TrackSign: Guided Web Tracking Discovery

Ismael Castell-Uroz
Universitat Politècnica de Catalunya
Barcelona, Spain
icastell@ac.upc.edu

Josep Solé-Pareta
Universitat Politècnica de Catalunya
Barcelona, Spain
pareta@ac.upc.edu

Pere Barlet-Ros
Universitat Politècnica de Catalunya
Barcelona, Spain
pbarlet@ac.upc.edu

***Abstract*—Current web tracking practices pose a constant threat to the privacy of Internet users. As a result, the research community has recently proposed different tools to combat well-known tracking methods. However, the early detection of new, previously unseen tracking systems is still an open research problem. In this paper, we present *TrackSign*, a novel approach to discover new web tracking methods. The main idea behind *TrackSign* is the use of code fingerprinting to identify common pieces of code shared across multiple domains. To detect tracking fingerprints, *TrackSign* builds a novel *3-mode network graph* that captures the relationship between fingerprints, resources and domains. We evaluated *TrackSign* with the top-100K most popular Internet domains, including almost 1M web resources from more than 5M HTTP requests. Our results show that our method can detect new web tracking resources with high precision (over 92%). *TrackSign* was able to detect 30K new trackers, more than 10K new tracking resources and 270K new tracking URLs, not yet detected by most popular blacklists. Finally, we also validate the effectiveness of *TrackSign* with more than 20 years of historical data from the Internet Archive.**

***Index Terms*—web tracking, code signature, fingerprinting, network graph, 3-mode graph**

I. Introduction

Although online privacy has brought much attention in the recent years (e.g. [1], [2]), the fast evolution [3] and inherent complexity of the Internet [4] make it very difficult to develop effective privacy protection methods. Web tracking –a collection of techniques developed to identify users across multiple domains, browsers and devices– is the main tool used by web services to compile large amounts of personal data about their users [5]. Previous works have shown that the collected information is used for many purposes, such as targeted advertisement [6] and search customization [7], but also for more obscure practices, including price discrimination [8], credit scoring [9] or personal financial assessment [10].

Over the last decade, the research community has made great efforts to combat web tracking (e.g., [11]–[20]). Although some of these works have succeeded in the detection of well-known tracking techniques, the early detection of new, previously unseen web tracking methods still remains an open research problem. Nowadays, the only reliable way to discover new tracking systems falls to human experts with the daunting task of analyzing millions of websites. Fortunately, there are ways to narrow down the search to a manageable number under specific circumstances. For instance, experts can study the characteristics introduced by new web standards and programming languages that could potentially be exploited by new tracking methods. Although the community has developed some privacy measurement tools to facilitate this task (e.g. [13], [21]), all the process requires a high degree of expertise and it is both, hard and time consuming.

In this work, we seek to transform this blind chase for new web tracking mechanisms into a guided hunt that can lead to results in a much faster and effective way. The intuition behind our proposal is based on the observation that: (i) there are relatively few tracking approaches, but they are shared across many different domains, and (ii) different Internet resources including the same or similar tracking methods share some fragments of code (e.g., JavaScript API calls used to perform the actual tracking). If we could focus our lens on relevant pieces of code with those characteristics from all the resources available on the Internet, there would be a high chance they belong to tracking systems.

Based on this observation, we present *TrackSign*, a new web tracking discovery system that automatically crawls the Internet in the search of small pieces of code that are shared across multiple domains. To this end, we use a file partition method based on Rabin Fingerprinting [22], which allows us to split the Internet resources (e.g., HTML, JavaScript files) in an unambiguous way based on its content. To distinguish between fingerprints with higher probability to belong to tracking systems from other shared code (e.g., JavaScript libraries), *TrackSign* builds a *3-mode network graph* of the Internet, which captures the relationship between the computed fingerprints and the resources and domains that use them. With the 3-mode network graph and the generated code signatures, *TrackSign* automatically analyzes the most popular code looking for "dirty" fingerprints; signatures with a high probability of pertaining to web tracking algorithms. Once discovered, all the resources using those signatures are automatically classified as new tracking systems as well.

We used *TrackSign* to explore the top 100K most popular domains as per the Alexa's list [23]. The resulting data set contains about 933K resources and 73M signatures collected from more than 5M HTTP requests. The data set and source code of *TrackSign* and *ORM* (described later in this paper) is publicly available on [24]. The resources were also labeled using the most popular URL pattern lists currently available. Our results show that our method can detect new tracking resources with high precision (over 92%). After exploring the 100K most popular domains, *TrackSign* was able to detect 30.000 new trackers, almost 12.000 new tracking JavaScript

 DOI 10.1109/INFOCOM42981.2021.9488842

files and more than 273K new tracking URLs, which are not recognized by the exiting pattern lists. Finally, in order to show the potential of *TrackSign* in the detection of new tracking systems raising over time, we explored more than 20 years of historical data from the top 500 most popular websites using the Internet Archive's *Wayback Machine* [25].

The rest of the paper is organized as follows. Section II reviews the most relevant related work. Section III describes the fundamentals behind TrackSign. The design of the measurement system is introduced in Section IV, while Section V presents the evaluation. Section VI discusses limitations and future work. Finally, Section VII concludes the paper.

II. Related work

This section reviews the related work most relevant to *TrackSign*. Many previous works tried to block web tracking methods using machine learning (ML). In contrast, our proposal tries to discover new tracking algorithms and resources using a completely different approach. Thus, *TrackSign* can be seen as complementary to previous literature.

1) ***JavaScript and DOM properties****:* Acar et al. in [13] and Ikram et al. in [16] proposed ML classifiers to inspect specific JavaScript API calls that could be used for browser or device fingerprinting. Similarly, Wu et al. in [11] use support vector machines trained with tracking JavaScript code to differentiate between tracking and functional code. All of them base their classification on static code analysis, making them vulnerable to obfuscation techniques [15].

Iqbal et al. propose in [12] a method based on the instrumentation of *Google Chromium* called AdGraph. It makes use of a complex combination of JavaScript code unit attribution, HTML DOM inspection and network link annotation to characterize changes being executed in the document. With the obtained output they train a random forest with high accuracy replicating the labels of human-generated filter lists. However, in those cases where there is a dispute between them (the classification does not match) the rate of false positives and false negatives is higher than the tracking lists themselves.

2) ***Network properties****:* In [20] Gugelmann et al. present a system developed to improve filter lists using network measurements to characterize web trackers. The system looks the behavior of the explored website requests (e.g., received bytes, #referrers) and aggregates them by domain. With the collected information they train several ML algorithms and select the best performing one. The resulting system presents mixed results, with a high precision in the detection of some new web tracking domains, but a lack of fine-grained classification.

Kalavri et al. explore in [14] the network as a bipartite graph, where nodes are either the main site opened by the user, or the resource URL called from the main site. The edges then indicate a relationship between a domain and its loaded URLs. Using a projection of the graph, they find that subsets of web trackers form well-defined communities. Based on this property, they develop a system with high accuracy finding those web trackers. However, the method is unable to find trackers outside the communities (e.g., tracking systems using quickly changing domains to serve the content).

3) ***URL properties****:* Yu et al. in [18] and Metwalley et al. in [19] present both similar systems based on the detection of unique identifiers inside URL requests. For this purpose, they compare the requests of multiple users accessing the same domain, looking for parameters that could be used as an identifier. Thanks to this collaborative approach, the method presents high accuracy finding possible identifiers. However, despite the high accuracy, they are only able to detect tracking based on direct identification, which does not include more complex techniques like fingerprinting.

On the other hand, popular content-blockers, such as *AdBlock Plus* [26], *Ghostery* [27], *uBlock Origin* [28] and *Disconnect* [29], block resources at the URL level. Despite their differences, all of them use filter lists, such as the well-known *EasyList* [30] or *EasyPrivacy* [31], to block host tracking resources. The maintenance of those lists is done manually by the community. The main advantage of this approach is that URL filtering is lightweight and easy to deploy in browsers. However, their main drawback is the manually-curated and static nature of the filter lists, which significantly slows down the detection of new web tracking systems. Moreover, this approach can be easily circumvented by changing periodically the domains used to serve the tracking resources.

4) ***Measurement frameworks****:* The research community has also developed tools and frameworks to perform large-scale studies on web tracking topics. For instance, Englehardt et al. present in [21] a framework to explore a large number of websites and analyze the calls or parameters of special interest to the user. In [32] Lécuyer et al. share a framework to audit DOM objects to study the relation between the data sent by the browser and the advertisements received as result. Li et al. present in [33] an instrumented version of Google Chromium designed for forensic studies, recording the website loading process in a way that it can be reconstructed.

III. TrackSign

A. The concept

In this paper we introduce *TrackSign*, a new technique to discover new web tracking mechanisms. The basis of our proposal is to use signatures to massively compare code of millions of online resources and find pieces of code shared by multiple non-related websites. Our proposal is based on two observations; First, there is a limited number of ways for web tracking systems to legitimately obtain user information. For instance, using HTML or JavaScript programming languages to obtain user data restricts the tracking possibilities to the HTML or the JavaScript API calls respectively. Thus, web tracking systems based on those languages will necessarily share some common code making use of those API calls. Second, any new effective web tracking algorithm will, at some point, become popular enough to be present in multiple websites. Considering both of them, if we are able to find common code shared by different websites, there is a high chance that web tracking code is included in it.

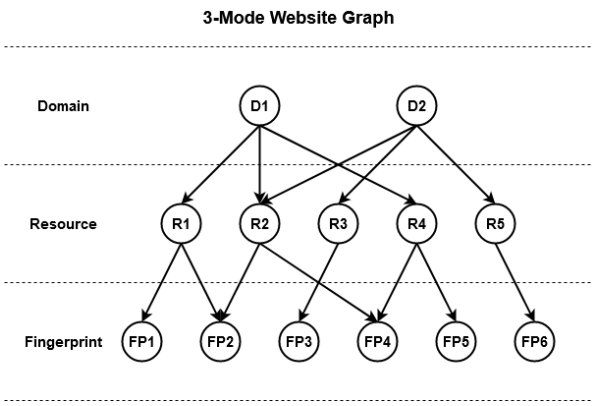

Fig. 1. **Restricted 3-mode network graph**: Represents the relation between the resources loaded by each domain and the code included in each resource. *Domain* and *Fingerprint* nodes could not have edges connecting them.

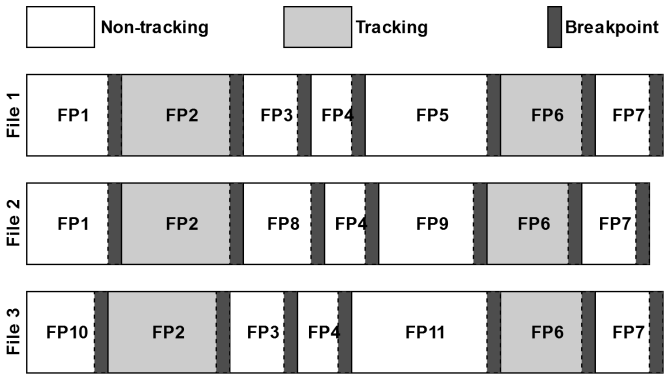

Fig. 2. **File partition and signature example**. Three files with different hash values share most part of the code. Fingerprinting algorithm finds shared code by deciding breakpoints based on the contents of the file.

There is a clear obstacle, though. Resources sharing pieces of code are probably not limited to tracking resources, but also include functional resources. To be able to identify them we can use the source of the fingerprint, differentiating between signatures pertaining to web tracking resources and fingerprints obtained from non-tracking files. Unfortunately, the inherent distributed nature of the Internet allows the same file to be hosted in several different servers and linked by many different URLs. Accounting for them as different resources would artificially inflate the actual figure of code included in that resource, whether it is tracking or not. This would modify its popularity and break the correct classification of the corresponding fingerprints. To solve it, we make use of the Online Resource Mapper (ORM), a new open-source framework designed to explore the web from the perspective of its resources (described later in Section IV).

B. Network graph

ORM defines the web as a 3-mode network graph focused on resources and fingerprints. A 3-mode or tripartite graph is defined as a graph $G = (U, V, W, E)$ whose nodes can be divided into three disjoint and independent sets U, V and W such that every edge E connects a vertex in one of the sets to a vertex in other different set. In our model, each of the node sets corresponds to *domains*, *resources* and *fingerprints*. Edges between *resource* and *fingerprint* nodes symbolize the relation between the resource file and its code pieces. On the other hand, edges between *domain* and *resource* nodes state the relation between the domain and the resources loaded by that domain, regardless the host, country, URL or server where they are hosted. Figure 1 shows the structure of the graph. In our case, the resulting graph is a *restricted* 3-mode graph, where there should not be edges going from *domain* nodes to *fingerprint* nodes and vice versa.

Looking at the network from this perspective has several advantages. First, it solves the problem exposed above, aggregating all the different URLs pointing to the same file in only one *resource* node. Moreover, the indegree (incoming edges) of each node represents the popularity of that node. This allows us to easily know the most popular resources and fingerprints looking only at the degree of each node. Furthermore, inspecting the resources connected to each *fingerprint* node we can quantify the number of tracking resources containing the fingerprint, useful for the classification process.

C. File partition

TrackSign is based on computing code signatures in order to compare online content. The comparison could be done dividing the files into fixed-size, non-overlapping blocks, and then calculate the prevalence of such blocks. However, introducing only one character at the first position of the file would generate completely different blocks making the comparison to fail. Instead, we divide the file into variable-length content blocks based solely on the content of the file. The selected algorithm, similar to the solution presented in [34] to deduplicate file blocks on a network file system, computes a *rolling hash* over a 48-byte sliding window of the content. A rolling-hash is nothing more than a hash function computed over a sliding window. Any hash function can be used, but the most appropriate functions compute the next value directly from the current value and the new received byte. We selected Rabin fingerprinting [22] as our hash function, because it is based on polynomials and meets the condition explained above. A Rabin fingerprint fp is the modulo of a polynomial representation of the string and a predetermined prime polynomial.

The signature generation process begins by the start of the file and moves towards the end one byte at a time. The content block length is determined probabilistically selecting a *breakpoint* bp value. When the signature fp matches the breakpoint ($fp \equiv bp \pmod{a}$) the block finishes and the fingerprint computed until that point is assigned as the signature of the block. Figure 2 shows a visual example of three files sharing common pieces of code whose fingerprints mostly match although the files are not completely equal. The algorithm ensures that, if the same substring is present in different files, the fingerprint computed to identify the substring will be the same independently of its position in the files.

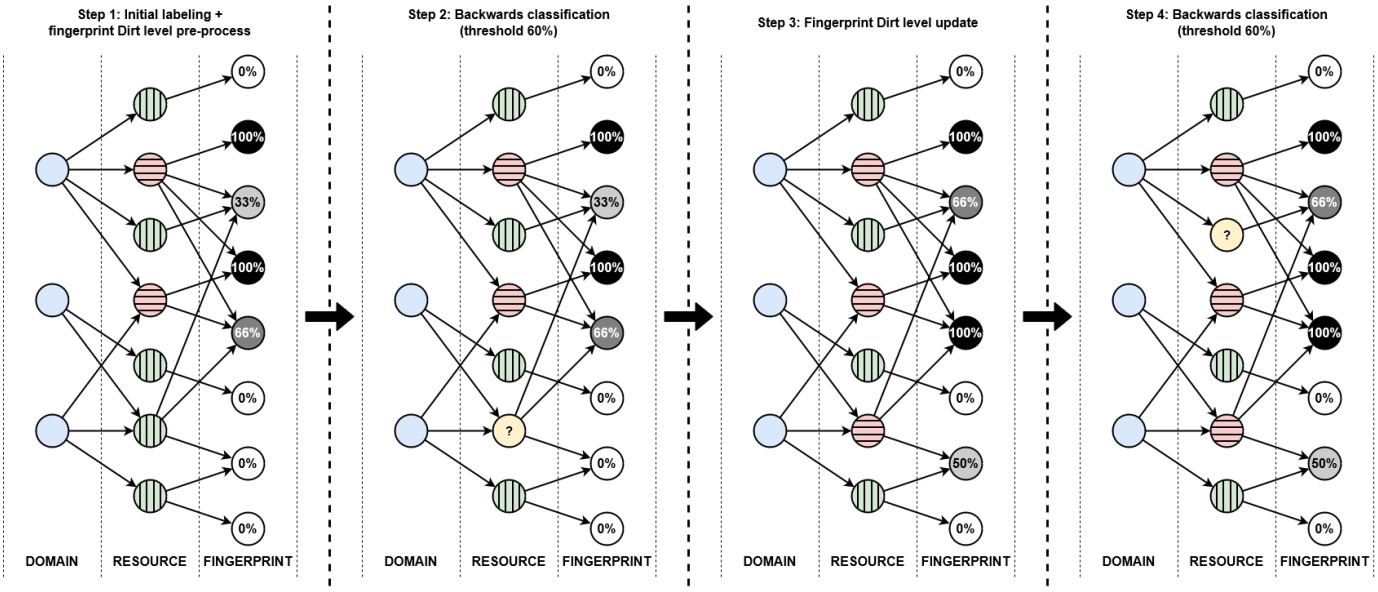

Fig. 3. **Dirt level graph convergence**: Representation of the iterative detection algorithm. Each two steps form a new iteration. The first step updates the *Dirt Level* of each fingerprint. The second step searches non-tracking resources connected to a *dirty fingerprint* and classifies them as tracking.

Thanks to this probabilistic partition approach, the algorithm permits to setup an *average block length*; assuming random content, the probability that a signature fp is equal to the breakpoint bp (mod a) is 1/a. Unfortunately, this also implies that may be cases where the blocks are very short or very long, selecting only a few characters or all the file if the breakpoint value is not present. To solve it, the algorithm permits to set a *minimum* and *maximum block length*.

D. Web tracking detection

Unknown web tracking can be grouped in three categories: (i) completely new web tracking algorithms with previously unseen code, (ii) unknown tracking files based on already known tracking methods or (iii) known tracking files on unknown URLs. TrackSign detection algorithm is able to detect (ii) and (iii) in a fully automatic fashion, and highlight the code with a high probability of belonging to (i), so that an expert can further verify it.

Our method first focuses on the popularity of the signatures of the graph. If a new fingerprint becomes popular, chances are that it pertains to a new popular service or it is part of a new web tracking system. Either way, it is a new point of special interest to be revised and correctly classified. We define the popularity as the number of different domains loading resources that contain the inspected fingerprint. In other words, we are accounting for the number of *domain* nodes that are at distance 2 of the *fingerprint* node. The formal definition is:

$$Popularity = |\{d | d \in D, distance(d, fp) = 2\}|$$

To discover "popular" signatures we setup a threshold on the number of domains to be considered as new relevant content. For completely new files, not sharing code with any previously classified resource, human experts have to discover the reason for the increase in popularity. In this context, the popularity threshold provides a way to automatically narrow down the search to the most important resources. The lower the threshold the earlier we will discover the new code. Thus, the threshold value represents a trade-off between the amount of code we want the algorithm to focus on, and the human resources available to revise the obtained content.

However, for new signatures becoming popular but shared by already known resources, TrackSign permits to automatically classify them. To this end, we can take advantage of the source of each obtained fingerprint. The intuition behind it is that, if a popular fingerprint is mostly present in files performing tracking, there is a high probability that new files containing the same fingerprint are also performing tracking. In order to calculate that probability, we explore all the *resource* nodes connected to each *fingerprint*, quantifying the percentage of them that were already classified as trackers. We called that probability the *Dirt* level of the fingerprint. The formal definition is given by:

$$Dirt\ level = \frac{\sum_{r \in Neighborhood(fp)} is\_tracking(r)}{deg^{-}(fp)}$$

Once we have quantified the *Dirt* level of all the popular fingerprints, we setup a threshold based on their "dirtiness". All the resources connected to a *dirty fingerprint* are automatically classified as tracking resources. Note that reclassifying some resources modifies again the *Dirt* level of each fingerprint connected to those resources, starting over the cycle again. This iterative algorithm is executed until the graph converges and all the resources connected to fingerprints exceeding the selected threshold are classified as tracking.

Figure 3 shows an example of this iterative process using a threshold of 60%. In the first step the system computes

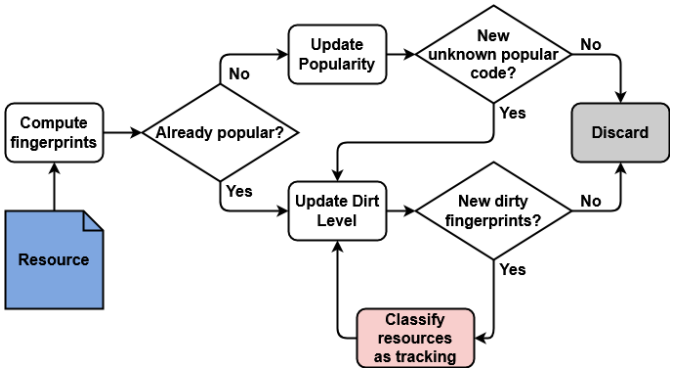

Fig. 4. TrackSign high level detection diagram

the *Dirt* level for each fingerprint using the labeled ground-truth (red→tracking, green→non-tracking). The second step classifies as tracking all the resources connected to fingerprints with a *Dirt* level higher than the selected threshold. In the figure there is only one dirty fingerprint (66%) connected to a resource not already classified as tracking. The third step starts the iteration process again updating the *Dirt* level of the fingerprints connected to new tracking resources found in step 2. The fourth step searches again for new resources connected to dirty fingerprints. This process allows us to detect resources with a high probability of performing web tracking. As we will show in Section V, selecting the correct threshold ensures a high precision while detecting thousands of new web tracking resources. Figure 4 includes a high level diagram of the TrackSign detection algorithm.

IV. System Design

In this section, we present the design and implementation of the Online Resource Mapper (ORM), an open source large-scale web measurement framework specifically designed to explore the web as a 3-mode graph of the resources contained in it. ORM builds on similar technologies as many previous platforms, but with key design differences that allows it to aggregate information at the resource level. We divided the data collector system in two modules: *driver managers* which are in charge of browsing the web, and *data managers* which explore the network events looking for new resources and aggregate the graph information sent to the database. The analysis is done using individual scripts in a post-processing phase. This modular implementation allows the system to be executed in parallel and perform large-scale experiments. Python is the language used to build the entire platform. A high level diagram of the system is depicted in Figure 5.

1) ***Driver manager****:* ORM uses *driver managers* to artificially explore the web and obtain data. Each driver manager maintains an instance of *Selenium* [35] running a full-fledged browser on an independent process. The selected browser is *Chromium*, the open source version of *Google Chrome*. This combination was selected to be able to automatize the collection process while maintaining compatibility with all the technologies needed to load current websites. Moreover,

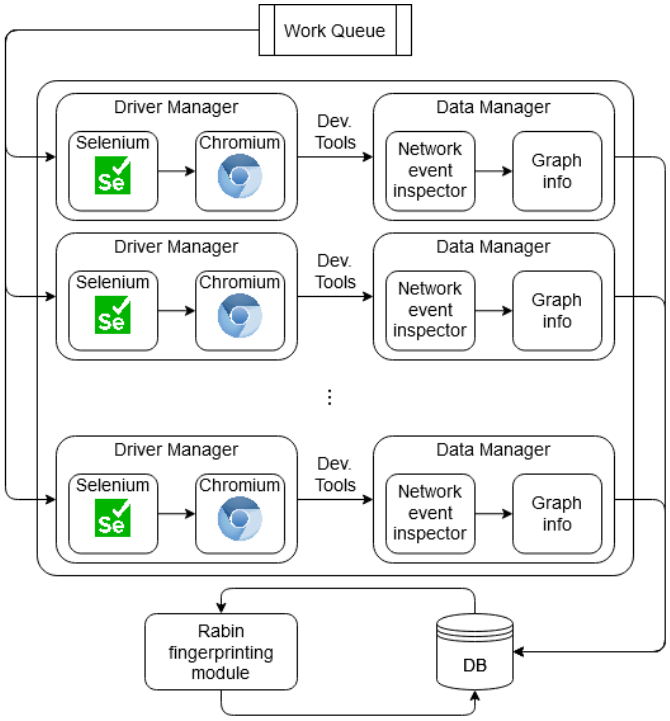

Fig. 5. ORM high-level structure diagram

Chromium supports *Google*'s DevTools protocol, allowing us to access all the internal parameters and communications made by the browser. *Driver managers* are also in charge of providing stability to the system. Selenium is known to crash under some circumstances. The unpredictability of network events, and the presence of website code not following the standards can make the driver to hang. *Driver managers* detect those cases and reboot the Selenium driver, skipping the website presenting the problem. Finally, the collected information is sent to data managers.

2) ***Data manager****: Data managers* receive all the network events produced by the DevTools protocol and inspect them looking for URLs being loaded by the browser. The found URLs are independently downloaded, compressed and saved inside the database. For each file the system computes a hash value used as a primary key. This serves to deduplicate files and aggregate information on the same resource nodes. Furthermore, the system also computes a fuzzy-hash value to correlate different versions of the same file, very common between popular libraries like *JQuery*. Finally, the *data manager* stores the information aggregated by resource in the database, including the relation between the resource and the domain that made the call, and the relation between the resource and the URL that pointed to it.

3) ***Analysis scripts****:* Code fingerprinting is not embedded inside the data managers to detach the data acquisition process from the experimental phase. Moving it to the analysis phase permits us to test several different parameters and configurations with a single exploration of websites. For other analyses, a simple access to the database provides large amounts of

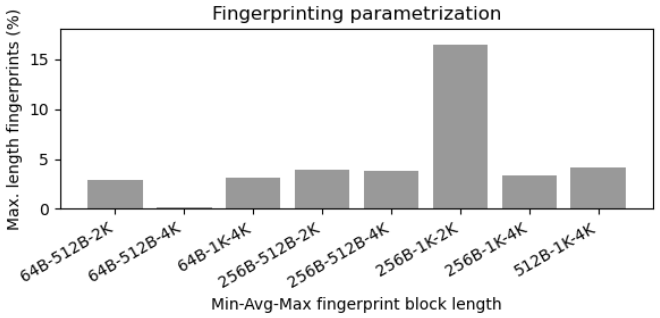

Fig. 6. Fingerprinting parametrization comparison

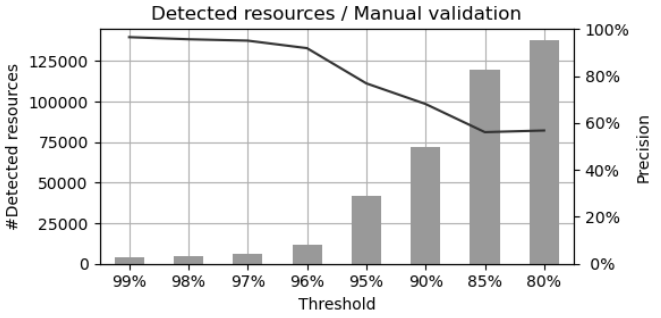

Fig. 7. **Web tracking detection results**: Bars represent the total new tracking resources detected for each *Dirt* threshold. The line states the detection precision based on manual validation (top 50 more and top 50 less popular detected resources)

information that can be extracted with specific analysis scripts.

V. Evaluation

A. Setup

To evaluate the system, we launched ORM to collect an initial ground-truth of the top 100K most popular websites as per Alexa's list [23]. We decided to focus on JavaScript code (including code embedded in HTML) as previous works found that most tracking methods are based on it [5]. Nevertheless, our system is flexible enough to be applied also to other languages, such as HTML or CSS. The exploration was executed from an educational network infrastructure operating at several Gbit/s. We setup a timeout of 30 seconds for each explored web, usually long enough to request all the documents. From the initial population of 100K websites we ended up scraping successfully a total of 90.637 domains. The corresponding data set has more than 5.2M unique URLs and almost 933K online resources. All the data was collected on the period May∼June of 2020. The labeling process was performed using *uBlock Origin* [28], one of the most popular content-blockers on the market. It has been recommended by many works (e.g. [36], [37]) as one of the best privacy protecting tools currently available. *uBlock Origin* checks by default over a dozen different pattern lists including the two most popular and precise: EasyList [30] and EasyPrivacy [31]. It detected about 585K (≈11%) URLs performing tracking. Each resource loaded by those URLs was labeled as a tracking resource, giving a total sum of 318K (≈34%) tracking resources. To mitigate the effects of minification and obfuscation techniques [15], all the files are prettified using JSBeautifier [38]. This allows us to also find shared code between minified and non-minified files.

B. Parametrization

In a second phase we computed the code signatures for each resource, executing the fingerprinting modules over the collected ground-truth. As described in Section III-C, the module has three main configuration parameters: the *minimum*, *average* and *maximum block length*. The combination of them defines the length of the code block to be translated to a fingerprint. To investigate the influence of these parameters on the results we first executed manually a series of small experiments with a subset of only a few thousand resources.

As expected, setting small values partitions the code in too small pieces, oversizing the graph and losing the lexical value of the language. On the contrary, selecting large values results in entire files considered as a single code block. Thus, we selected empirically a *minimum block length* of at least 64 bytes and a maximum block length of 4 Kbytes. Moreover, according to our observations selecting non-optimal values for the *average* and *maximum block length* parameters forces the algorithm to prematurely cut some code blocks that normally would have a different fingerprint. This can be easily verified looking at the number of fingerprints having length equal to the *maximum block length* parameter. The cut makes the algorithm to start computing the next fingerprint from a code point where usually the algorithm would not start. Consequently, the code belonging to the new fingerprint may partially appear inside other fingerprints. In summary, this opens the door to create some additional fingerprints containing repeated code.

To minimize the effect, we executed a battery of tests with different configurations over a subset of 200K resources to explore the real effect on the fingerprints. Figure 6 presents the obtained results. For most of the selected combinations the results were equivalent, presenting about 3∼4% of fingerprints with maximum length. The combination of 256B-1K-2K shows the worst result with more than 16% of fingerprints being cut prematurely. This suggests a relation with the small difference between the *average* and the *maximum block length*, where the second is just twice the first one. In the rest of the configurations the *maximum block length* is at least 4 times the *average block length*. Finally, the best result corresponds to the 64B-512B-4K combination, reducing the prematurely cut fingerprints to only a 0.2%. Thus, we selected it to fingerprint the entire resource collection. It took about one week to process all the resources, and the resulting data set includes more than 73M of code fingerprints.

C. Results

The system was evaluated trying different *Dirt* thresholds (80%∼99%) over the entire data set. The *Dirt* level of the fingerprint accounts the percentage of resources that were previously classified as web tracking and contain this specific

TABLE I
WEB TRACKING DETECTION

	Resources	Fingerprints	URLs	Trackers
Total	932.711	73.660.398	5.231.228	-
Pattern lists (uBlock Origin)	318.084	-	585.475	43.824
TrackSign (Dirt 96%) 1st iteration	320.960	12.641.415	847.892	71.007
TrackSign (Dirt 96%) Graph convergence	330.068	13.603.663	858.771	73.641

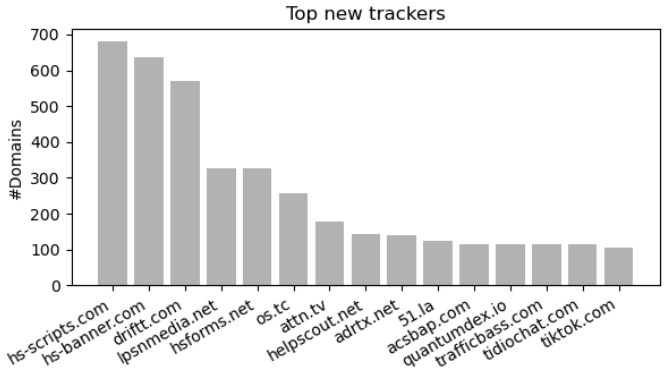

Fig. 8. Top new trackers detected

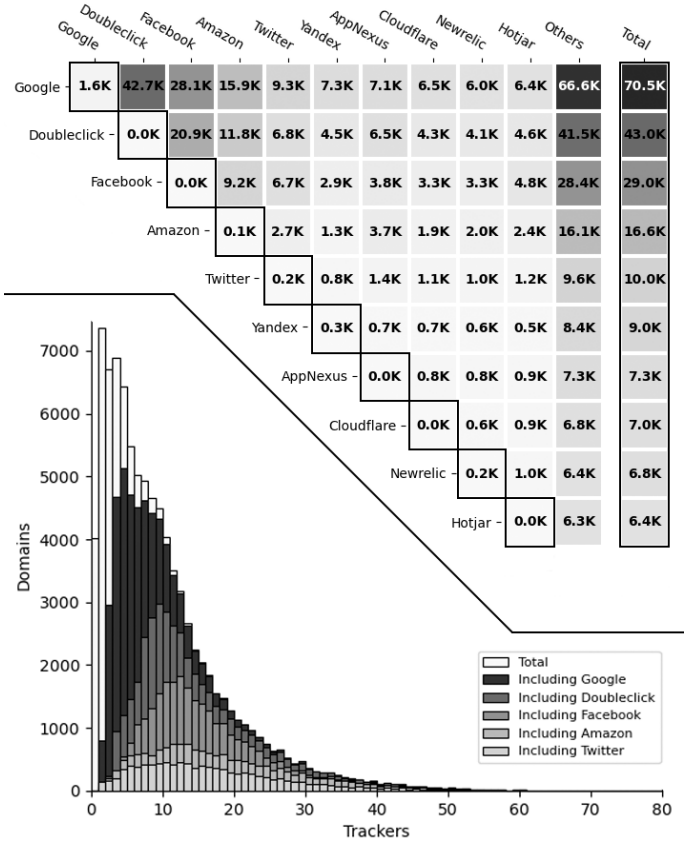

Fig. 9. **Top trackers heatmap (top)**: Cells represent the domains using simultaneously row and column trackers. The diagonal shows domains using only named tracker. Last column aggregates the total domains per tracker. **Trackers distribution (bottom)**: Distribution of websites by number of trackers used. Includes the top-5 trackers in colors.

fingerprint. To compute it we executed the iterative algorithm (Section III-D) until the graph converges. Figure 7 depicts the obtained results. Bars represent the number of new tracking resources detected for each selected *Dirt* threshold. Even if only fingerprints "dirtier" than 99% are considered, the method finds more than 4.000 new tracking resources. This number increased until 12.000 for a threshold of 96%. Starting at 95%, the number of resources rapidly increases up to more than 135.000 for a *Dirt* threshold value of 80%.

To verify the precision of the web tracking detection, we selected the subset containing the 50 most popular and 50 less popular new tracking resources detected for each threshold. The resulting subset (containing a total of 800 resources) was manually verified by an expert analyzing the code of each file individually looking for tracking patterns. The black line in Figure 7 shows the detection precision for the 800 manually verified resources. The figure indicates that the method obtains good precision (over 90%) for threshold values until 96%. For a *Dirt* level threshold of 95% the precision decreases to 79%. From this point on the precision keeps decreasing until stabilizing for thresholds about 80% and 85%.

Table I compares the results obtained by the pattern lists and TrackSign for a *Dirt* threshold of 96%. We selected this threshold to maximize the detection while maintaining a good precision. TrackSign results are split in two, one considering only the first iteration before the propagation of the *Dirt* level, and the other including the final results once the graph converges. The graph convergence iterative algorithm permitted us to detect 1M of new tracking fingerprints and almost 12.000 new tracking resources. Moreover, our system was able to detect almost 275K tracking URLs not detected by the lists. From these new URLs, 62% belong to already known trackers under previously unseen URLs, while the remaining 38% pertain to previously unknown trackers. Overall, our system was able to detect about 30.000 new web trackers.

Figure 8 shows the top new trackers detected by our method. Even if all trackers in the figure are currently present in more than 100 domains, they are not recognized at all by the existing blacklists. Among the newly detected trackers, we can find recently registered domains like *hs-banner.com*, *quantumdux.io* or *trafficbass.com* (registered on February and March 2020). *Driftt.com* is a new tracking domain owned by *drift.com*, the latter already blocked by EasyPrivacy. The same happens with *hsforms.net* and *hsforms.com* the second already included in EasyPrivacy. *51.la* also has patterns included in EasyPrivacy but only for the subdomains *js.51.la* and *ia.51.la*. TrackSign detected two new tracking subdomains (*js.users.51.la* and *images.51.la*). Other trackers like *hs-scripts.com*, belonging to *HubSpot*, *os.tc* from *One Signal* or *adrtx.net* belonging to *adality.de* are marketing tools using analytics scripts to track user behavior. *lpsnmedia.com*, *helpscout.net* and *tidiochat.com* are online chats also augmented with analytic files. Finally, we can find *tiktok.com*, an increasingly popular social network,

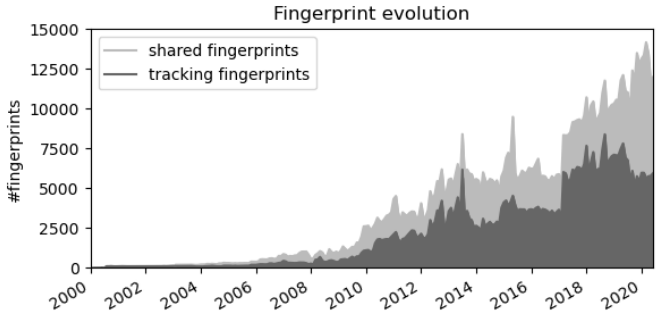

Fig. 10. Wayback Machine fingerprint evolution

especially in Asia.

Another interesting observation from Table I is that the number of unique URLs pointing to tracking resources is only about 16% of the total, but more than 35% of all JavaScript resources contain tracking code. This confirms the relevance of detecting web tracking at the resource level, instead of doing it at the URL level as existing content blockers based on blacklists. The picture at the resource level reveals that currently more than one third of the JavaScript resources present on the Internet include tracking algorithms. Overall, our system detected 73.641 trackers and 86.616 websites using at least one tracking system. According to those numbers more than 95% of websites include some sort of tracking algorithm (86.616 out of 90.637). However, our results show that only 92 trackers are present in more than 1% of websites (549 trackers in more than 0.1%), reinforcing the idea that only a few trackers will be accessed by the regular user on a daily basis.

The top chart of Figure 9 presents a heatmap with the most used web tracker combinations. Each cell represents the number of domains including tracking systems pertaining to *both*, the row and column trackers. Note that the same domain can be in more than one cell if the number of trackers used by the domain is higher than two. The diagonal are the number of domains including *only* the corresponding tracker. The last column aggregates the total number of domains using the tracker. We can see that *Google*, present in more than 70.000 websites (77%), continues to be the most pervasive tracker on the Internet. Note also that most websites include more than one tracker (the diagonal is almost empty). Not surprisingly, the most usual combinations include trackers from *Google*, *Doubleclick* (also belonging to *Google*), *Facebook*, *Amazon* and *Twitter*. All of them are present in more than 10% of the websites.

The bottom chart of Figure 9 shows the probability distribution of the number of trackers per domain. As already discussed, only 4.000 websites are tracker-free and about 7.000 websites load only one tracker. The average number of trackers per domain is 9.7, with a median of 7 trackers. The distribution also shows the number of websites loading one of the top-5 trackers. Surprisingly, websites loading only one or two tracking systems seem to use other trackers not in the top-5. Almost every website loading more than 3 trackers include at least one of *Google* or *Doubleclick*, highlighting once more their pervasiveness on the Internet.

TABLE II
MOST POPULAR SERVICES

Service	Tracking	Domains	URLs	First detection
Archive.org custom libraries	No	26	127	2000-09
Archive.org analytics	Yes	54	276	2001-04
JQuery library	No	64	465	2003-02
Flash player detection library	Yes	14	14	2007-09
Google analytics (library call)	Yes	24	52	2009-06
Google tracking	Yes	29	2578	2012-04
Facebook events (library call)	Yes	10	10	2015-06
Google Tag Manager (library call)	Yes	31	45	2016-08
Google analytics	Yes	14	14	2017-01
Youtube tracking	Yes	30	403	2017-03
Zoom.us	No	21	21	2020-03

D. Wayback Machine: detecting new tracking over time

As discussed in Section III-D, TrackSign can automatically detect new web tracking resources using similar methods to previously known tracking systems. However, the detection of completely new methods often requires some human intervention. This more manual process is based on the *popularity* level, which experts can tune to detect tracking systems at an early stage, before becoming widely used.

As the appearance of completely new tracking methods only happens occasionally, we needed a much larger ground-truth, spanning several months or even years, to evaluate this TrackSign feature. Thus, we decided to use the *Wayback Machine* [25], which is a digital archive that preserves snapshots of almost the entire World Wide Web over time. We modified ORM to scrape the 500 most popular websites from January 2000 to June 2020 from the Wayback Machine. After some preliminary analysis, we decided to set the popularity threshold in TrackSign to 10 domains, which represented a good trade-off between the amount of code to verify and our available human resources.

Figure 10 shows the evolution of the shared and tracking fingerprints over the last 20 years. We can observe a continuous increase of shared code across multiple websites. More than half of this code corresponds to web tracking, confirming our initial hypothesis that shared code can be indicative of the presence of tracking. The *popularity* level highlighted 735 "new" (i.e., new at that time) resources containing fingerprints shared by more than 10 domains. Table II presents the list of the most popular "new" services and "when" they were detected using TrackSign. After careful inspection, we discovered that the *Wayback Machine* dynamically replaces most commonly used libraries (including most tracking libraries, such Google Analytics) with its own version, probably as a space-saving measure. This is why many tracking practices were detected

TABLE III
TRACKING DETECTED

Method	Type	First detection	Last detection	Appearances
Storage-based	Cookies	2003-02-01	2020-06-01	270
Session-only	Session identifier	2003-02-01	2020-03-01	144
Cache-based	Web cache	2007-01-01	2020-06-01	170
Fingerprinting	Browser version	2007-01-01	2020-06-01	425
Fingerprinting	Browser plugins	2007-10-01	2020-01-01	51
Fingerprinting	Browser (other)	2007-10-01	2020-01-01	53
Fingerprinting	Device	2009-01-01	2020-03-01	26
Fingerprinting	Browser dimensions	2009-06-01	2020-03-01	108
Fingerprinting	Browsing history	2009-12-01	2020-03-01	34
Storage-based	Session storage	2009-12-01	2020-06-01	181
Fingerprinting	Browser capabilities	2010-07-01	2020-01-01	9
Storage-based	Local storage	2012-06-01	2020-06-01	224
Session-only	window.name DOM	2012-08-01	2015-12-01	2
Fingerprinting	WebGL capabilities	2015-05-01	2020-01-01	9
Fingerprinting	Browser language	2015-10-01	2019-10-01	21
Fingerprinting	Canvas	2016-09-01	2019-06-01	27

earlier with our system than their actual appearance (first two rows in Table II). For example, in the case of Google Analytics, our system reported new tracking as early as June 2009, but it could not determine it was actually Google Analytics until January 2017, when the Wayback machine decided to stop substituting Google Analytics calls by their own analytics libraries. This is an artifact of the Wayback machine that would not occur in reality when crawling the web directly with ORM.

Most detected services in Table II were actually tracking systems, while non-tracking services like JQuery are very easy to discard by an expert. Interestingly, the system was also able to detect an increasing number of websites linking to the videoconference service *Zoom.us* in March 2020, probably due to the social impact of the COVID-19 pandemic.

Table III shows a list of different tracking methods found within the suspicious files. We found more than 20 "new" tracking patterns, some of them very intrusive, such as fingerprinting methods. Most of these methods are still used today.

Despite the limitations of the Wayback Machine, our results demonstrate TrackSign's ability to focus on new code with high probability to belong to new tracking systems, which reduces significantly the time and resources required by experts to detect new web tracking systems.

VI. DISCUSSION AND FUTURE WORK

In this section, we discuss some limitations and additional possibilities of *TrackSign*. Some of the points introduced here are part of our future work.

1) ***Online blocking****:* Although the system has been designed to discover new web tracking offline, the output of *TrackSign* can be used to create a content-blocker with information about tracking resources. The plugin would compute the hash value for each downloaded file and compare it to a database with all the known resources performing tracking. In case of a match, the resource would be blocked. The main advantage of this approach is the robustness against web tracking systems constantly changing hosting domains, one of the main vulnerabilities of URL-based content-blockers. The main drawback, however, is that such blocking must be performed once the suspicious file has been downloaded, increasing the time and bandwidth required with respect to URL-based lists.

2) ***Enhancing blacklists****:* In comparison to existing blacklists, our results show a clear increase in the number of tracking URLs of about 46% (273K new URLs). This information could be used to improve the lists currently used by content-blockers, including those URLs discovered by *TrackSign*. This process could be easily automated with our system.

3) ***Crowd-sourcing****:* The *Popularity* level presented in this work introduces a delay between the appearance of a new service and its detection, as it needs to become popular before it can be detected. This delay can be mitigated at some extent by reducing the popularity threshold. However, for very small thresholds the number of popular fingerprints can increase significantly. A possible approach would be to rely on the force behind an active community, as in similar open-source projects, that could contribute to classify larger volumes of suspicious code than those we were able to analyze in Section V-D. This would probably increase even further the already huge (and disheartening) figure of tracking code in the Internet.

VII. CONCLUSIONS

In this work we presented *TrackSign*, a novel approach to discover new web tracking methods by means of code signatures. To this end, we propose a combination of a content-based file partition method, with a 3-mode network graph of the resources hosted online. Our algorithm then computes the popularity and the "dirtiness" of each signature to find web resources with a high probability of including tracking code. To evaluate our method, we developed ORM, a new open source framework specifically designed to explore the web and gather information about the resources therein.

Our results after analyzing the 100K most popular websites demonstrate the ability of our system to automatically detect new web tracking methods. The algorithm discovered 30.000 new trackers, some of them already present in more than 100 domains, but not detected by current pattern lists. The system also detected almost 12.000 new tracking resources and 275K new tracking URLs. We also observed an inexorable increase in the use of web tracking methods over the last 20 years using historical data from the Internet Archive. According to our results, at the time of this writing at least 95% of the websites include some sort of web tracking system. Moreover, we discovered that more than one third of all JavaScript resources hosted on the Internet include some sort of tracking code inside. The source code and data set collected for this study is publicly available at [24].

VIII. ACKNOWLEDGMENTS

This work was supported by the Spanish MINECO under contract TEC2017-90034-C2-1-R (ALLIANCE).

References

[1] B. Krishnamurthy, K. Naryshkin, and C. E. Wills, "Privacy leakage vs. Protection measures: the growing disconnect," in *Proceedings of the Web 2.0 Security & Privacy 2011*, p. 10, 2011.

[2] P. Leon, B. Ur, R. Shay, Y. Wang, R. Balebako, and L. Cranor, "Why Johnny can't opt out: a usability evaluation of tools to limit online behavioral advertising," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, (Austin, Texas, USA), pp. 589–598, Association for Computing Machinery, May 2012.

[3] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, USENIX Association, 2016.

[4] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: measurements, metrics, and implications," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, (Berlin, Germany), pp. 313–328, Association for Computing Machinery, Nov. 2011.

[5] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "A Survey on Web Tracking: Mechanisms, Implications, and Defenses," *Proceedings of the IEEE*, vol. 105, pp. 1476–1510, Aug. 2017.

[6] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," in *2012 IEEE Symposium on Security and Privacy*, pp. 413–427, May 2012.

[7] A. Hannak, G. Soeller, D. Lazer, A. Mislove, and C. Wilson, "Measuring Price Discrimination and Steering on E-commerce Web Sites," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, (Vancouver, BC, Canada), pp. 305–318, Association for Computing Machinery, Nov. 2014.

[8] J. Mikians, L. Gyarmati, V. Erramilli, and N. Laoutaris, "Crowd-assisted search for price discrimination in e-commerce: first results," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, CoNEXT '13, (Santa Barbara, California, USA), pp. 1–6, Association for Computing Machinery, Dec. 2013.

[9] K. Lobosco, "Facebook friends could change your credit score," Aug. 2013. https://money.cnn.com/2013/08/26/technology/social/facebook-credit-score/index.html.

[10] The Economist, "Very personal finance," 2012. https://www.economist.com/finance-and-economics/2012/06/02/very-personal-finance.

[11] Q. Wu, Q. Liu, Y. Zhang, P. Liu, and G. Wen, "A Machine Learning Approach for Detecting Third-Party Trackers on the Web," in *Computer Security – ESORICS 2016* (I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, eds.), Lecture Notes in Computer Science, (Cham), pp. 238–258, Springer International Publishing, 2016.

[12] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "ADGRAPH: A Graph-Based Approach to Ad and Tracker Blocking," *IEEE Symposium on Security and Privacy 2020*, p. 14, 2019.

[13] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "FPDetective: dusting the web for fingerprinters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, (Berlin, Germany), pp. 1129–1140, Association for Computing Machinery, Nov. 2013.

[14] V. Kalavri, J. Blackburn, M. Varvello, and K. Papagiannaki, "Like a Pack of Wolves: Community Structure of Web Trackers," in *Passive and Active Measurement* (T. Karagiannis and X. Dimitropoulos, eds.), Lecture Notes in Computer Science, (Cham), pp. 42–54, Springer International Publishing, 2016.

[15] H. Le, F. Fallace, and P. Barlet-Ros, "Towards accurate detection of obfuscated web tracking," in *2017 IEEE International Workshop on Measurement and Networking (M N)*, pp. 1–6, Sept. 2017.

[16] M. Ikram, H. J. Asghar, M. A. Kaafar, A. Mahanti, and B. Krishnamurthy, "Towards Seamless Tracking-Free Web: Improved Detection of Trackers via One-class Learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, pp. 79–99, Jan. 2017.

[17] T.-C. Li, H. Hang, M. Faloutsos, and P. Efstathopoulos, "TrackAdvisor: Taking Back Browsing Privacy from Third-Party Trackers," in *Passive and Active Measurement* (J. Mirkovic and Y. Liu, eds.), Lecture Notes in Computer Science, (Cham), pp. 277–289, Springer International Publishing, 2015.

[18] Z. Yu, S. Macbeth, K. Modi, and J. M. Pujol, "Tracking the Trackers," in *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, (Montréal, Québec, Canada), pp. 121–132, International World Wide Web Conferences Steering Committee, Apr. 2016.

[19] H. Metwalley, S. Traverso, and M. Mellia, "Unsupervised Detection of Web Trackers," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2015.

[20] D. Gugelmann, M. Happe, B. Ager, and V. Lenders, "An Automated Approach for Complementing Ad Blockers' Blacklists," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, pp. 282–298, June 2015.

[21] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (Vienna, Austria), pp. 1388–1401, Association for Computing Machinery, Oct. 2016.

[22] M. O. Rabin, "Fingerprinting by random polynomials," *Technical Report*, vol. Center for Research in Computing Technology, Harvard University, pp. 15–81, 1981.

[23] K. Cooper, "Alexa: Most popular website list," June 2020. https://www.alexa.com/.

[24] https://github.com/CBA-UPC/ORM, "Online Resource Mapper (ORM)," June 2020.

[25] archive.org, "Wayback Machine," 2020. https://web.archive.org/.

[26] AdBlock Plus, "Adblock Plus," Feb. 2020. https://adblockplus.org/en/.

[27] Ghostery, "Ghostery Makes the Web Cleaner, Faster and Safer!," Feb. 2020. https://www.ghostery.com/.

[28] R. Hill, "uBlock Origin," Feb. 2020. https://github.com/gorhill/uBlock.

[29] Disconnect, "Disconnect | Take back your privacy," 2020. http://disconnect.me.

[30] "EasyList," June 2020. https://easylist.to/.

[31] "EasyPrivacy," June 2020. https://easylist.to/easylist/easyprivacy.txt.

[32] M. Lécuyer, G. Ducoffe, F. Lan, A. Papancea, T. Petsios, R. Spahn, A. Chaintreau, and R. Geambasu, "XRay: Enhancing the Web's Transparency with Differential Correlation," in *23th USENIX Security Symposium*, pp. 49–64, 2014.

[33] B. Li, P. Vadrevu, K. H. Lee, and R. Perdisci, "JSgraph: Enabling Reconstruction of Web Attacks via Efficient Tracking of Live In-Browser JavaScript Executions," in *Proceedings 2018 Network and Distributed System Security Symposium*, (San Diego, CA), Internet Society, 2018.

[34] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, (Banff, Alberta, Canada), pp. 174–187, Association for Computing Machinery, Oct. 2001.

[35] Jason Huggins, "SeleniumHQ Browser Automation," Feb. 2020. https://www.selenium.dev/.

[36] J. Mazel, R. Garnier, and K. Fukuda, "A comparison of web privacy protection techniques," *Computer Communications*, vol. 144, pp. 162–174, Aug. 2019.

[37] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl, "Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 319–333, Apr. 2017.

[38] L. N. Lielmanis, Einar, "jsbeautifier: JavaScript unobfuscator and beautifier.." https://beautifier.io.