KHALEESI is also generally more robust against evasion attempts such as domain rotation, URL randomization, and CNAME cloaking. We also show that KHALEESI's accuracy holds well over time, degrading less than 5% after 8 months. Moreover, KHALEESI improves page load performance on 91.26% of the websites as compared to stock Firefox and 59.82% of the websites as compared to Firefox with Adblock Plus.

Our in-depth analysis of KHALEESI's deployment sheds light on the information sharing ecosystem enabled by request chains. We build a request chain graph to understand bilateral information sharing relationships between different entities and find that despite individual request blocking, through ad and tracker blocking extensions, trackers can still share information with each other via request chains. Our findings show that KHALEESI's improved accuracy helps significantly reduce the proliferation of tracking in the request chain graph. We also find emerging use cases of request chains to implement bounce tracking for circumventing recently introduced restrictions on third-party cookies.

In summary, our key contributions are as follows:

1. We present KHALEESI, an ML approach that capitalizes on sequential context to detect advertising and tracking request chains.
2. We conduct a rigorous evaluation of KHALEESI's accuracy and robustness in detecting advertising and tracking request chains and compare it against prior approaches.
3. We present a lightweight implementation of KHALEESI as a browser extension. We show that KHALEESI is feasible for online deployment and improves page load performance.
4. We use KHALEESI to analyze how advertisers and trackers use request chains for information sharing and circumvention.

2 Background & Related Work

2.1 Background

A URL redirect (short for redirection) is a standard technique to forward a user's request for a resource to another address. Redirects form a *chain* of interrelated requests that are often used to share data between trackers. In a typical HTTP request, the browser will send a request to a server for a resource at a specific location. The server evaluates the request and responds with the requested resource if it is available. However, if the server cannot fulfill the request on its own then it may forward the browser to another server. To this end, the server responds with an HTTP 3XX response code and includes a new URL for the resource in the `Location` response header. The browser then issues a subsequent HTTP request to the new URL and the same process ensues. Subsequent request-response sequences form a *request chain*.

Request chains can be implemented at multiple layers in the web stack: via HTTP redirects, HTML redirects, and in JavaScript.[1] *HTTP-layer request chains* are implemented as part of the HTTP protocol using 3XX response codes. In this type of request chain, a browser will be bounced between locations on one or more servers to retrieve content for a single resource. *HTML-layer request chains* are implemented through *meta refreshes*. This type of request chain occurs when the HTML of a page includes a `meta` tag that specifies `http-equiv="Refresh"` with a new URL in the `url` attribute. Because they rely on HTML, meta refresh redirection can only occur at the frame level. *JavaScript-layer request chains* can be implemented in a number of ways. Request chains can be built in the top-level frame when a script updates `window.location` to a new URL. Request chains can be built from embedded resources when a script loads a chain of resources with intertwined dependencies [46]. A recent measurement study sampled sites across the Alexa top 1M and found that almost 80% of them use HTTP redirects and 35% use JavaScript-based top-level redirects [61].

Request chains are most often built from redirects. Redirects serve many necessary functions on the web: they allow websites to seamlessly migrate content between hostnames, upgrade connections to more secure protocols (i.e., HTTP to HTTPS), and implement URL shortening services. However, redirects are also often used for abuse. Most notably, redirects are used to obscure URLs that distribute spam or other types of malware [56]. Request chains also play a core role in online advertising: they allow advertisers and trackers to share tracking information across origins [21, 26, 57, 58].

Cookie Syncing. Advertisers most often track users across the web using client-side identifiers stored in cookies. When advertisers wish to collaborate, they first need to "sync" these identifiers with each other. Advertisers cannot directly share these identifiers with each other due to the same-origin policy. To bypass this restriction, advertisers embed their identifiers in requests to other advertisers. Such information sharing between different origins is called *cookie syncing* [21, 39, 57]. Prior work has shown that some tracking origins sync cookies with 100+ other origins [38], and that syncing enhances their coverage by as much as a 7× [58].

Bounce Tracking. To combat cookie-based cross-site tracking, several browsers, such as Safari and Firefox, now block third-party cookies [6, 68].[2] Advertisers have been

[1] DNS CNAME records also provide a form of redirection at the DNS level, but we do not study these.

[2] Safari blocks all third-party cookies and Firefox blocks third-party cookies from known trackers.

found to bypass third-party cookie blocking by "bouncing" the browser through a tracking website during an otherwise unrelated top-level navigation (i.e., by using a top-level redirect) [55, 71]. When embedded as a third-party, the bounce tracker has no access to cookies. However, once a tracker is visited directly during a "bounce" it can read and write cookies as a first-party. This allows it to associate tracking data with identifiers stored in the users cookies. A recent study showed that as many as 87% of popular websites might be bypassing third-party cookie restrictions using such techniques [61].

2.2 Related Work

There are no purpose-built countermeasures against advertising and tracking request chains. Existing countermeasures generally operate at the level of individual requests even if they are occurring as part of a request chain. These countermeasures do not fully take into account the available context of request chains. Based on the context they leverage, existing countermeasures can be divided into three categories: (1) approaches that only use request information, (2) approaches that use both request and response information, and (3) approaches that use both request and response information in a sequential manner.

There is prior work on detecting malicious use of request chains such as drive-by malware download [30, 56]. However, these approaches cannot be readily repurposed to detect advertising and tracking request chains due to the inherent differences between malware and tracking. Conceptually, malware and tracking fundamentally differ, both in their goals and implementation. Specifically, malware typically attempts to directly compromise a victim's device. In contrast, trackers do not directly aim to compromise a victim's device. They instead aim to harvest/exfiltrate information that can then be used to identify and track users across the web. Further, malware attempts to hide and uses request chains (e.g., URL shorteners [34, 54]) to this end. In contrast, trackers use request chains to carry out their regular activity in a fairly standardized and distinct manner (e.g., for cookie syncing [43]) compared to malware. Due to these differences, advertising and tracking request chains have distinct features that are not captured by prior approaches for detecting malicious request chains. For example, prior work on detecting malicious request chains [30, 56] only consider aggregate sequential information (e.g., number of redirects with obfuscated request URLs) and generally ignore request and response properties, which are crucial to detect advertising and tracking behaviors in request chains.

Request based detection. Content blockers, such as Adblock Plus [2], are the most commonly used defense against advertising and tracking. Content blockers rely on filter lists (e.g., EasyList [8]). Prior research has tried to use machine learning (ML) on request information to enhance filter lists. For example, Bhagavatula et al. [28] extracted features from the URL (e.g., query string parameters) to train different ML classifiers for detecting advertising requests. However, filter lists ultimately rely on a subset of information available in HTTP requests: the request URL, the content type of the requested resource, and the top level domain of page. Thus, filter lists cannot detect if trackers engage in cookie syncing because they do not look at the HTTP `Cookie` header. Filter lists do not use sequential context that reflects the *intent* of the request occurring in a chain. The lack of consideration of sequential context also contributes to their susceptibility to adversarial evasion [22, 35, 65].

Request and response based detection. Some approaches [15, 45, 76] have tried to use both request and response information to detect ads and trackers using ML classifiers. Yu et al. [76] proposed to detect trackers by observing the unique values shared by a significant number of third parties. Privacy Badger [15] labels third-party domains as trackers if they set cookies on three or more websites. Gugelmann et al. [45] proposed to use HTTP request and response meta data, such as the size of requests and whether cookies are set, to train their classifier that detects ads and trackers. Other approaches [38, 39, 58] have been purpose-built to detect cookie syncing in request chains. Papadopoulos et al. [58] proposed to detect cookie syncing events using an ML classifier by relying on keywords in HTTP requests, such as domain name and URL parameters, as features. Both Fouad et al. [39] and Englehardt and Narayanan [38] proposed to detect cookie syncing by observing identifiers across consecutive requests and responses. Though these approaches perform better than the request based detection approaches, they are far from ideal because they are only able to observe a subset of the advertising and tracking behaviors that occur in a request chain. For example, these approaches cannot observe bounce tracking because it requires an analysis of multiple request and response pairs in a sequential manner. Similar to request based approaches, request and response based approaches are also susceptible to adversarial evasion [22, 35, 65] because of their reliance on domain and parameter names (as we demonstrate later in § 4.3.2).

Request and response sequence based detection. Recent ML-based approaches have proposed using sequential context to detect ads and trackers, including Safari's Intelligent Tracking Prevention (ITP) [68], Brave's AdGraph/PageGraph [49, 64], and WebGraph [63]. ITP uses an ML classifier that detects third-party trackers by monitoring the redirects from third parties to other domains, the presence of third parties on other websites as a resource, and the presence of third parties on other websites as `iframes` [74]. Though ITP utilizes partial sequential information, by monitoring the number of redirects, it does not capitalize on tracking information revealed in the requests and responses. Furthermore, ITP is a "post-hoc" approach to tracker detection, i.e,. ITP is only able to detect tracking after it has observed the tracking behavior. Ad-

Graph/PageGraph [49, 64] use a graph-based ML approach to detect ads and trackers by monitoring their interactions across HTML, HTTP, and JavaScript layers. While AdGraph implicitly uses sequential context that is obtained through structural graph properties, it does not explicitly capture HTTP redirects and HTTP request header information [13, 47]. The absence of redirects and request header information makes AdGraph/-PageGraph oblivious to requests that are part of request chains. More recently, WebGraph [63] improves AdGraph's robustness by adding features that capture the flow of information from one entity to the browser's storage, the network, and to other entities loaded on a page as well as by excluding content features. Different from AdGraph, WebGraph incorporates HTTP redirects and HTTP request header information in its graph representation. While WebGraph's graph representation does capture HTTP redirects, it does not capitalize on the sequential nature of HTTP requests in a redirect as well as information in HTTP responses. Moreover, due to significant performance overheads, WebGraph operates in an offline manner and is envisioned to generate filter lists to block advertising and tracking resources, making it susceptible to the same issues that hamper filter lists [22, 35, 65].

In conclusion, existing approaches do not fully consider the necessary sequential context and thus cannot effectively detect advertising and tracking request chains. To bridge that gap, we next propose KHALEESI, an ML approach that captures the sequential context in request chains to effectively detect advertising and tracking request chains. As compared to prior work, KHALEESI is novel in two main aspects: (1) it is tailor-made to detect advertising and tracking request chains (see § 3) and (2) it is thoroughly evaluated on simulated real-world browser configurations and conditions (see § 4).

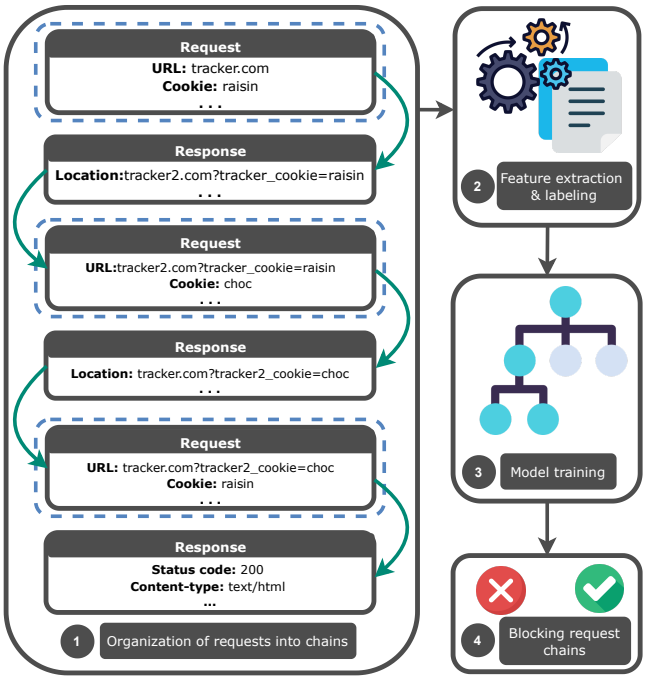


Figure 1: KHALEESI: (1) We first organize network and JavaScript-layer requests into chains. (2) We then label the request chains with ad and tracker blocking filter lists and extract features from them. (3) We use the extracted features to train a machine learning model and (4) use it to detect and block advertising and tracking request chains. The example cookie syncing request chain in (1) demonstrates the benefit of sequential context leveraged by KHALEESI. Specifically, the dotted blue blocks represent the context utilized by existing approaches in isolation. Whereas the green arrows represent the increasing sequential context utilized by KHALEESI.

3 KHALEESI

In this section, we present the design and implementation of KHALEESI, a machine learning based approach that uses sequential context for the early detection of advertising and tracking request chains. At a high level, KHALEESI organizes requests into a chain-like structure which captures the sequential context of interrelated requests. It then leverages this sequential context to train a machine learning classifier to effectively detect advertising and tracking resources. Figure 1 gives an overview of KHALEESI.

3.1 Motivation & Key Idea

Existing approaches detect individual advertising and tracking requests in isolation, even when they are part of a request chain. They utilize only partial request information, which might indicate what the resource is (i.e., a *tracker*) but not what the resource does (i.e., *tracking*). Such partial consideration of request information not only makes existing approaches less accurate, but also susceptible to evasion attempts by advertisers and trackers. Prior research has shown that the URL-based ad and tracker detection approaches are vulnerable to hostname and URL path randomization [22, 67]. There have been several instances in the wild, where determined advertisers and trackers have used domain generation algorithms (DGA), to rotate domains, to evade ad and tracker blocking [35, 77]. Since redirects provide an apparatus to load resources from different endpoints at the runtime, evasions attempts (e.g., domain rotation) are even more applicable.

However, the sequentially chained nature of many advertising and tracking requests puts us in an advantageous position to detect them. For example, we can easily observe cookie syncing—a fundamental component of advertisement-related tracking—by analyzing the sequential chain of requests. KHALEESI is the first privacy-enhancing blocking approach that leverages the sequential context available in request chains for early detection. To demonstrate the benefit of the sequential context leveraged by KHALEESI, we show an example of a cookie syncing request chain implemented using

HTTP redirects in Figure 1 (1). The dotted blue blocks show the visibility of existing request-based detection approaches and the green arrows show KHALEESI's visibility of sequential context. Existing approaches will use incomplete request and response information and will fail to link the redirects in a sequence. Thus, these approaches will miss the fact that the resources are cookie syncing. In contrast, KHALEESI operates with a much richer context: it includes information from requests, responses, and their sequential connectivity. This allows KHALEESI to make a classification decision that incorporates aspects of cookie syncing visible only when multiple request-response pairs are examined.

3.2 Request Chain Construction

KHALEESI's request chains capture a wide range of advertising and tracking behaviors such as cookie syncing [18], bounce tracking [69], and HSTS [16], that are known to exploit information across multiple requests. KHALEESI captures advertising and tracking behaviors from both network-layer and JavaScript-layer request sequences.

Network-layer request chains are constructed by linking HTTP requests that instruct the browser to initiate a redirect. Redirects are continuously linked until the response specifies that the request is complete. We also include Javascript-initiated top-level frame redirections in this category. Top-level redirects can be triggered in a number of ways, including by an HTTP 3XX response status or by HTTP response that includes embedded JavaScript to automatically navigate the frame to a new URL (e.g., via `window.location`). Regardless of how the redirects are triggered, the end result is the same: the server sending the HTTP response forwards the browser to another server.

Network-layer request chains provide decentralized control because each server in the chain may choose to redirect the browser to a new location. Decentralized control is preferred in use cases where each entity wants to control the navigation flow. For example, cookie syncing is a use case where decentralized control is preferred because each entity decides who they want to sync cookies with.

JavaScript-layer request chains are constructed by linking together JavaScript-initiated HTTP requests initiated by the same script. Specifically, we intercept the JavaScript stack trace each time a request is initiated and associate the request to the script at the top of the stack. However, not all HTTP requests that originate from a script are interrelated. For example, tag management scripts will initiate a bunch of unrelated HTTP requests. We filter out unrelated requests by only linking requests that share identifiers with each other. Below we describe our request linking method:

1. Tokenize the values stored in the Cookie and Set-Cookie headers, the query string parameters from requests and referrers, and the values of non-standard HTTP headers. Values are split on any character other than `a-zA-Z0-9_=-`.
2. Filter out the tokens that have fewer than 8 characters to prevent false matches with common identifiers, such as `en-US`.
3. Consider plain, Base64 encoded, MD5 hashed, and SHA-1 hashed versions of the tokens as identifiers.
4. Match the identifiers from the preceding request with the cookies, query string parameters, and non-standard HTTP request headers of all future requests.
5. If there is a match, we consider the request a part of the chain.

In principle, subsequent JavaScript-layer requests achieve the same objective as HTTP redirects and they have been linked into chains in prior work [46]. In contrast with network-layer request chains, JavaScript request chains provide centralized control because the requests are solely determined by the originating script without any intervention from external web servers. Centralized control is preferred in use cases where only a single central entity wants to control the navigation flow. For example, header bidding is a use case where centralized control is preferred because a single script at the client side conducts the bidding process.

3.3 Feature Extraction

Next, we use sequential context to extract features that distinguish advertising and tracking request chains from functional request chains. These features are designed based on our domain knowledge and expert intuition. We use sequential context to extract three types of features: (1) sequential, (2) response, and (3) request. Sequential features capture chain-level properties, whereas request and response features capture individual HTTP request-level properties. As a whole, these features provide complete sequential context. In total we extract 28 features from the request chains (Table 1). Below we describe some of the key features in each category. We analyze key features in § 4.4.

Sequential features capture communications that reveal the collaboration between domains. For example, a chain in which several domains redirect to each other may indicate that it is an advertising and tracking redirect chain where domains are trying to share information and identifiers. KHALEESI captures these properties by considering the *number of unique domains* and the *length of the chain* as features. The *number of unique domains* represents the unique number of domains contacted and the *length of the chain* capture the total number of requests in the chain.

Relying on sequence further allows KHALEESI to capitalize on the growing chains. Specifically KHALEESI uses the *probability of the previous prediction* and the *average probability of the previous predictions* features to summarize properties of the prior request and response pairs. The *probability of the previous prediction* feature represents the

Sequential Features	IG(%)
Average probability of the previous predictions	43.64%
Probability of the previous prediction	41.09%
Length of the chain	6.26%
Consecutive requests to the same domain	5.91%
Number of unique domains in the chain	3.97%
Response Features	
Content length	13.56%
P3P in response header	9.88%
Content sub-type (e.g. png)	8.82%
Status code	8.43%
ETag in response header	6.52%
Whether the response sets a cookie	4.55%
Content type (e.g. image)	4.44%
Number of response headers	3.97%
Request Features	
Third-party	35.10%
Length of the query string	30.34%
Ad/track. keywords in URL (e.g. pixel, track)	28.09%
Ad/track. keywords in URL surrounded by special char.	27.92%
Number of special characters in query string	27.80%
Length of the URL	22.26%
Subdomain check	12.54%
Accept type (e.g. image, script)	11.92%
Subdomain of the top-level domain check	9.07%
Top-level domain in query string	7.68%
Number of cookies in request	7.22%
UUID in URL	2.71%
Number of request headers	1.78%
Request method (GET or POST)	0.47%
Semicolon in query string	0.13%

Table 1: Features extracted from the request chains along with their information gain (IG).

classification probability in the previous classification and the the *average probability of the previous predictions* feature represents the average classification probability in prior classifications, indicating the likelihood of a request chain being advertising and tracking. To compute the *probability of the previous prediction* and *average probability of the previous predictions* features during the training phase, we mimic how these features would be computed during the real-time classification. Specifically, for each position in the chain, we train separate surrogate classifiers and classify (test) the redirects to compute the classification probability. To ensure that we do not test and train on the same data, we create 10 folds of data and test each fold by training on the remaining 9 folds. We limit the training of surrogate classifiers to 21 times because it is the maximum number of HTTP redirects allowed in Firefox and only 3.29% of the JavaScript request chains exceed that length. Sequential features are updated after receiving each response, and can only be computed after receiving the first response.

Response features capture properties that indicate the actions of the loaded content. For example, a response that loads a 1x1 image and sets a cookie is likely a tracking pixel. KHALEESI captures these properties by considering *content type*, *content length*, and *whether the response sets a cookie* as features. We also use *P3P* and *ETag* presence in response header as features, because P3P is a W3C standard used to specify cookie access policies [14] and prior research has shown that the ETag is exploited for tracking [24]. Similar to sequential features, response features are also computed after a response is received for a request.

Request features capture properties that can reveal the intent of the requester. For example, a request to a third-party domain that contains a large number of parameters in the query string may indicate that the third party is a tracker. KHALEESI captures these properties by considering the *length of the URL* and whether the request is from a *third party*. The *length of the URL* feature captures the total number of characters in a URL and the *third party* feature represents whether a request is first-party or third-party. We also try to capture the semantics of the content shared in the query string which may indicate that the request is advertising and tracking. Specifically, we use regular expressions to look for *UUID in URL* and *Ad or screen dimensions in URL*. The presence of *UUID in URL* indicates the leakage of a unique identifier and *Ad or screen dimensions in URL* indicates a request for an ad with certain dimensions. In contrast to the sequential and response features, the request features capture information that is available before a request is sent.

Network-layer vs. JavaScript-layer chains. For most of the features, the network-layer and JavaScript-layer chains have similar patterns. This allows us to use a single classifier to classify both types of chains. However, certain features are drastically different between network-layer and JavaScript-layer request chains, and that level of variance can confuse the classifier. For example, the *length of the chain* feature has smaller values for network-layer chains than JavaScript-layer chains. To mitigate these differences we include *chain type* as a meta-feature, which allows the classifier to avoid confusion when feature values vary significantly.

3.4 Classification

KHALEESI uses the random forest machine learning algorithm to classify request chains because it is better suited to avoid overfitting [29]. Random forest is an ensemble learning method that combines multiple decision trees and makes a prediction by taking the majority decision among trees. Each decision tree in random forest is trained on a random subset of the data and a random selection of the features (selected with replacement). Branches in a decision tree are constructed by splitting on features that provide the best separation for positive and negative samples. The random selection of data and features helps random forest avoid overfitting. We config-

Configuration	Network # Requests	Network # Chains	Network %AT	JavaScript # Requests	JavaScript # Chains	JavaScript %AT	% of Requests
Cookies allowed (homepage)	192,038	76,816	78.7%	253,582	44,747	65.6%	30.05%
Cookies allowed (interactive)	575,550	229,151	82.3%	1,320,733	145,135	61.1%	34.67%
Cookies blocked (interactive)	432,935	183,230	78.6%	1,195,188	120,879	61.8%	32.00%
Spoofed Safari (interactive)	384,404	166,018	70.5%	1,328,986	130,187	63.0%	32.78%
Webmail (no JavaScript)	217,102	76,938	78.1%	–	–	–	29.54%

Table 2: Total number of requests and request chains crawled for each configuration. AT refers to Advertising and Tracking. % of requests represents the percentage of network requests in each dataset that are part of request chains.

ure our random forest model to have 100 decision trees with randomly selected $int(\sqrt{N})$ features for each decision tree, where N is the total number of features.

KHALEESI's random forest model classifies individual requests in a request chain based on that request's sequential context. Since each request may potentially leak data to an ad and tracking server, KHALEESI tries to detect the ad and tracking request chains as early as possible. Note that in the case of network-layer request chains, no further requests are made from a chain once a request in the chain is detected and blocked as advertising and tracking. This is because each new request in the chain is initiated directly by the previous request. For JavaScript-layer request chains, all requests that are not classified as advertising and tracking are still sent by the script. This is because each request in the chain is initiated by an external script; blocking one request does not prevent the script from making another request to a non-blocked domain.

4 Evaluation

We evaluate KHALEESI along several dimensions.

4.1 Accuracy

Data Collection. We evaluate KHALEESI on crawl data collected using version 0.10.0 of OpenWPM [38] in August 2020 in the US. In each crawl, we visit the Alexa top-10K homepages. Interactive crawls additionally navigate to random internal pages by clicking on `iframes` and anchor tags.

We test the following configurations:

1. A non-interactive crawl using Firefox configured to *allow* all third-party cookies.
2. An interactive crawl using Firefox configured to *allow* all third-party cookies.
3. An interactive crawl using Firefox configured to *block* all third-party cookies.
4. An interactive crawl using Firefox configured to spoof some basic properties of Safari. Specifically, we configure Firefox to block all third-party cookies, override the `User-Agent` HTTP header, and override the JavaScript-accessible `useragent`, `vendor`, `appVersion`, and `platform` properties.
5. A dataset of emails collected by Englehardt et al. [37], which used an older OpenWPM configured to emulate an email client by disabling JavaScript and stripping the `Referer` header.

These configurations represent a sample of browsing conditions that users may experience and choose while browsing the web. For example, spoofed Safari with third-party cookie blocking is an emulation of default settings in the actual Safari web browser. Table 2 summarizes the chains extracted from all of the crawled dataset configurations.

Ground truth labeling. We compare KHALEESI's classifications against a ground truth of advertising and tracking request chains provided by filter lists. In line with prior literature [28, 45, 49], we compile the set of advertising and tracking request chains by using filter lists as ground truth. Specifically, we use EasyList [8] and EasyPrivacy [9], two of the most widely used filter lists, to label advertising and tracking request chains. We label a request chain as advertising and tracking if at least one of the requests in the chain matches the filter lists. Further, we address the imperfect nature of the ground truth provided by filter lists [22, 48, 65] by doing a post-hoc validation of KHALEESI's disagreements with the filter lists.

Classifier training and testing. We train separate random forest classifiers for each of the crawled dataset configurations and use 10-fold cross validation to test the datasets. Specifically, we divide request chains into 10 folds, where we use 9 folds for training and 1 fold for testing and repeat the process for 10 times, ensuring that we do not train and test on the same data.

Results. Table 3 presents KHALEESI's evaluation across all of the datasets. When trained on the same configuration where testing occurs, KHALEESI's machine learning models perform well. However, users won't always have access to a classifier trained on their exact browser configuration. For example, KHALEESI may be used alongside other privacy extensions, or in a browser that blocks all third-party cookies. Thus, we evaluate KHALEESI's performance in these unanticipated use cases. Specifically, we train a random forest classifier on the homepage dataset and test this classifier on the other datasets.

Configuration	Recall	Precision	Accuracy
Cookies allowed (home.)	98.87%	98.76%	98.63%
Cookies allowed (int.)	99.24%	99.13%	99.06%
Cookies blocked (int.)	99.10%	99.05%	98.97%
Spoofed Safari (int.)	99.06%	99.02%	98.99%
Webmail (no JS)	99.97%	99.97%	99.95%

Table 3: KHALEESI's accuracy in detection advertising and tracking request chains with a separate classifier for each of the crawled datasets.

Configuration	Recall	Precision	Accuracy
Cookies allowed (int.)	95.59%	94.78%	94.44%
Cookies blocked (int.)	94.90%	92.00%	92.60%
Spoofed Safari (int.)	94.26%	92.16%	92.77%
Webmail (no JS)	81.11%	97.33%	80.57%

Table 4: KHALEESI's accuracy in detection advertising and tracking request chains when trained on the homepage crawl configuration and tested on a different configuration.

We find that KHALEESI is less accurate when trained on one dataset and tested on another (Table 4). Specifically, the accuracy drops by around 4% to 6% for each of our web crawl configurations, and by 19.38% for our webmail configuration. This decline in accuracy is mainly due to differences in the feature distributions. We highlight a few of these differences below.

There are several key differences between the homepage configuration and the others. First, 302 redirects are over-represented in the homepage crawl. 42.91% of the advertising and tracking redirect requests in the homepage configuration use 302 redirects as compared to the interactive crawls where cookies were allowed (28.56%), where cookies were blocked (22.91%), and where we spoofed Safari (18.81%). Second, navigations to new domains are more common in the homepage crawl. Specifically, 51.32% of the advertising and tracking requests navigate to new domains in the homepage crawl as compared to the interactive crawls where cookies were allowed (47.49%), where cookies were blocked (35.59%), and where we spoofed Safari (35.41%). Third, non advertising and non tracking URLs are shorter in the homepage crawl. Specifically, the lengths of non advertising and non tracking URL in the interactive crawls are between around 16 to 25 characters longer than the homepage crawl. Fourth, cookie-related features are also different between the stock and cookie blocking configurations. Specifically, there are an average of 2.68 cookies per advertising and tracking request in the homepage crawl (which does not block third-party cookies) compared to 0.37 cookies for the configurations that block third-party cookies. In the latter case, cookies can only be set and retrieved in the first-party context.

The webmail configuration has several important differences compared to the web crawls. First, the chains in the webmail configuration are 50.90% smaller than the homepage crawl. Second, the content embedded in webmail is almost exclusively images (i.e., 99.99% of requests), whereas only 59.03% of requests in the homepage crawl are for images. Third, 301 redirects are much more common in the webmail dataset. Specifically, 301 redirects are initiated by 25.59% requests in the webmail configuration, and are almost nonexistent in other configurations (e.g., only 0.48% of requests in the homepage configuration initiate 301 redirects).

KHALEESI vs. filter lists. Since our ground truth is imperfect, we analyze disagreements between KHALEESI and filter lists using the following heuristics inspired from prior work [49]. We check whether the URL contains any of the usual advertising and tracking keywords, such as *rtb*, *tracking*, and *adsbygoogle*. We also check whether a small tracking pixel (i.e., less than 5x5 pixels) is loaded. We apply this heuristic to the disagreements that occurred while testing with the homepage configuration classifier (Table 4). We find that many of the KHALEESI's "mistakes" are in fact mistakes in the ground truth. Overall, KHALEESI's accuracy improves by 0.78% for the non-interactive homepage crawl, 1.26% for the interactive crawl that does not block cookies, 1.75% for the interactive crawl that blocks cookies, 1.53% for the interactive crawl where Safari is spoofed, and 17.08% for webmail crawl as compared to the original accuracy, computed with filter lists as reference in Table 4.

We note that EasyList and EasyPrivacy recently (June'21, i.e., approximately a year after the initial experiments) added 43 of the domains earlier detected as advertising/tracking by KHALEESI. Some examples include `sync.taboola.com`, `siteintercept.qualtrics.com`, `s0.2mdn.net`, and `log.popin.cc`. To our surprise, we notice that some of these domains are popular advertising services which should have been blocked by EasyList and EasyPrivacy. We further notice that the parent domains of some of these advertising services are already blocked by filter lists on some websites. For example, all resources from `taboola.com` are blocked on `independent.co.uk`, `scoopwhoop.com` and `technobuffalo.com`. We surmise that the filter lists take a conservative approach and avoid creating generic rules to block top level domains to mitigate breakage. Moreover, we find that the several of the KHALEESI's detected domains are still currently unblocked by EasyList and EasyPrivacy. Notable examples include `mediaiqdigital.com`, `quantumdex.io`, and `intentiq.com`. It is noteworthy that KHALEESI also misclassifies some functional requests as advertising and tracking. KHALEESI generally makes mistakes when functional request features are similar to advertising/tracking request features. For example, KHALEESI mistakenly detects a thumbnail image served from a CDN (`cdn.neighbourly.co.nz`) as ad/-tracking because of its similarity to tracking pixels.

Overall, as compared to filter lists, KHALEESI blocks a

total of 2,326 more tracking requests, corresponding to 1,634 request chains on 1,284 websites. The additional requests blocked by Khaleesi are to 1,259 distinct advertising/tracking domains, including taboola.com and intentiq.com, which are present on 26,096 chains and 6,602 websites.

4.2 Baseline Comparison

Next, we compare KHALEESI's classification accuracy with other machine learning approaches that are proposed in prior research and implemented in browsers. Specifically, we compare KHALEESI with an ML based approach by Bhagavatula et al. [28] (called BD+) that analyzes each request individually, Safari's Intelligent Tracking Prevention (ITP) [74] that analyzes request-response pairs, and WebGraph [63] a recent graph-based ML approach that analyzes structural properties of requests. Our goal is to highlight the marginal benefit of capitalizing on the full sequential context available in request chains. We also compare KHALEESI with an ML based approach by Papadopoulos et al. [58] (called CONRAD) that specializes in detecting cookie syncing events by analyzing individual requests. Our goal with this comparison is to determine whether a specialized cookie syncing detection approach generalizes to detect all advertising and tracking request chains. Table 5 reports the accuracy of BD+, ITP, WebGraph, and CONRAD in detecting advertising and tracking request chains.

	Configuration	Recall	Precision	Accuracy
BD+ [28]	**Cookies allowed (home.)**	93.68%	89.58%	89.72%
	Cookies allowed (int.)	87.24%	76.3%	77.09%
	Cookies blocked (int.)	88.84%	77.54%	79.54%
	Spoofed Safari (int.)	88.76%	73.56%	77.34%
	Webmail (no JS)	8.52%	95.64%	15.46%
ITP [74]	**Cookies allowed (home.)**	97.43%	58.44%	58.56%
	Cookies allowed (int.)	98.00%	58.58%	59.02%
	Cookies blocked (int.)	97.68%	56.76%	57.44%
	Spoofed Safari (int.)	97.82%	54.05%	55.18%
	Webmail (no JS)	100%	86.40%	86.40%
WebGraph [63]	**Cookies allowed (home.)**	42.31%	74.24%	68.32%
	Cookies allowed (int.)	12.97%	49.13%	42.48%
	Cookies blocked (int.)	16.75%	65.96%	42.93%
	Spoofed Safari (int.)	16.04%	59.46%	44.04%
	Webmail (no JS)	–	–	–
CONRAD [58]	**Cookies allowed (home.)**	95.78%	95.75%	95.12%
	Cookies allowed (int.)	91.93%	92.74%	91.23%
	Cookies blocked (int.)	78.66%	82.14%	78.61%
	Spoofed Safari (int.)	77.77%	77.74%	76.63%
	Webmail (no JS)	96.24%	97.51%	94.36%

Table 5: BD+, ITP, WebGraph, and CONRAD's classification accuracy in detecting advertising and tracking request chains across different datasets.

Comparison with BD+ [28]. We use the best performing k-Nearest Neighbors (kNN) ML classifier called BD+ by Bhagavatula et al. [28] (see § 2.2 for details). For a fair comparison, we train BD+ on the same set of top-10K websites as used to train KHALEESI. Further, we consider the *offline* setting for BD+, where features are computed at the end of page load, as opposed to the *online* setting used by KHALEESI, where features are computed on the fly as the page is being loaded.[3] We again use 10-fold cross validation to compute the accuracy of the classifier. As compared to KHALEESI, BD+ achieves 5.19% less recall, 9.18% less precision, and 8.91% less accuracy. Table 5 shows that the trend holds for the interactive, cookie blocking, spoofed Safari, and webmail configurations. The lack of sequential context makes it challenging for such request-based approaches to accurately detect advertising and tracking chains. For example, Listing 1 provides an advertising and tracking request chain that appears to be cookie syncing, but is missed by BD+ because it does not use the response and sequence information in its detection. KHALEESI is able to successfully capture the appearance of cookie syncing in the request chain and correctly detects it as advertising and tracking.

```
/* 1st request and response pair */
URL: sync.republer.com/match?dsp=sape&qset=1
Cookie: 88661a5f-2d71-4780-95af-e9d2edd1eebb
Location: sync.bumlam.com/?src=rp1&uid=
          88661a5f-2d71-4780-95af-e9d2edd1eebb

/* 2nd request and response pair */
URL: sync.bumlam.com/?src=rp1&uid=
      88661a5f-2d71-4780-95af-e9d2edd1eebb
Location: sync3.adsniper.ru/?src=ssl&s_data=
          CAEQABjikM32BV....ZDFlZWJi
........
/* 4th request and response pair */
URL: sync3.adsniper.ru/?src=ss1&s_data=
     CAIQARjikM32BVIEioaQK2Ik....gkNw**
Location: sync.bumlam.com/?src=rp1&s_data=
          CAIQARjikM32BVIEioaQK2Ik....gkNw**
........
```

Listing 1: A request chain that appears to be cookie syncing missed by the request based approach used by BD+ [28]. The shared identifiers are highlighted in red.

Comparison with ITP [74]. We use ITP's SVM classifier that is shipped in the production Safari web browser [74] (see § 2.2 for details).[4] We test ITP on the request chains extracted from top-10K websites. We find that ITP suffers from a lot of false positives – with only 58.44% precision. ITP achieves 1.44% less recall, 40.32% less precision, and 40.07% less overall accuracy as compared to KHALEESI. Table 5 shows that the trends hold for interactive stock, cookie blocking, spoofed Safari, and webmail configurations. Note that Safari uses ITP's detections only to purge storage and does not

[3] Note that the offline setting puts BD+ in an advantageous position because post-hoc feature computation uses additional information that is not otherwise available in the online setting.

[4] All versions of the ITP use the same classifier released with the initial version 1.0.

block ads and trackers. Thus, ITP does not cause excessive website breakage as it would if it was used to outright block requests. The lack of comprehensive sequential context makes it challenging for such request-response based approaches to accurately detect advertising and tracking chains. For example, Listing 2 provides an example of a request chain that appears to be used for authentication, but is incorrectly detected by ITP as advertising and tracking due to over-generalization on partial sequential context. Note that the KHALEESI is able to successfully capture the non-advertising and non-tracking signal, i.e. the absence of shared identifiers, in the request chain and detects it as a functional.

```
/* 1st request and response pair */
URL: adobe.sharepoint.com/_forms/default.aspx?
     ReturnUrl=/_layouts/15/Authenticate
     .aspx?Source%2F&Source=cookie
Cookie: RpsContextCookie=U291cmNlPSUyRg==
Location: login.windows.net:443/fa7b1b5a-7b34-
          4387-94ae-d2c178decee1/oauth2/authorize
          ?client_id=00000003-0000-0ff1-ce00-0000
          ...
          &redirect_uri=adobe.sharepoint.com....
........
```

Listing 2: A request chain that appears to be authentication related. It is incorrectly classified as advertising and tracking by ITP.

Comparison with WebGraph [63]. We also compare KHALEESI with WebGraph [63] (see § 2.2 for a high-level comparison of WebGraph and KHALEESI). For a fair comparison, we train WebGraph's random forest classifier on the same set of top-10K websites as used to train KHALEESI. We use 10-fold cross validation to compute the accuracy of the classifier. We find that WebGraph suffers from a lot of false negatives – with only 42.31% recall. As compared to KHALEESI, WebGraph achieves 56.56% less recall, 24.52% less precision, and 30.31% less accuracy. Table 5 shows that the trend holds for the interactive, cookie blocking, and spoofed Safari configurations.[5]

WebGraph performs worse because it is designed to capture a wide variety of advertising and tracking patterns that overshadow the patterns specifically implemented through request chains. For example, `Number of requests sent by node` and `Average degree connectivity of node` are two of the most important features of WebGraph; however, they disproportionately impact the detection of HTTP request chains. Specifically, in WebGraph's representation, nodes corresponding to HTTP request chains have lower values for these features because each node is only connected to the prior and the next node in a sequence, forming a chain like structure. Whereas, nodes corresponding to JavaScript request chains have higher values for these features because all originating requests from a script are connected to one script node, forming a start like structure. These differences are reflected in its classification results, WebGraph achieves an accuracy of 85.40% (on par with the reported accuracy in the paper [63]) for JavaScript request chains but only 22.18% for HTTP request chains.

Comparison with CONRAD [58]. We implement the decision tree ML classifier called CONRAD by Papadopoulos et al. [58] (see § 2.2 for details). For a fair comparison, we train the decision tree classifier on the same set of top-10K websites as used to train KHALEESI. However, we do not consider the *Browser* and *TypeOfEntity* features used in the original classifier. We ignore the *Browser* feature because we only rely on one browser to collect the data. We ignore the *TypeOfEntity* feature because it relies on labels from the disconnect tracking prevention list [7], which is essentially the same as using ground truth as a feature. We use 10-fold cross validation to compute the accuracy of the classifier.

As compared to KHALEESI, CONRAD achieves 3.09% less recall, 3.01% less precision, and 3.51% less accuracy. Table 5 shows that the trend holds for the interactive, cookie blocking, spoofed Safari, and webmail configurations. For webmail configuration, CONRAD performs better as compared to KHALEESI. CONRAD's high accuracy comes from its reliance on domain and URL parameter names as features. Specifically, the top performing features include whether the requests are going to popular advertisers and trackers, such as `doubleclick.net` and `fbcdn.net` and whether the URL embeds parameters commonly used by advertisers and trackers, such as `id` and `uid`. However, relying on hard coded values makes the classifier overfit and susceptible to evasion attacks (see § 4.3.2 for details). For example, Listing 3 provides an advertising and tracking request chain from `luxup.ru`, a less known advertising service, which appears to engage in cookie syncing but it is missed by CONRAD. KHALEESI is able to successfully capture the appearance of cookie syncing signal in the request chain and correctly detects it as advertising and tracking.

```
/* 1st request and response pair */
URL: luxup.ru/tr/22710/&r=&t=1590939122591
Location: adlmerge.com/md/?mdback=luxup.ru/tr/22710/
          &r=&t=1590939122591

/* 2nd request and response pair */
URL: adlmerge.com/md/?mdback=luxup.ru/tr/22710/
     &r=&t=1590939122591
Set-Cookie: __LXGUID=6833031521866537697
Location: luxup.ru/tr/22710/&r=&t=1590939122591
          &md=6833031521866537697
........
```

Listing 3: A request chain from a lesser known advertiser that appears to be cookie syncing. This chain is missed by CONRAD [58] because of CONRAD's reliance on popular domain and parameter names as features. The shared identifiers are highlighted in red.

Overall, the comparison with prior approaches, which do

[5] We were only able to test 150 randomly selected websites due to WebGraph's significant performance overhead. WebGraph's open-sourced code took several hours and more than 300 GB of memory on 50 processor cores to process these websites, for each configuration. We could not evaluate Webmail crawl because its schema is not compatible with WebGraph's code.

not or only partially use sequential context, demonstrates the importance of leveraging the full sequential context for accurate detection of advertising and tracking request chains.

4.3 Robustness Analysis

Next, we evaluate KHALEESI's classification accuracy over time and its robustness against evasion.

4.3.1 Classification accuracy over time

Machine learning models are prone to lose their effectiveness over time. Specifically, the underlying data distributions on which the model is trained change over time, in unforeseen ways, making predictions less accurate as the time passes [66]. KHALEESI tries to slow down the degradation by learning on fundamental properties of request chains, i.e., their sequential context, and by avoiding frequently changing variables such as domain names. Learning on sequential context makes KHALEESI's machine learning model generalizable, which can adapt to new, previously unseen, request chains because they will have the same distribution as the request chains that were used to train the initial model. To evaluate KHALEESI's accuracy over time, we test it on a new data set crawled in April 2021, i.e., 8 months after the initial crawl on which the model was trained. The results show that KHALEESI achieves an accuracy of 94.07% with a recall of 97.26% and a precision of 93.11% in detecting advertising and tracking request chains. We note that the accuracy only drops by 4.56%, recall by 1.61%, and precision by 5.65% due to the drift in the request chain distributions over time.

Over 8 months, we note few changes in distribution of advertising and tracking requests. The most significant changes are in number of advertising and tracking keywords in requests and the number of `XMLHttpRequests`, which increase by 6.45% and 6.37%, respectively. We also note that the advertising and tracking requests contains 0.63% less cookies on average. The increase in number of advertising and tracking keywords and `XMLHttpRequests`, coupled with decrease in cookies, perhaps indicates a increasing trend towards usage of alternative tracking mechanisms, e.g., fingerprinting, triggered by third party cookie blocking by main-stream browsers [5, 68]. We conclude that, to keep the accuracy consistent over time, the model must be fine-tuned continuously, which might include adding features to capture new patterns or removing features that capture inconsistent patterns.

4.3.2 Robustness against evasions

Due to the countermeasures against cross-site tracking by main-stream browsers, such as Safari [68] and Firefox [6], trackers are increasingly relying on evasive tactics to circumvent the deployed countermeasures. Prior research has shown that the URL-based ad and tracker detection approaches are vulnerable to domain rotation and URL path randomization [22, 67]. Since KHALEESI also relies on request properties, in addition to response and sequential properties, we evaluate its robustness against real world domain rotation and URL randomization attacks and as well as hypothetical response and sequence manipulation attacks. Specifically, we evaluate KHALEESI against real-world domain rotation [35, 77] and CNAME cloaking [60] attacks and hypothetical query string manipulation, response header removal, and chain shortening attacks. We describe these evasion attacks below:

1. **Domain rotation** involves randomizing the label that is followed by the effective Top Level Domain (eTLD), e.g., example.com, example is the label and com is the eTLD.
2. **CNAME cloaking** involves switching the third party's eTLD+1 to the first party's eTLD+1, along with the addition of the third party's label as a sub domain.
3. **Query string manipulation** involves the randomization, removal, and MD5 hashing of parameter name-value pairs.
4. **Response header removal** involves the removal of the ETag and P3P headers from the response.
5. **Chain shortening** involves the random truncation of request chains by 50–70%.

To launch these attacks, we randomly select 100 adversaries from the top 20% of the most common advertising and tracking third parties, which account for 16.35% of all the chains in our data set, i.e., the cookies allowed homepage configuration. Our selection includes prominent advertisers and trackers, such as Criteo and PubMatic. Further, to provide a relative perspective, we compare KHALEESI robustness with CONRAD [58] and BD+ [28], i.e., two of the best performing approaches from § 4.2.[6]

Table 6 presents the adversaries' success rate against KHALEESI, CONRAD, and, BD+ with different evasion attacks. Success rate is defined as the percentage of classification switches from advertising and tracking to non advertising and non tracking. In case of KHALEESI, we note that the adversaries are able to evade KHALEESI between 1.31–13.85% of the time. Whereas, in case of CONRAD and BD+, the adversaries are able to evade the classifier between 8.57–9.71% and 10.41–21.51% of the time, respectively. It is noteworthy that KHALEESI is much more robust as compared to CONRAD and BD+ for all of the evasion attacks, except for CNAME cloaking as compared to CONRAD, where the adversaries are able to switch 4.35% more instances from advertising and tracking to non advertising and non tracking.

We note that KHALEESI is not solely reliant on any of the request, response, or sequential features and instead its robustness, against adversarial evasion comes from complementary

[6] Response header removal and chain shortening attacks are not applicable to CONRAD and BD+ because they do not use response headers, other than the status code, and sequential context of request chains.

Evasion attack	KHALEESI	CONRAD [58]	BD+ [28]
Domain rotation	1.75%	8.57%	10.41%
CNAME cloaking	13.85%	9.50%	21.51%
QS manipulation	3.94%	9.71%	11.67%
Response removal	1.67%	–	–
Chains shortening	1.31%	–	–

Table 6: Evasion rate against KHALEESI, CONRAD, & BD+.

accuracy gained from request, response, and sequential features. Specifically, we note that request, response, and sequential features alone provide an accuracy of 98.69%, 91.94%, and 93.11%, respectively. In contrast, CONRAD has a strong reliance on domain and URL parameter names as features. We note that domain and URL parameter names alone provide an accuracy of 93.51%, while their exclusion provides an accuracy of only 82.32%.

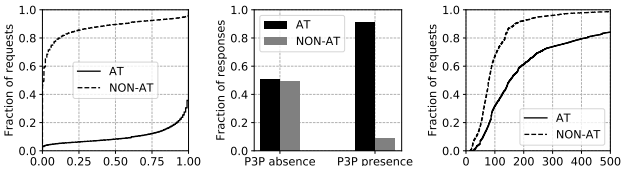
Overall, we acknowledge that KHALEESI is not fool-proof against an adversary that can manipulate most of its features simultaneously (see § 6 for more details). We conclude that KHALEESI is fairly robust against adversarial evasions.

4.4 Feature Analysis

To shed light into KHALEESI's inner workings, we analyze a few of the most important features, with high information gain [59] across sequential, response, and request feature categories. These features are the most helpful for KHALEESI in distinguishing advertising and tracking requests from non-advertising and non-tracking requests in the request chains.

Probability of previous prediction. The probability that the previous prediction was advertising and tracking captures the information capitalized by KHALEESI as the chains grow longer in length. Intuitively, using the prior probability helps KHALEESI make a better determination. For example, if KHALEESI is moderately confident that the request will load an ad or a tracker, it can reaffirm that by leveraging the confidence from its prior prediction. Figure 2a plots the distribution of the probability of previous prediction. We note that 85.47% of the non-advertising and non-tracking requests have less than 25% probability of being an ad or a tracker. Whereas, 93.74% of the advertising and tracking requests have greater than 25% probability in being an ad or a tracker.

P3P in response header. P3P (Platform for Privacy Preferences) was a W3C standard that allowed websites to convey their privacy policies in a standardized format [14]. Internet Explorer by default rejected third-party cookies in the first-party context unless the cookie usage was specified by a P3P header. P3P was implemented by Internet Explorer, Edge, and Mozilla [1, 53] as well as supported by thousands of domains at one point [36]. Currently, no modern browser supports P3P; only some older versions of Internet Explorer have support P3P. Despite this, we find that many third parties still use P3P as the HTTP response header to specify their cookie usage



(a) Probability of previous prediction. (b) P3P is response header. (c) Length of URL.

Figure 2: Distribution of top sequential, response, and request features. AT refers to advertising and tracking requests and NON-AT refers to non advertising and non tracking requests.

policy. Figure 2b shows the prevalence of P3P headers in request chains. It can be seen from the figure that 90.99% of advertising and tracking responses have a P3P header as compared to only 9.01% of the non-advertising and non-tracking responses.

URL length. Advertisers and trackers collect sensitive information from users' devices and share it with others as URL query strings. This so-called *link decoration* is increasingly being used to bypass privacy protections against third party cookies [71–73]. As can be seen in Figure 2c, this information sharing leads to longer URLs for advertising and tracking requests. 38.78% of advertising and tracking URLs are longer than 200 characters, while only 7.79% of the non-advertising and non-tracking URLs are longer than 200 characters.

4.5 Breakage Analysis

Content blocking tools are prone to website breakage because of incorrect blockage of functional resources or their dependence on advertising and tracking resources. To be usable in the real-world, KHALEESI' breakage must be on par with existing content blocking tools such as Adblock Plus [2]. Therefore, we evaluate and compare KHALEESI's breakage with Adblock Plus[7] on popular websites. We manually quantify website breakage on a random sample of 100 websites from top-1K list by conducting common browsing actions such as reading articles, adding products to carts, and opening sub pages of a website. We open each website using stock Firefox as a control and with Adblock Plus and KHALEESI as treatments. We open three browser instances side by side, analyze the website functionality on each of them, and assign one of the following labels:

1. **Major:** The core functionality of the website is broken and the user cannot fulfill their objective. For example, the user is unable to submit a form or book a flight.

2. **Minor:** The non-core functionality of the website is broken but the user is able to fulfill their objective. For example, some icons or images are missing on the webpage.

[7]We configure Adblock plus with the same filter lists, i.e., EasyList [8] and EasyPrivacy [9], on which KHALEESI was trained.

3. **None:** The experience is similar across control and treatments.

Tool	None		Minor		Major	
	#	%	#	%	#	%
KHALEESI	90.0	93.8	3.5	3.6	2.5	2.6
Adblock Plus	90.5	94.3	3.0	3.1	2.5	2.6

Table 7: Average breakage assessment of 2 reviewers.

To mitigate potential bias and subjectivity in manual breakage analysis and avoid any inconsistencies due to dynamic content, we asked two reviewers to independently analyze the test websites at the same time. The two reviewers achieved a high 94.79% agreement in their breakage determinations. Table 7 presents the averaged breakage results[8]. We note that, KHALEESI and Adblock Plus cause no breakage on 93.8% and 94.3% of the tested sites, respectively. Both cause major breakage on the same 2.6% of the sites. Only in one instance, KHALEESI misclassified and blocked a request to `intercomcdn.com`, leading to a minor breakage (missing chat button) on the website.

Authentication & Purchase Scenarios. Since request chains are widely used in authentication and purchase flows, we manually evaluate KHALEESI's breakage in these scenarios. We analyze both same-site and cross-site scenarios. To test same-site authentication flows, we attempt to log into Facebook, Amazon, and Twitter. Similarly, to evaluate cross-site authentication flows, we attempt to log into NYTimes and EBay using 3 popular federated identity providers (Google, Facebook, Apple). To test purchase flows, we log into Amazon, search for a product, add it to the cart, and make an actual purchase.

We are able to successfully complete authentication and purchase flows when using KHALEESI. Specifically, KHALEESI only blocked advertising and tracking resources on the tested websites and did not break any login and purchase flows. Note that authentication and purchase flows cannot be automatically tested at scale. Specifically, testing large-scale authentication scenarios require the availability of login/payment credentials for a large number of online services. Further, automated login and purchase flows often trigger CAPTCHAs that cannot be trivially bypassed.

We conclude that KHALEESI's breakage is on par with Adblock Plus for regular browsing sessions and it does not break any authentication and purchase flows.

4.6 Performance

Ad and tracker blocking tools generally improve the page load time by blocking content; however, at the same time, they also incur performance overhead. In traditional filter list based ad and tracker blocking tools, overheads are incurred in matching requests against filter lists and removing content from DOM. In ML based ad and tracker blockers, there are overheads to extract features and use the model to classify requests. As we show next, KHALEESI incurs overheads but improves the performance, significantly more, as compared to other ad and tracker blocking tools.

KHALEESI's performant implementation. We implement KHALEESI as a browser extension by extending the open source implementation of Adblock Plus [3]. Since KHALEESI is designed to only block advertising and tracking request chains, we complement KHALEESI with EasyList [8] and EasyPrivacy [9] for blocking non-request-chain based ads and trackers. Specifically, request chains are routed through KHALEESI and non-request-chain based requests are routed through filter lists. We create a lightweight in-memory representation of request chains and extract features from requests in a streaming fashion, before they leave the browser, and response headers, before even the whole response is received and parsed by the browser. Extracting features from response headers, before they are parsed, allows us to save time by avoiding the need to wait for their parsing and rendering. Moreover, we flush the in-memory representation when the browser navigates to another page. For network layer request chains, however, we flush the in-memory representation as soon as we block them or they return a non-redirect response.

KHALEESI's performance comparison. We quantify KHALEESI's overheads and performance improvements by comparing it against stock Firefox and Firefox with Adblock Plus.[9] We use a standard desktop machine with Windows 10 OS, 16GB of RAM, and an i7 processor. We also simulate consistent network conditions by setting the bandwidth to 15 Mbps with a latency of 100 ms. We measure the page load time[10], averaged over five runs for each website, on Alexa top-500 websites. Figure 3 shows KHALEESI's page load time as compared to stock Firefox and Adblock Plus and as well as the Adblock Plus's performance against stock Firefox configuration. As compared to stock Firefox, we find that KHALEESI improves the page load time on 91.26% of the websites. KHALEESI's improvements over stock Firefox are due to blocking advertising and tracking content, which results in less network requests and less rendering. As compared to Adblock Plus, we find that the KHALEESI improves the page load time on 59.82% of the websites. The performance gains over Adblock Plus highlight KHALEESI's added benefit of blocking advertising and tracking request chains. Thus, we conclude that KHALEESI's improved accuracy outweighs its minor feature extraction and classification overheads, making it suitable for a performant online deployment.

[8] Out of the 100 websites, 5 websites failed to load in both control and treatments, so we exclude them from our analysis.

[9] We disable Firefox's Enhanced Tracking Protection (ETP) and configure Adblock Plus with EasyList [8] and EasyPrivacy [9].

[10] Page load time is measured as a difference between `navigationStart` and `loadEventEnd` events of the `Performance` API.

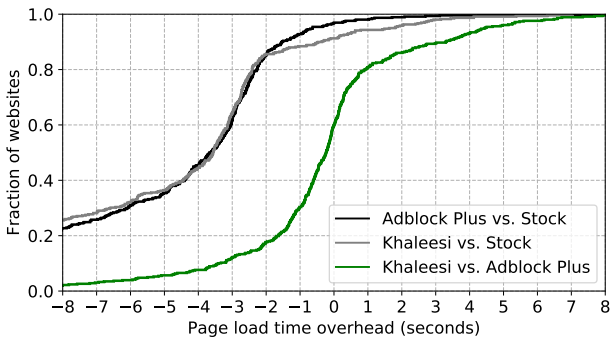


Figure 3: KHALEESI's overhead in terms of page load time.

5 Discussion

In this section, we analyze KHALEESI's findings to shed light into the online information sharing ecosystem through graph analysis and case studies emerging use cases of request chains.

5.1 Request Chain Graph

Next, we build the request chain graph to understand bilateral information sharing relationships between different entities. To this end, we construct a graph such that the nodes represent domains and the edges represent consecutive domains in request chains. Note that edges are directed and weighted: the source/destination of the directed edge represents the source/destination in the redirect and the edge weight represents the frequency of their co-occurrence.[11] Also, the size of a node is proportional to its degree and the color represents degree ratio—ratio of in-degree and out-degree—red/blue represents higher/lower in-degree as compared to out-degree. Note that for the node colors, darker shades of either blue or red mean higher asymmetry and lighter shades mean higher symmetry.

Figure 4(a) plots the largest strongly connected component (LSCC) of the request chain graph constructed using the first crawl configuration (cookies allowed (homepage)). The nodes with higher/lower degree ratios (in shades of red/blue) are destinations/sources of redirects. We note that the destinations of most of the redirects are typically well-known advertising and tracking domains such as doubleclick.com and pubmatic.com. In contrast, their sources typically include general-purposed cloud providers such as cloudfront.net as well as tag management services such as googletagmanager.com. This indicates that advertising and tracking domains are generally the recipients of tracking information from other domains.

Other SCCs (not shown in Figure 4) are much smaller in size (most of them with around 10 nodes or less). We note that most of the nodes in these smaller SCCs are mostly cliques of related domains (e.g., amazon.fr → amazon.com, towardsdatascience.com → medium.com) and sometimes country-specific domains (e.g., admaster.com.cn, reachmax.cn, yoyi.com.cn). Also not shown in Figure 4 are self-loops, which mostly represent sub-domains which appear to be trying to sync cookies with their parent domains (e.g., aliexpress.com, admicro.vn).

Figures 4(b) and (c) plot the LSCC of the request chain graph after blocking using filter lists and KHALEESI, respectively. It is evident that the request chain graph becomes significantly more sparse when advertising and tracking requests are blocked by either filter lists or KHALEESI. More specifically, we note that both node sizes and number of edges significantly reduce for filter lists or KHALEESI in Figures 4(b) and (c) as compared to no blocking in Figure 4(a). In fact, some of the largest nodes in Figure 4(a) (e.g., rubiconproject.com) completely disappear in Figures 4(b) and (c). Comparing Figures 4(b) and (c), we note that even more nodes (e.g., taboola.com) disappear and edges' weights decrease going from filter lists to KHALEESI. In Figure 4(c), we observe mostly smaller sized nodes in blue colors represent popular cloud providers such as amazonaws.com and cloudfront.net.

	No Blocking	Filter lists	KHALEESI
Average Clustering	0.03283	0.00214	0.00155
# of SCC	10221	9852	9732
# of Nodes in LSCC	865	83	52
Top Degree in LSCC	603	88	77

Table 8: Statistics of graphs in three configurations.

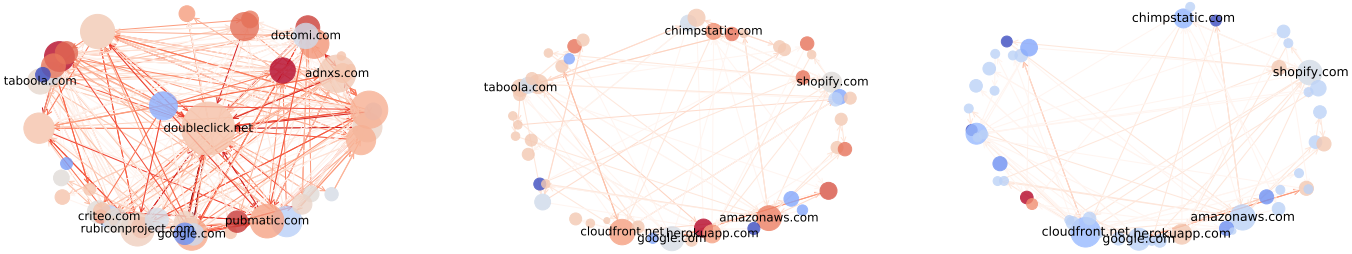
To further analyze these differences quantitatively, Table 8 lists a few key graph connectivity metrics based on clustering and connected components. These metrics quantitatively demonstrate that redirect graphs progressively become more fragmented as we go from no blocking to blocking with filter lists and KHALEESI. For example, the size of the LSCC decreases by 10× due to filter lists, and further decreases by 37% due to KHALEESI. We note a similar trend for other reported metrics in Table 8.

In sum, our analysis shows that request chains enable widespread information sharing between third-party advertising and tracking domains. Both filter lists and KHALEESI help mitigate this by degrading the graph connectivity. We also conclude that KHALEESI is more effective than just filter lists in mitigating the information sharing by advertising and tracking request chains.

5.2 Analysis of Request Chains

We analyze use cases of request chains for advertising and tracking.

[11] For example, a request chain example1.com → sub.example2.com → example3.com would be represented in a graph with 3 nodes (example1.com, example2.com, example3.com) and 2 edges (example1.com → example2.com, example2.com → example3.com).



(a) No Blocking (b) Filter Lists (c) Khaleesi

Figure 4: Request chain graph of redirects between top-50 most popular domains.

Bounce tracking occurs in cookies blocked crawl configurations. Bounce tracking is fairly broadly defined [69], and there are a number of different ways it can be achieved. For example, a bounce tracker may directly redirect a tab to the tracker's webpage or it may rewrite links on the webpage, such that when links are clicked, the webpage is redirected to the tracker's webpage. Further, the redirect may end on the same or a different webpage. However, the end objective of bounce tracking still remains the same, i.e. it provides cookie access to domains that are typically only loaded in the third-party context (where cookie access may be blocked). To measure bounce tracking, we create a fairly precise heuristic. We consider third parties to be bounce trackers if they:

1. Start from the top-level main frame
2. Navigate to one or more third party webpages,
3. The navigated third party webpages set cookies, and
4. The third party webpage navigates back to the first party webpage.

Since bounce tracking can only be conducted by network level requests chains or JavaScript request chains originated with `window.location`, we exclude other JavaScript request chains for measuring bounce tracking in the first heuristic. Further, we only investigate the cookies blocked configurations, i.e. cookie blocked interactive and spoofed Safari interactive, for bounce tracking because bounce tracking is a workaround to bypass third-party cookie blocking.

Overall, we find 14 domains, across both cookie blocked interactive and spoofed Safari interactive configurations, that are classified as bounce tracking. Table 9, shows the top domain classified as bounce trackers. We find that two of the domains classified as bounce trackers, i.e. `adform.net` and `flashtalking.com`, provide purpose-built workarounds to third party cookie blocking [4, 11]. For instance, Adform claims to use a "universal ID based on a first party cookies" by redirecting the users from their visited website to an intermediate website and redirecting them back to the visited website. The two most popular domains classified as bounce trackers, i.e. `googleadservices.com` and `adsrvr.org`, are services that provide generic ad-tech solutions. `queue-it.net` is a non-advertising and non-tracking service that provides queuing services to control website traffic.

Top Domains	Spoofed Safari # of websites	Cookies blocked # of websites
googleadservices.com	3073	1773
adsrvr.org	1377	1538
adform.net	322	-
flashtalking.com	141	-
queue-it.net	9	-
ojrq.net	-	19
bngpt.com (NSFW)	2	5

Table 9: Top domains classified as bounce tracking across the cookie blocked interactive and spoofed Safari interactive configurations.

Request chains are widely used in cookie syncing. Table 10 shows the prevalence of cookie syncing detected using CONRAD [58] across different crawl configurations.[12] We find that more than half of the request chains in most configurations participate in cookie syncing. In fact, more than half of the domains in request chains participate in cookie syncing across all of the configurations with more than two thirds of domains participating in cookie syncing in the webmail configuration. We notice that domains in the interactive, cookies allowed crawl send and receive the most identifiers on average. Note that `doubleclick.net` is the most popular domain detected to participate in cookie syncing across all web crawl configurations, and syncs to as many as 1,100 different domains in the interactive crawl that spoofed Safari. Surprisingly, we note that cookie syncing is more common in configurations where we block cookies, i.e. the interactive cookies blocked crawl and the interactive spoofed Safari crawl. We find that a vast majority of cookie syncing cases involve exchange of potential identifiers in query strings and non-standard HTTP request headers. Not shown in the table, but 91.13% of the cases in the spoofed Safari interactive crawl and 91.25% of the cases in the interactive cookies blocked

[12] We tweak the detection method to consider identifiers from multiple locations in the request/response header and also consider their hashed versions (see JavaScript request linking in § 3.2 for more details).

Configuration	Request Chains (%)	Domains (%)	Send (average)	Receive (average)	Top Domain	Send Domain (count)	Receive Domain (count)
Cookies allowed (home.)	54.37%	50.87%	6.49	5.86	doubleclick.net	322	336
Cookies allowed (int.	54.54%	52.82%	7.39	7.00	doubleclick.net	828	872
Cookies blocked (int.)	28.49%	55.40%	6.80	6.81	doubleclick.net	1,060	1,012
Spoofed Safari (int.)	27.82%	57.08%	6.54	6.67	doubleclick.net	1,100	1,071
Webmail (no JS)	70.59%	78.72%	2.24	4.00	liadm.com	19	84

Table 10: Prevalence of cookie syncing in request chains across all of the crawl configurations. Send/Receive Average represents the average number of domains per request chain to/from which cookies are shared/received. Top Domain refers to the domain that is most frequently detected to cookie sync. Send/Receive Domain Count represent the total number of distinct domains to which top domains share/receive cookies to/from.

crawl represent such cases. We also find a high prevalence of identifiers that have been Base64 encoded, MD5 hashed, and SHA-1 hashed. Not shown in the table, 2.77% of domains in the homepage cookies allowed crawl, 3.74% domains in interactive cookies allowed crawl, 3.73% of domains in the interactive cookies blocked, 4.65% of domains in the interactive spoofed Safari crawl, and 0.94% of domains in the webmail crawl use encoded and hashed identifiers while syncing cookies.

6 Concluding Remarks

In this paper we proposed KHALEESI, a machine learning based approach that capitalizes on sequential context to detect advertising and tracking request chains. We conclude the paper by discussing some limitations and future work.

Data collection: Our data collection has some limitations that pose internal and external threats to the validity of our findings. For crawling, we used OpenWPM instrumented browser, which is more complete than primitive crawlers such as PhantomJS [25] but is still detectable [44]. Our web crawler used a vantage point at an academic institute and the results may vary across different networks (e.g., residential) and geographic locations.

Feature robustness: KHALEESI's robustness against feature manipulation is not fool-proof. A motivated adversary can, in theory, dial-up these feature manipulations (§4.3.2) to evade detection by KHALEESI. However, that would require significant changes to the infrastructure, techniques, and working model of the current advertising and tracking ecosystem. For example, changing domain names and query string parameters requires non-trivial coordination between front-end and back-end across several advertisers and trackers [19, 20]. P3P's continued presence in many advertising and tracking requests, despite its discontinuation for more than 6 years, also highlights the slow change of affairs in the advertising and tracking ecosystem [1].

Adversarial attacks: KHALEESI's random forest ensemble classifier may also be susceptible to adversarial attacks on general machine learning classifiers (e.g., FGSM [42]). If these adversarial attacks are realized against KHALEESI, countermeasures such as adversarial training can be used to harden random forest ensemble classifier [32, 33, 50].

Model accuracy over time: The web is continuously changing and so is the online advertising and tracking ecosystem. As the advertising and tracking techniques evolve, this may change the underlying data distributions on which KHALEESI is trained. While we showed that KHALEESI's accuracy degraded only minimally (§4.3.1), it might require periodic retraining to ensure high accuracy. Future work can look into automatically updating KHALEESI's ML classifier using online learning techniques based on user feedback.

Integration with other tools: KHALEESI focuses only on request chains and other standalone network requests are considered outside the scope. It is meant to be complemented with filter lists to handle standalone network requests. Future work can look into integrating KHALEESI into existing ML-based ad and tracker blocking tools (e.g., [49, 64]).

Code and data release: KHALEESI's browser extension, code, and data is available at https://uiowa-irl.github.io/Khaleesi/.

Acknowledgment

We thank anonymous reviewers and our shepherd Blase Ur for their constructive feedback. This work is supported in part by the National Science Foundation under grant numbers 2051592, 2102347, 2103038, and 2103439.

References

[1] A Quick Look at P3P. https://blogs.msdn.microsoft.com/ieinternals/2013/09/17/a-quick-look-at-p3p/.

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

[3] Adblock Plus Github Repo. http://github.com/adblockplus/adblockplus.

[4] Adform CookieApocalypse. https://site.adform.com/media/86216/id-management-final.pdf.

[5] Building a more private web: A path towards making third party cookies obsolete. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html.

[6] Disable third-party cookies in Firefox to stop some types of tracking by advertisers. https://support.mozilla.org/en-US/kb/disable-third-party-cookies.

[7] Disconnect tracking protection. https://disconnect.me/trackerprotection.

[8] EasyList. http://easylist.to/easylist/easylist.txt.

[9] EasyPrivacy. https://easylist.to/easylist/easyprivacy.txt.

[10] Firefox Storage Access Policy. https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Privacy/Storage_access_policy.

[11] Flashtalking Cookie Rejection. https://www.flashtalking.com/identity-management#cookie-rejection.

[12] Full Third-Party Cookie Blocking and More. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[13] PageGraph's open source implementation. https://github.com/brave/brave-browser/wiki/PageGraph.

[14] Platform for Privacy Preferences. https://www.w3.org/P3P.

[15] Privacy Badger. https://www.eff.org/privacybadger.

[16] Protecting Against HSTS Abuse. https://webkit.org/blog/8146/protecting-against-hsts-abuse/.

[17] Setting First-Party Cookies by Redirection. https://patents.google.com/patent/US20150052217A1.

[18] SSP to DSP Cookie Syncing Explained. https://www.adopsinsider.com/ad-exchanges/cookie-syncing/, 2011.

[19] Salesforce KUID. https://konsole.zendesk.com/hc/en-us/articles/115013802488-KUID, 2020.

[20] Understanding Calls to the Demdex Domain. https://experienceleague.adobe.com/docs/audience-manager/user-guide/reference/demdex-calls.html?lang=en#reference, 2021.

[21] ACAR, G., EUBANK, C., ENGLEHARDT, S., JUAREZ, M., NARAYANAN, A., AND DIAZ, C. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *CCS* (2014).

[22] ALRIZAH, M., ZHU, S., XING, X., AND WANG, G. Errors, Misunderstandings, and Attacks: Analyzing the Crowdsourcing Process of Ad-blocking Systems. In *IMC* (2019).

[23] ANTONIADES, D., POLAKIS, I., KONTAXIS, G., ATHANASOPOULOS, E., IOANNIDIS, S., MARKATOS, E. P., AND KARAGIANNIS, T. We.b: The web of short urls. In *World Wide Web Conference* (2011).

[24] AYENSON, M. D., WAMBACH, D. J., SOLTANI, A., GOOD, N., AND HOOFNAGLE, C. J. Flash cookies and privacy ii: Now with html5 and etag respawning. *World Wide Web Internet and Web Information Systems* (2011).

[25] AZAD, B. A., STAROV, O., LAPERDRIX, P., AND NIKIFORAKIS, N. Web runner 2049: Evaluating third-party anti-bot services. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2020).

[26] BASHIR, M. A., ARSHAD, S., ROBERTSON, W., AND WILSON, C. Tracing information flows between ad exchanges using retargeted ads. In *USENIX Security Symposium* (2016).

[27] BENCHAITA, W., GHAMRI-DOUDANE, S., AND TIXEUIL, S. On the optimization of request routing for content delivery. In *SIGCOMM* (2015).

[28] BHAGAVATULA, S., DUNN, C., KANICH, C., GUPTA, M., AND ZIEBART, B. Leveraging Machine Learning to Improve Unwanted Resource Filtering. In *ACM Workshop on Artificial Intelligence and Security* (2014).

[29] BREIMAN, L. Random Forests. In *Machine learning* (2001).

[30] BURGESS, J., CARLIN, D., O'KANE, P., AND SEZER, S. Redirekt: Extracting malicious redirections from exploit kit traffic. In *2020 IEEE Conference on Communications and Network Security (CNS)* (2020).

[31] CHANG, L., HSIAO, H.-C., JENG, W., KIM, T. H.-J., AND LIN, W.-H. Security implications of redirection trail in popular websites worldwide. In *World Wide Web* (2017).

[32] CHEN, H., ZHANG, H., BONING, D., AND HSIEH, C.-J. Robust decision trees against adversarial examples. In *International Conference on Machine Learning* (2019).

[33] CHEN, H., ZHANG, H., SI, S., LI, Y., BONING, D., AND HSIEH, C.-J. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems* (2019).

[34] CHHABRA, S., AGGARWAL, A., BENEVENUTO, F., AND KUMARAGURU, P. Phi.sh/$ocial: The phishing landscape through short urls. In *Proceedings of the Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference* (2011).

[35] CIMPANU, C. Ad Network Uses DGA Algorithm to Bypass Ad Blockers and Deploy In-Browser Miners. https://www.bleepingcomputer.com/news/security/ad-network-uses-dga-algorithm-to-bypass-ad-blockers-and-deploy-in-browser-miners/, 2018.

[36] CRANOR, L. F., EGELMAN, S., SHENG, S., MCDONALD, A. M., AND CHOWDHURY, A. P3P Deployment on Websites. In *Electronic Commerce Research and Applications* (2008).

[37] ENGLEHARDT, S., HAN, J., AND NARAYANAN, A. I never signed up for this! Privacy implications of email tracking. In *PETS* (2018).

[38] ENGLEHARDT, S., AND NARAYANAN, A. Online Tracking: A 1-million-site Measurement and Analysis. In *ACM Conference on Computer and Communications Security (CCS)* (2016).

[39] FOUAD, I., BIELOVA, N., LEGOUT, A., AND SARAFIJANOVIC-DJUKIC, N. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. In *PETS* (2020).

[40] FULGHAM, B. Protecting Against HSTS Abuse. https://webkit.org/blog/8146/protecting-against-hsts-abuse/, 2018.

[41] GAO, H., HU, J., WILSON, C., LI, Z., CHEN, Y., AND ZHAO, B. Y. Detecting and characterizing social spam campaigns. In *ACM IMC* (2010).

[42] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[43] GOOGLE. RTB - Cookie Matching. https://developers.google.com/authorized-buyers/rtb/cookie-guide.

[44] GOSSEN, D., JONKER, I. H., AND POLL, I. E. *Design and implementation of a stealthy OpenWPM web scraper*. PhD thesis, Master's thesis, Radboud Universiteit Nijmegen, 2020.

[45] GUGELMANN, D., HAPPE, M., AGER, B., AND LENDERS, V. An Automated Approach for Complementing Ad Blockers' Blacklists. In *PETS* (2015).

[46] IKRAM, M., MASOOD, R., TYSON, G., KAAFAR, M. A., LOIZON, N., AND ENSAFI, R. The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading. In *The Web Conference (WWW)* (2019).

[47] IQBAL, U. AdGraph's open source implementation. https://github.com/uiowa-irl/AdGraph.

[48] IQBAL, U., SHAFIQ, Z., AND QIAN, Z. The Ad Wars: Retrospective Measurement and Analysis of Anti-Adblock Filter Lists. In *IMC* (2017).

[49] IQBAL, U., SNYDER, P., ZHU, S., LIVSHITS, B., QIAN, Z., AND SHAFIQ, Z. AdGraph: A Graph-Based Approach to Ad and Tracker Blocking. In *To appear in the Proceedings of the IEEE Symposium on Security & Privacy* (2020).

[50] KANTCHELIAN, A., TYGAR, J. D., AND JOSEPH, A. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning* (2016).

[51] KOOP, M., TEWS, E., AND KATZENBEISSER, S. In-Depth Evaluation of Redirect Tracking and Link Usage. In *Proceedings on Privacy Enhancing Technologies (PETS)* (2020).

[52] LE PAGE, S., JOURDAN, G., BOCHMANN, G. V., FLOOD, J., AND ONUT, I. Using url shorteners to compare phishing and malware attacks. In *2018 APWG Symposium on Electronic Crime Research (eCrime)* (2018).

[53] LENDACKY, T. The Platform for Privacy Preferences (P3P). https://www-archive.mozilla.org/projects/p3p.

[54] MAGGI, F., FROSSI, A., ZANERO, S., STRINGHINI, G., STONE-GROSS, B., KRUEGEL, C., AND VIGNA, G. Two years of short urls internet measurement: Security threats and countermeasures. In *Proceedings of the 22nd International Conference on World Wide Web* (2013).

[55] MDN. Redirect Tracking Protection. https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Privacy/Redirect_Tracking_Protection, 2020.

[56] MEKKY, H., ZHI-LI, R., ZHANG, SAHA, S., AND NUCCI, A. Detecting malicious http redirections using trees of user browsing activity. In *INFOCOM* (2014).

[57] OLEJNIK, L., TRAN, M.-D., AND CASTELLUCCIA, C. Selling off privacy at auction. In *Network and Distributed System Security Symposium (NDSS)* (2014).

[58] PAPADOPOULOS, P., KOURTELLIS, N., AND MARKATOS, E. P. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The Web Conference* (2019).

[59] QUINLAN, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[60] REN, T., WITTMAN, A., CARLI, L. D., AND DAVIDSON, D. An analysis of first-party cookie exfiltration due to cname redirections. In *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)* (2021).

[61] SANCHEZ-ROLA, I., BALZAROTTI, D., KRUEGEL, C., VIGNA, G., AND SANTOS, I. Dirty Clicks: A Study of the Usability and SecurityImplications of Click-related Behaviors on the Web. In *The Web Conference (WWW)* (2020).

[62] SHUBA, A., MARKOPOULOU, A., AND SHAFIQ, Z. NoMoAds: Effective and Efficient Cross-App Mobile Ad-Blocking. In *PETS* (2018).

[63] SIBY, S., IQBAL, U., ENGLEHARDT, S., SHAFIQ, Z., AND TRONCOSO, C. Webgraph: Capturing advertising and tracking information flows for ro-bust blocking. In *To appear in the USENIX Security Symposium* (2022).

[64] SJÖSTEN, A., SNYDER, P., PASTOR, A., PAPADOPOULOS, P., AND LIVSHITS, B. Filter List Generation for Underserved Regions. In *WWW* (2020).

[65] SNYDER, P., VASTEL, A., AND LIVSHITS, B. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. In *ACM SIGMETRICS* (2020).

[66] TSYMBAL, A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin 106* (2004).

[67] WANG, W., ZHENG, Y., XING, X., KWON, Y., ZHANG, X., AND EUGSTER, P. WebRanz: Web Page Randomization For Better Advertisement Delivery and Web-Bot Prevention. In *ACM SIGSOFT FSE* (2016).

[68] WEBKIT. Tracking Prevention in WebKit. https://webkit.org/tracking-prevention/.

[69] WILANDER, J. Bounce Tracking Protection. https://github.com/privacycg/proposals/issues/6.

[70] WILANDER, J. Clear-Site-Data For Cross-Site Tracking. https://github.com/privacycg/storage-partitioning/issues/11.

[71] WILANDER, J. ITP 2.0. https://webkit.org/blog/8311/intelligent-tracking-prevention-2-0/.

[72] WILANDER, J. ITP 2.2. https://webkit.org/blog/8828/intelligent-tracking-prevention-2-2/.

[73] WILANDER, J. ITP 2.3. https://webkit.org/blog/9521/intelligent-tracking-prevention-2-3/.

[74] WILANDER, J. Safari ITP Classifier. https://bugs.webkit.org/show_bug.cgi?id=168347.

[75] WILANDER, J. CNAME Cloaking and Bounce Tracking Defense. https://webkit.org/blog/11338/cname-cloaking-and-bounce-tracking-defense/, 2020.

[76] YU, Z., MACBETH, S., MODI, K., AND PUJOL, J. M. Tracking the Trackers. In *World Wide Web Conference* (2016).

[77] ZAIFENG, Z. Who is Stealing My Power III: An Adnetwork Company Case Study, 2018. .

A Artifact Appendix

A.1 Abstract

We propose KHALEESI, a machine learning (ML) approach that captures the essential sequential context needed to effectively detect advertising and tracking request chains. We release KHALEESI's classification code, ML model, browser extension, and data sets. Classification code is written in Python 3.6, the ML model is trained using Scikit, the browser extension is written in JavaScript/HTML, and the data is crawled using OpenWPM.

A.2 Artifact check-list (meta-information)

- **Binary:** A browser extension to block advertising and tracking request chains. The extension is designed and tested in Mozilla Firefox.
- **Model:** ML model to detect advertising and tracking request chains. Released ML model was trained on request chains from homepages of Alexa top-10K websites.
- **Data set:** Data sets to train and test ML model. We release crawls of homepages, home and sub pages, home and sub pages with cookies blocked, and home and sub pages with browser spoofed as Safari. All data sets are crawls of Alexa top-10K websites. The data contains requests, responses, and JS execution.
- **Run-time environment:** Scripts can be run using Python 3.6 and above. The code was tested on Ubuntu 16.04.7 LTS.
- **How much disk space required (approximately)?:** We recommend a disk space of ∼100GB to train the classifier. The browser extension does not have any disk space constraints.
- **How much time is needed to complete experiments (approximately)?:** The classifier can be trained in ∼ 10 hours. The browser extension blocks the ads instantaneously.
- **Publicly available (explicitly provide evolving version reference)?:** KHALEESI's code, data, and browser extension is available at https://uiowa-irl.github.io/Khaleesi/.
- **Archived (explicitly provide DOI or stable reference)?:** KHALEESI's code, data, and browser extension is available at https://github.com/uiowa-irl/Khaleesi/tree/bd28513878a363b39b0ee9e7a6a4350f71672912

A.3 Description

A.3.1 How to access

KHALEESI's code, ML model, and browser extension are available on Github at: https://uiowa-irl.github.io/Khaleesi/. Data sets are available on Zenodo at: https://doi.org/10.5281/zenodo.6084582.

A.3.2 Hardware dependencies

KHALEESI ML model was trained on a machine with 16 cores and 96 GB RAM. We recommend a disk space of ∼100 GB to train the classifier. The model can be tested on hardware with less resources.

A.3.3 Software dependencies

KHALEESI browser extension was designed and tested on Mozilla Firefox. We trained and tested KHALEESI ML model on Ubuntu 16.04.7 LTS.

A.3.4 Data set dependencies

KHALEESI is trained on data set crawled through OpenWPM version 0.10.0. The code might require some minor modifications to process data from newer versions of OpenWPM.

A.4 Installation

We provided instructions to run KHALEESI on Github.

A.5 Experiment workflow

In addition to instructions on Github, we provide detailed instructions to run the code below:

A.5.1 Training & Testing ML model

We list the step-by-step process to train and test KHALEESI's ML model below:

1. *Data collection:* Collect network and JavaScript initiated requests using OpenWPM.
2. *Request chain construction:* Organize network and JavaScript initiated requests into chains. Request chains can be constructed with HTTP and JavaScript chain construction scripts.
3. *Request chain labeling:* Once constructed, label request chains using EasyList (EL) and EasyPrivacy (EP) filter lists. Use filter list labeling script and EL/EP filter lists to label the chains.
4. *Feature extraction and transformation:* After labeling, extract features from the request chains using using feature extraction script and encode them using feature encoding script.
5. *Model training:* Since, KHALEESI relies on previous confidence as a feature, extract the previous confidence for each request in a chain before training the final model. The previous confidence can be extracted using compute previous confidence script. The last block of previous confidence script stores the final trained model. An already trained model is available in data directory.
6. *Testing the model:* KHALEESI uses 10-fold cross validation to test the data sets. The encoded features with previous confidence can be tested using test classifier script and the accuracy can be computed using compute accuracy script.

A.5.2 Analysis of Request Chains

We release scripts to analyze cookie syncing and bounce tracking instances in request chains. Use the cookie syncing and bounce tracking scripts to identify cookie syncing and bounce tracking instances, respectively.

A.5.3 Browser Extension

To add KHALEESI to Firefox, enter `about:debugging` in the URL bar, click *This Firefox*, click *Load Temporary Add-on*, navigate to the extension's directory and open `manifest.json`. To view the requests blocked by KHALEESI, open extension's console by clicking *Inspect* in `about:debugging` or see the network tab in the Firefox Developer Tools.

A.6 Evaluation and expected results

Training & Testing ML Model: Upon successful execution, the workflow should produce a trained ML model and output its accuracy.

Analysis of Request Chains: Upon successful execution, the scripts should list the cookie syncing and bounce tracking instances in request chains.

Browser Extension: After installation, the browser extension should block advertising and tracking request chains.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.