A Machine Learning Approach for Detecting Third-Party Trackers on the Web

Qianru Wu[1](✉), Qixu Liu[2], Yuqing Zhang[1], Peng Liu[3], and Guanxing Wen[4]

[1] National Computer Network Intrusion Protection Center, University of Chinese Academy of Science, Beijing, China
wuqianru11@mails.ucas.ac.cn, zhangyq@ucas.ac.cn

[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
liuqixu@iie.ac.cn

[3] College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA

[4] Team Pangu, Shanghai, China

Abstract. Nowadays, privacy violation caused by third-party tracking has become a serious problem and yet the most effective method to defend against third-party tracking is based on blacklists. Such method highly depends on the quality of the blacklist database, whose records need to be updated frequently. However, most records are curated manually and very difficult to maintain. To efficiently generate blacklists, we propose a system with high accuracy, named DMTrackerDetector, to detect third-party trackers automatically. Existing methods to detect online tracking have two shortcomings. Firstly, they treat first-party tracking and third-party tracking the same. Secondly, they always focus on a certain way of tracking and can only detect limited trackers. Since anti-tracking technology based on blacklists highly depends on the coverage of the blacklist database, these methods cannot generate high-quality blacklists. To solve these problems, we firstly use the structural hole theory to preserve first-party trackers, and only detect third-party trackers based on supervised machine learning by exploiting the fact that trackers and non-trackers always call different JavaScript APIs for different purposes. The results show that 97.8 % of the third-party trackers in our test set can be correctly detected. The blacklist generated by our system not only covers almost all records in the Ghostery list (one of the most popular anti-tracking tools), but also detects 35 unrevealed trackers.

1 Introduction

A website page (called first-party website) always embeds many unrelated websites belonging to different administrative entities (called third-party website) in the form of JavaScript, iframe, images or flash to gain functionality (such as advertisement, web analytic and social network). We call the websites who identify and collect private information (such as browser history) about users as trackers. First-party websites and third-party websites may both track users for different purposes. A first-party tracker usually tracks users for antifraud or

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part I, LNCS 9878, pp. 238–258, 2016.
DOI: 10.1007/978-3-319-45744-4_12

paywalls [1]. If the web tracking was blocked, this first-party website may not work right or pose a threat to users' security. A third-party tracker can stealthily collect users' web browsing history for purposes such as targeted advertising or predicting trends, which generates enormous benefits at the expense of users' privacy [2].

Nowadays, privacy violation caused by third-party tracking has become a serious problem, and a considerable amount of effort has been made to protect users' privacy against online tracking. Anti-tracking technology based on blacklists is most effective [2–4]. Many commercial anti-tracking tools (Adblock [5], DoNotTrackMe [6], Ghostery [7]) are based on blacklists. They generate blacklists offline and block requests to the URLs in the blacklist online. This method highly depends on the records in the blacklist whereas a tracking company can adopt new domains to track users [8]. These known blacklists need to be updated regularly. However, these blacklists are usually manually curated and difficult to maintain.

To efficiently generate blacklists, several approaches have been proposed to detect trackers automatically [9–11]. However, these solutions are grossly inadequate. Firstly, existing methods to detect online tracking treat first-party trackers and third-party trackers the same. Secondly, they always focus on a certain way of tracking and can only detect limited trackers. Since anti-tracking technology based on blacklists highly depends on the coverage of the blacklist database, these methods cannot generate high-quality blacklists.

In this paper, we propose an efficient and adaptive system with high accuracy, named DMTrackerDetector, to detect third-party trackers while preserving first-party trackers, which can make it easier to generate a blacklist and reduce human work. Firstly, since a first-party file only exists in this website while a third-party file exists in many websites, we use structural hole theory, which is always used in social networks to find the 'tie' among several communities, to filter out first-party trackers. Secondly, instead of focussing on a certain way of tracking, DMTrackerDetector detects third-party trackers based on supervised machine learning by exploiting the fact that trackers and non-trackers always call different JavaScript APIs for different purposes. To exploit this fact, DMTrackerDetector takes all JavaScript APIs features to build a classifier. DMTrackerDetector can automatically generate the blacklist of third-party trackers based on the structural holes and the classifier.

The contributions of this paper can be summarised as follows:

1. We distinguish first-party trackers and third-party trackers in a straightforward and effective way based on structural hole theory.
2. We propose an adaptive method to detect all JavaScript-based tracking technologies based on supervised machine learning.
3. We provide not only the effective features to distinguish trackers and non-trackers but also the effective way to extract these features.
4. We evaluate the effectiveness of our system, and 97.8 % of the third-party JavaScript are classified correctly. We also compare our system with Ghostery (one of the most popular anti-tracking tools), and the results show that the

list generated by DMTrackerDetector not only covers almost all JavaScript-based trackers records in the Ghostery list, but also contains more trackers than Ghostery.

5. We make a detailed analysis about the correlation of JavaScript APIs which can help us better understand trackers. We distill the top 20 JavaScript APIs most correlated to the fact that a JavaScript file is a tracker and the top 20 JavaScript APIs most uncorrelated to this fact.

The rest of this paper is organised as follows. First, we introduce the background and related work in Sect. 2. Section 3 is the detailed description of the design and implementation of DMTrackerDetector. The experiment evaluation results are presented in Sect. 4. Next, we discuss the limitation of the proposed system in Sect. 5. Finally we conclude this paper in Sect. 6.

2 Background and Related Work

2.1 Background

Web tracking technologies can be divided into stateful tracking and stateless tracking according to whether or not there is a dependence on client-side information storage [4]. The most commonly used stateful tracking technique is HTTP cookies, which can store limited bytes and be deleted easily. Later on, Flash cookies and LocalStorage are used because of their larger storage and better concealment compared to HTTP Cookies [11–13]. However, stateful tracking can be deleted by users as they store on the client side, which motivates trackers to find new ways to link users to their browsing histories. Trackers learn properties about the browser that, taken together, form a unique or nearly unique identifier, which is called stateless (fingerprinting) tracking [4,9,14].

JavaScript is used mainly to dynamically manipulate a page's DOM, control the browser, and communicate asynchronously. In this paper, we refer to all JavaScript objects, properties and methods provided by browsers as JavaScript APIs. Most JavaScript-based behaviours are non-tracker behaviours, such as loading new page content, submitting data to the server without reloading the page, animation of page elements, interactive content, validating input values of a web form to make sure that they are acceptable before being submitted to the server, and so on.

JavaScript also plays an important role in web tracking, and we focus on detecting JavaScript-based third-party trackers in this paper. JavaScript can implement most tracking behaviours:

Stateful Tracking. JavaScript can set, read, modify and remove HTTP Cookies, LocalStorage and SessionStorage by calling APIs. What is more, trackers may pass pseudonymous IDs associated with a given user, typically stored in cookies, amongst each other via JavaScript execution in order to better facilitate targeting and real-time bidding [15].

Stateless Tracking. The privileged position inside the browser makes JavaScript a strong fingerprinting tool, which can access browser resources [11].

Information about the browser vendor, supported plugins, MIME types, operating system, display settings and installed fonts can all be gained by calling JavaScript APIs.

2.2 Related Work

Existing Anti-tracking Mechanisms. Although web tracking has garnered much attention, no effective defense system has been proposed. Roesner et al. [16] proposed a tool called ShareMeNot, but it can only defend against social media button tracking, a small subset of tracking practices. Disabling script execution [17] provides protection at the cost of pages failing to open or render properly [18]. Private browsing mode significantly affects user experience as users cannot consistently save anything on the client-side state and users can still be tracked before closing the browser. The Do Not Track (DNT) header and legislation require tracker compliance and cannot effectively protect users from tracking in reality [4,16]. Opting out of cookies [19,20] and disabling third-party cookies can be easily bypassed through non-cookie-based tracking approaches [12,13]. Moreover, as trackers can make regular information available in a typical HTTP request, disabling HTTP cookies or flash cookies [21] is useless. Rather, the most effective method to defend against third-party tracking is based on blacklists, and most commercial anti-tracking tools [2–4] are based on blacklists.

Existing Non-machine Learning-based Tracking Detection Mechanisms. Existing non-machine learning-based focussed on a certain way of tracking and cannot be used to generate blacklists alone. Nikiforakis et al. [9] studied three previously known fingerprinting companies and found 40 such sites among the top 10 K sites employing practices such as font probing and the use of Flash to circumvent proxy servers. Acar et al. [10] used behavioural analysis to detect fingerprinting scripts that employ font probing and found that 404 sites in the top million deployed JavaScript-based fingerprinting. In the followup study, Acar et al. [11] proposed three heuristics to estimate whether a canvas is being used to fingerprint.

Existing Machine Learning-based Tracking Detection Mechanisms. To efficiently generate blacklists, several machine learning-based approaches have been proposed to detect trackers automatically. Most of these approaches are used to detect advertisement-related web tracking since web tracking is usually used in advertising.

Kushmerick et al. [22] first suggested using machine learning to block online advertisements, utilising the C4.5 classification scheme to build an advertisement image blocker called AdEater, which only blocked advertisement images of static pages while many advertisements on the current Web are loaded dynamically via JavaScript or as flash objects. Orr et al. [23] trained a classifier for detecting advertisements loaded via JavaScript code with the features extracted through static program analysis. In this study, they manually labelled the advertisement-related JavaScript code and other JavaScript code of 339 websites by visiting each website and using the Firebug extension. Bhagavatula et al. [24] presented a technique for detecting advertisement resources utilising the k-nearest neighbours classification based on EasyList, which is the primary subscription of Adblock aimed at removing advertisements from web pages. Different from Orr, their basic idea was using the classification criteria of an older version of EasyList to train a classifier to accurately identify advertisements according to a much newer blacklist.

However, the machine learning-based approaches discussed above only focussed on detecting advertisement-related contents, including loading advertisement content (image or flash), which is not considered to be tracking behaviour in some studies because no privacy information is leaked in this situation.

We only focus on detecting tracking behaviours in this paper. Yamada et al. [25] had the same goal as us, and they proposed web tracker detection and blacklist generation based on a temporal link analysis. Their system classified suspicious sites by using machine-learning algorithms, and only 62 %–73 % blacklisted sites were detected. In our former work [2], we trained an incremental classifier to detect third-party trackers through static JavaScript analysis, and 93 % trackers in the test set can be classified correctly. However, that method had some shortcomings compared to this paper: (1) we made no distinction between first-party trackers and third-party trackers in the former work; (2) code obfuscation makes feature extraction hard through static analysis, which will introduce errors; (3) it is easy to bypass the detection by adding some useless JavaScript APIs. We have solved all these problems in this paper.

3 Design and Implementation

3.1 Basic Idea and System Overview

First-party websites and third-party websites may both track users by executing JavaScript, while our goal is to block third-party tracking while preserving first-party tracking. Therefore, we should filter out first-party files before any other action. The most intuitive and effective way to determine whether a file belongs to the first-party website or a third-party website is based on the file location. If the file is located in the first-party server, we consider it to be a first-party file. However, a first-party website may choose to cache files downloaded from third-party websites on its own server for performance or security issues. According to its location, we consider this kind of file to be first-party files in this paper.

Third-party trackers track users in two steps: (1) obtain the users' information and (2) generate an HTTP request with users' information to third-party servers. Take the third-party tracker google-analytics.com for example, as shown in Fig. 1. Case 1 and Case 2 demonstrate how it tracks users. The JavaScript code ga.js is acquired from google-analytics.com when a user visits the first-party website a.com (**Case 1**) or is downloaded from google-analytics.com and cached in first-party website b.com (**Case 2**). Then, an HTTP request with users' information, which is generated by executing the JavaScript code ga.js, is sent to the third-party server google-analytics.com.

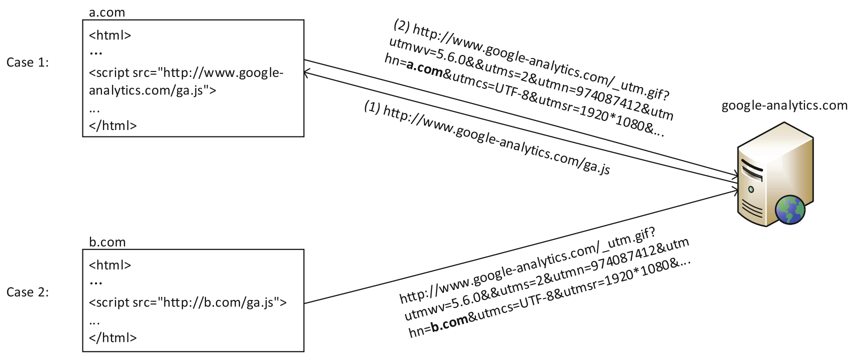

Case 1:
a.com
<html>
...
<script src="http://www.google-analytics.com/ga.js">
...
</html>
(2) http://www.google-analytics.com/_utm.gif?utmwv=5.6.0&&utms=2&utmn=974087412&utmhn=a.com&utmcs=UTF-8&utmsr=1920*1080&...
(1) http://www.google-analytics.com/ga.js
google-analytics.com
Case 2:
b.com
<html>
...
<script src="http://b.com/ga.js">
...
</html>
http://www.google-analytics.com/_utm.gif?utmwv=5.6.0&&utms=2&utmn=974087412&utmhn=b.com&utmcs=UTF-8&utmsr=1920*1080&...

Fig. 1. Third-party tracking examples

The JavaScript ga.js is a tracking JavaScript file and the HTTP request _utm.gif is generated by executing ga.js. In case 1, we can block third-party tracking by blocking the tracking JavaScript (ga.js) or its generated request with users' information (_utm.gif). In case 2, since we remove all first-party JavaScript at first, it cannot block third-party tracking by blocking the tracking JavaScript. Therefore, we can block third-party tracking by blocking the HTTP request with users' information generated by executing the tracking JavaScript in case 2.

Different behaviours lead to different API sets being called, so JavaScript-based trackers and non-trackers call different API sets because of different purposes. Based on these facts, we can get the tracking JavaScript through machine learning. If the tracking JavaScript was blocked, its generated request would not be generated. To get the generated HTTP requests, we can compare the HTTP requests crawling when no JavaScript is blocked with the HTTP requests crawling when the tracking JavaScript is blocked. At last, we add both the tracking JavaScript and its generated HTTP request to the blacklist.

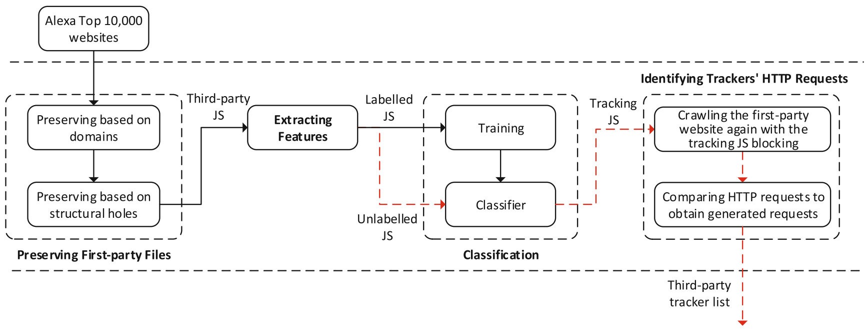

Alexa Top 10,000 websites
Preserving based on domains
Preserving based on structural holes
Preserving First-party Files
Third-party JS
Extracting Features
Labelled JS
Unlabelled JS
Training
Classifier
Classification
Tracking JS
Identifying Trackers' HTTP Requests
Crawling the first-party website again with the tracking JS blocking
Comparing HTTP requests to obtain generated requests
Third-party tracker list

Fig. 2. System overview

As shown in Fig. 2, at a high level, the process includes four parts: preserving first-party files, extracting features, classification and identifying trackers' HTTP requests. We crawled the homepages of the Alexa top 10,000 websites. Firstly, we filter out all first-party files and only focus on third-party files. Secondly, we extracted features of third-party JavaScript files via Chromium browser with hooked JavaScript interface. Then, we labelled some JavaScript instances and built a classifier with the labelled third-party JavaScript instances in the classification part. With the classifier, the blacklist can be automatically generated as follows:

1. Unlabelled third-party JavaScript instances are classified by using the classifier to get third-party tracking JavaScript.
2. Crawl these first-party websites again by blocking the third-party tracking JavaScript. Compare the HTTP requests crawling when the tracking JavaScript is blocked with the HTTP requests crawling when no JavaScript is blocked in order to get the requests generated by the tracking JavaScript.
3. Add third-party tracking JavaScript and its generated requests to the blacklist.

3.2 Preserving First-Party Files

As introduced in Sect. 3.1, we consider a file to be a first-party file if it is located in the first-party server. First-party files and third-party files have one different feature: a first-party file only exists in this website, while a third-party file exists in many websites. Especially, if the url of a file has the same domain as the first-party website', it is a first-party file.

Therefore, we automatically preserved first-party files in two steps. Firstly, if a file has the same domain as the first-party website, it is considered to be a first-party file. This step is easy to determine by checking domain names. Secondly, if a file only exists in one first-party website, it is considered to be a first-party file. This step can be determined based on structural hole theory: if a file is not a structural hole of the relation graph, it is considered to be a first-party file. First-party files will be preserved.

Structural hole theory is always used in social networks to find the 'tie' among several communities. If a first-party website and all its HTTP requests can be regarded as a community, then a third-party file should be a structural hole since it exists in many first-party websites.

To be specific, we built a relation graph which consists of nodes and directed edges. All HTTP request URLs consist of nodes. If HTTP request b is sent when visiting the first-party website A, there is a directed edge from A to b as shown in Fig. 3. A first-party website (the big nodes) and all its HTTP requests (the small nodes) form a community. A first-party file (the small and white nodes) only exists in the first-party website. A third-party file (the black nodes) exists in several communities, and it becomes the tie between them. The structural holes in the relation graph have larger in-degree than other nodes, so we find structural holes via in-degree of the nodes.

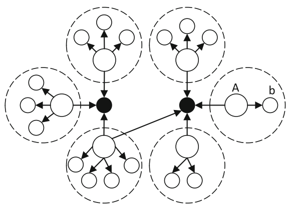


Fig. 3. The relation graph of websites

3.3 Feature Extraction

We have proved that it can successfully classify trackers and non-trackers by using JavaScript API sets [2]. In our former work, we extracted features via static analysis. However, code obfuscation, which is a common technique for JavaScript aimed at making the code difficult to read so that it can protect the code from theft and reuse, makes it hard to extract APIs precisely through static analysis. Moreover, it is easy for trackers to add some useless features to bypass the detection when extracting features based on static analysis.

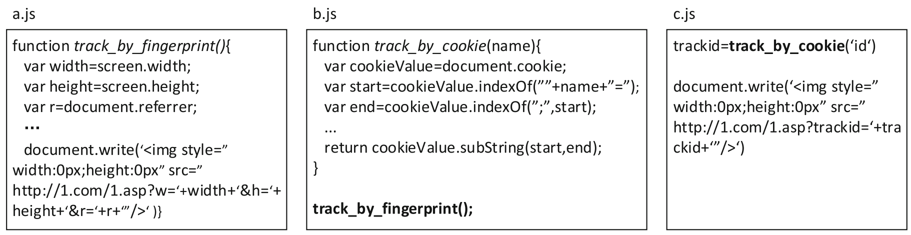
Therefore, we extracted APIs through dynamic analysis. We modified parts of the WebKit source code, which is the rendering engine used by Chromium, to intercept and log the JavaScript APIs a JavaScript file invokes when executing. We preferred to work at the native code level instead of developing browser extensions or JavaScript patches for several reasons: to detect the origin of events more precisely and to defend against JavaScript attacks that block or circumvent extensions and getter methods[1].

We try to hook as many APIs as possible when modifying the WebKit source code, and a JavaScript file is encoded as a 505-dimensional binary vector through feature extraction: if an API is invoked by the JavaScript file it is set to be 1, or it is set to be 0. For example, as shown in Fig. 4, a.js, b.js and c.js are embedded in the same page. The JavaScript file b.js invokes the function track_by_fingerprint written in a.js, and c.js invokes itself and the function track_by_cookie written in b.js. Thus, a.js invokes no APIs; b.js invokes screen.width, screen.height, document.referrer, document.write etc.; and c.js invokes document.cookie, document.write etc. Then a.js is encoded as a 505-dimensional 0 vector, b.js is encoded as a 505-dimensional binary vector whose features screen.width, screen.height, document.referrer and document.wirte are set to be 1, and c.js is encoded as a 505-dimensional binary vector whose features document.cookie and document.write are set to be 1.

3.4 Classification

The training component has two subcomponents: labelling the dataset and training the classifier.

[1] http://code.google.com/p/chromium/issues/detail?id=55084.



a.js

```
function track_by_fingerprint(){
  var width=screen.width;
  var height=screen.height;
  var r=document.referrer;
  …
  document.write('<img style="
width:0px;height:0px" src="
http://1.com/1.asp?w='+width+'&h='+
height+'&r='+r+'"/>' )}
```

b.js

```
function track_by_cookie(name){
  var cookieValue=document.cookie;
  var start=cookieValue.indexOf(""+name+"=");
  var end=cookieValue.indexOf(";",start);
  ...
  return cookieValue.subString(start,end);
}

track_by_fingerprint();
```

c.js

```
trackid=track_by_cookie('id')

document.write('<img style="
width:0px;height:0px" src="
http://1.com/1.asp?trackid='+tra
ckid+'"/>')
```

Fig. 4. The example code of function invocation

Labelling Dataset. Before we trained the classifier, we had to label the training set. According to the JavaScript invocation, we call the JavaScript which provides functions and is invoked by other JavaScript as *JavaScript Library*. As opposed to the JavaScript library, we call the JavaScript which invokes a JavaScript library or invokes itself as *JavaScript caller*. A JavaScript file may be both a library and a caller according to different invocation relation. For example, as shown in Fig. 4, b.js invokes the function written in a.js, so b.js is a JavaScript caller to a.js and a.js is a JavaScript library to b.js. C.js invokes the function written in b.js, so c.js is JavaScript caller to b.js and b.js is JavaScript library to c.js.

We only labelled tracking callers as tracking JavaScript. Because blocking tracking libraries or tracking callers can both block tracking, while tracking behaviours happen in tracking callers and through feature extraction a tracking library may be encoded as 0 vector which cannot be used to train the classifier. As shown in Fig. 4, a.js should be labelled as non-tracker since a.js only provides function and no features are extracted from a.js, b.js should be labelled as tracker since b.js invokes function track_by_fingerprint to track users by fingerprinting, and c.js should be labelled as tracker since c.js invokes function track_by_cookie and itself to track users via HTTP cookies.

Easy List[2], Easy Privacy[3] and Ghostery [7] are the most effective blacklists [26]. Unfortunately, third-party trackers and non-trackers cannot be labelled simply based on these lists, because these blacklists label third-party trackers without distinguishing tracking libraries and tracking callers, while we could not label tracking libraries as trackers when training the classifier. Therefore, to train the classifier, we labelled the training set firstly by using Ghostery, Easy List and Easy Privacy separately, then manually confirmed the JavaScript files determined as trackers by these lists and relabelled the JavaScript files determined as non-trackers by these lists according to our former experience [2].

Most obfuscation used by normal websites cannot conceal JavaScript APIs. We can understand the code with the help of deobfuscated tools. However, it is possible that some JavaScript code may be highly obfuscated, and we did not label the highly obfuscated JavaScript files because we could not understand them.

[2] https://easylist-downloads.adblockplus.org/easylist.txt.
[3] https://easylist-downloads.adblockplus.org/easyprivacy.txt.

Training Classifier. We trained the classifier by using the labelled instances. To determine what classification scheme fit our data best, we implemented several of the most common classifiers used for supervised learning with the help of WEKA [27], a statistical software package. We will discuss the choice of the classifiers in Sect. 4.3.

4 Evaluation

The experiment was conducted on a machine with 8-GB memory and Inter (R) Core (TM) 2 Quad 2.93-GHz processor. The classification was implemented with WEKA [27], a statistical software package. Other parts of the system are programmed in Python language.

Firstly, we evaluated the effectiveness of every single part of our system. Then we evaluated the overall effectiveness of DMTrackerDetector and compared the efficiency of DMTrackerDetector and Ghostery. For our experiments, we crawled the home pages of the Alexa top 10,000 websites. We built the relation graph used to preserve first-party files based on all HTTP requests of the crawled websites. We randomly selected 500 websites and spent less than 3 weeks (one person) to manually label all third-party JavaScript files to train the classifier, turning out 1,237 unique third-party non-tracker instances and 1,199 unique third-party tracker instances. We randomly selected another 100 websites to test the effectiveness of the classifier.

We did not use the dataset in our former work because at that time we labelled instances mainly based on Ghostery, whereas Ghostery labels third-party trackers without distinguishing tracking libraries and tracking callers. As introduced in Sect. 3.4, in this paper we could not label tracking libraries as trackers when training the classifier, so we had to label the training set manually. The dataset in our former work was too large to manually label. However, our training set in this paper is large enough to build a strong classifier, since high accuracy is obtained with the test set.

4.1 The Results of Preserving First-Party Files

As noted in Sect. 3.2, we preserved first-party files in two steps. Firstly, if a file has the same domain as the first-party website, it is considered to be a first-party file. This step is easy to determine by checking domain names.

Secondly, if a file only exists in one first-party website, it is considered to be a first-party file. This step can be determined based on structural hole theory. We made a relation graph based on all HTTP request URLs of the crawled 10,000 websites. Many files have fundamentally the same content but different URLs, such as http://s7.addthis.com/js/250/addthis_widget.js and http://s7.addthis.com/js/300/addthis_widget.js. Therefore, instead of URLs, we took domains as nodes. We chose the nodes whose in-degree is larger than 2 as structural holes (third-party domains) because: (1) Many websites have two domain names, e.g. renren.com and xiaonei.com are the same website; (2) A tracker existing in two websites tracks users only across the two first-party websites, and it will not bring severe privacy threat. The relation graph consists of 15,892 nodes. Among these, 1,999 nodes have larger than 2 in-degrees. So the 1,999 nodes are considered to be third-party domains. The domain google-analytics.com is the structural hole with the largest in-degrees of 4,317.

We tested the quality of the structural holes by manually checking nodes in the relation graph whose in-degree was 3. We determined a domain to be a third-party domain through search engine, whois and visiting websites directly. If a domain appears in more than one website under the control of different administrative entities, it is a third-party domain. There are 544 domains whose in-degree is 3 in our dataset, and 522 (95.96 %) of them are third-party domains. The errors occur in this situation: when an administrative entity has many domains, all domains belonging to the administrative entity may be considered to be third-party domains. For example, the domains aliexpress.com, alibaba.com and aliimg.com belong to the same administrative entity and interact with each other, so they will be considered to be third-party domains.

We do not test the quality of the structural holes whose in-degree is larger than 3, because domains with larger in-degrees are more likely to be third-party domains and we only need to check the worst situation. We can raise the quality of the structural holes by choosing nodes whose in-degree is larger than 3 as third-party domains, since a tracker existing in three websites tracks users only across the three first-party websites and it still will not bring severe privacy threats. However, we chose the nodes whose in-degree is larger than 2 as structural holes at last, because the quality of the structural holes whose in-degree is 3 is acceptable.

4.2 The Effectiveness of Features

To determine the effectiveness of features we used in the classification process, we evaluated which features contributed more to the classification at first by using χ^2 algorithm. The χ^2 of a feature represents the degree of correlation with classification. The feature with larger χ^2 can be considered that it contributes more to the classification. Table 1 lists the top 40 features ranked by χ^2.

Then, we looked at the correlation of the features in greater detail. We wanted to know which features are correlated to being a tracker. The correlation analysis we have adopted is based on the value of the Spearman's correlation coefficient. As shown in Fig. 5, after calculating the Spearman's correlation coefficient, we list the top 20 features which are the most correlated to the fact that a JavaScript file is a tracker, and the top 20 features which are the most uncorrelated to that fact. As shown in Table 1, the bold features are positively correlated to being a non-tracker. Not surprisingly, we can infer the following conclusions from Table 1 and Fig. 5:

1. According to the correlation of the features, trackers are chiefly concerned with the operations of getting information (such as screen information, location information, navigator information, referrer, plugin information, etc.)

Table 1. Top 40 features of the classification ranked by χ^2

Rank	Feature	χ^2	Rank	Feature	χ^2
1	Document::cookie	851.29	21	DOMPluginArray::length	74.25
2	Document::referrer	631.60	22	Navigator::javaEnabled	69.99
3	Document::setCookie	624.74	23	DOMPlugin::name	69.99
4	Screen::height	286.38	24	DOMPluginArray::pluginData	69.31
5	Screen::width	279.72	25	Document::domain	65.98
6	Location::url	274.24	26	Location::host	59.40
7	Location::hostname	234.70	27	**Document::createDocumentFragment**	57.47
8	Storage::getItem	231.44	28	**HTMLInputElement::value**	52.73
9	Document::title	140.92	29	**HTMLInputElement::maxLength**	52.51
10	Screen::colorDepth	138.28	30	**HTMLInputElement::setValue**	51.98
11	Location::protocol	128.56	31	Document::webkitHidden	51.71
12	Location::href	125.83	32	**HTMLElement::setInnerText**	50.60
13	Storage::setItem	124.46	33	DOMWindow::innerWidth	45.71
14	Navigator::language	117.30	34	**Element::hasAttribute**	42.61
15	Location::search	103.28	35	**Node::getElementsByName**	42.41
16	Navigator::cookieEnabled	100.76	36	**Document::defaultView**	41.81
17	HTMLImageElement::setHeight	98.92	37	Navigator::vendor	38.45
18	HTMLImageElement::setWidth	98.92	38	**Document::createComment**	36.69
19	DOMPluginArray::item	78.99	39	**Element::webkitMatchesSelector**	34.09
20	Location::hash	77.11	40	**HTMLTextAreaElement::value**	33.80

and manipulating HTTP cookies and LocalStorage (such as invoking Document::cookie, Document::setCookie, Document::domain, Storage::getItem, Storage::setItem, etc.) for the main goal of trackers is to get and record users' privacy information.

2. Stateful tracking may be more widely used than stateless tracking, because the APIs for HTTP cookies (such as Document::cookie, Document::setCookie, Location::url, etc.) have larger Spearman's coefficient value than the APIs for fingerprinting (such as class Screen, Navigator, DOMPlugin, DOMPluginArray, etc.).
3. Non-trackers tend to be used for the operations of HTML element (such as class HTMLElement, HTMLInputElement, Element, etc.) and DOM nodes (such as class Node) as the main goal of non-trackers is to enrich the user experiences.
4. As shown in Table 1, the features which are the most correlated to the fact that a JavaScript file is a tracker have larger χ^2 than the features which are the most correlated to the fact that a JavaScript file is a non-tracker. Therefore, the behaviours of trackers play a more important role than non-trackers in the classification.

4.3 The Classifier Results

As introduced in Sect. 3.3, a JavaScript file may be encoded as 0 vector. The 0 vector instances were considered to be non-trackers and we removed all 0 vector

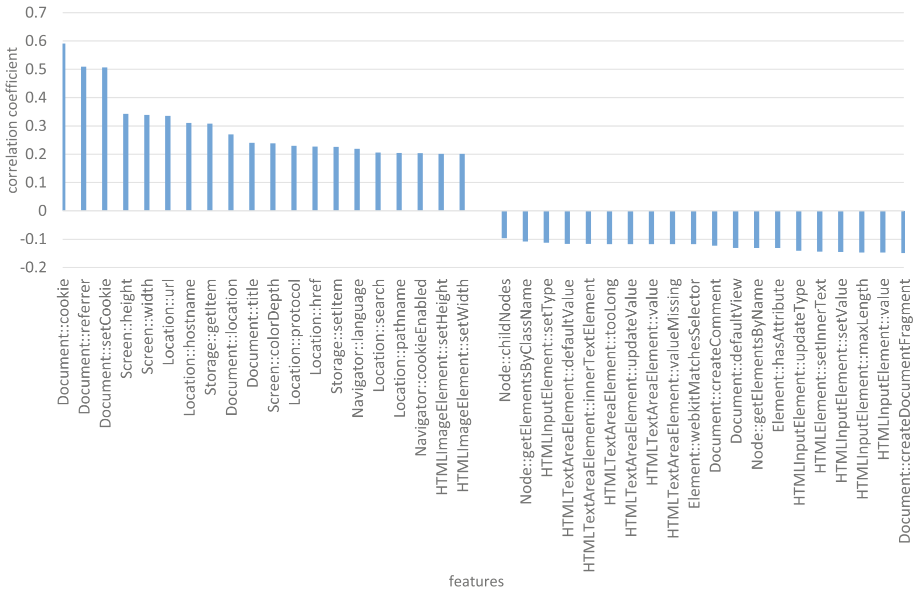


Fig. 5. Spearman's Correlation Coefficient between features and being a tracker in the classification

instances when training the classifier. The classifier is trained by using 2,436 third-party non-zero instances in 500 websites, which consists of 1,199 unique trackers and 1,237 unique non-trackers.

We employed Naive Bayes, Logistic Regression, SMO, Id3, ADTree, J48 and Random Forest classification models. Table 2 shows a comparison of the accuracy measures among different classifiers. Even though Id3 and Random Forest perform well in the case of the training set, both of them have low accuracy in the 10-fold cross validation. J48 shows high accuracy in the case of the training set and the best accuracy values in the 10-fold cross validation. As a result, we selected the J48 classification model to train our classifier.

Table 3 shows the results of our classifier obtained using the training set, and Table 4 shows the results of our classifier in 10-fold cross validation. To evaluate the classifier in real scenarios, we also manually labelled a test set. We randomly selected 100 websites and manually labelled all third-party JavaScript files, turning out 314 non-zero non-tracking instances and 388 non-zero tracking instances. Table 5 lists the confusion matrix on the test set, and 97.8 % instances are classified correctly.

4.4 The Results of Identifying Trackers' HTTP Requests

We compared the HTTP requests crawling when no JavaScript is blocked with the HTTP requests crawling when the tracking JavaScript is blocked in order to get the missing requests. In addition to the generated requests, the missing

Table 2. Comparison of classifier models

Classifier models	Accuracy obtained using training set(%)	Accuracy in 10-fold cross validation(%)
NaiveBayes	85.8	84.98
ADTree	93.56	93.06
Logistic	97.33	93.1
SMO	96.51	94.17
RandomForest	99.92	95.32
Id3	99.92	96.02
J48	98.72	96.31

Table 3. Results obtained using the training set

Accuracy(%)	TP rate	FP rate	Class
98.72	0.988	0.013	Non-tracker
	0.987	0.012	Tracker

Table 4. Results of the 10-fold cross validation

Accuracy(%)	TP rate	FP rate	Class
96.31	0.959	0.033	Non-tracker
	0.967	0.041	Tracker

Table 5. Confusion matrix on test set of the classifier

Prediction / Truth	Non-tracker	Tracker
Non-tracker	311	3
Tracker	12	376

requests may contain HTTP requests for files (such as images, css, JavaScript files, etc.), which are randomly generated and different each time the website is visited. The HTTP requests generated by the third-party tracking JavaScript are always sent to third-party servers with some parameters in the URLs. Therefore, we get the trackers' HTTP requests in two steps: (1) Compare the HTTP requests crawled twice to get the missing requests. (2) Remove the requests sent to the first-party server and the requests whose URLs contain less than 2 parameters.

To test the effectiveness of the method above, we crawled the 500 websites again by blocking the manually labelled third-party tracking JavaScript, compared the HTTP requests crawled twice to get the missing requests, and removed the first-party requests and the requests whose URLs contain less than 2 parameters.

Only one unique normal HTTP request is determined by mistake, and less than 10 unique HTTP requests generated by executing third-party tracking JavaScript are determined by mistake because they contain only one parameter in the URLs. In conclusion, based on this method the trackers' HTTP requests can be almost completely correctly obtained when crawling the 500 websites. That is to say, **the overall effectiveness of DMTrackerDetector depends almost entirely on the effectiveness of the classifier**.

Since google-analytics.com is the third-party tracker which appears most frequently in our crawled websites, we also evaluated the situation where first-party websites cache the tracking JavaScript downloaded from google-analytics.com. We found that 62 websites in the 10,000 crawled websites had cached the tracking JavaScript downloaded from google-analytics.com in their own servers.

4.5 Comparison with the Ghostery List

Easy List, Easy Privacy and Ghostery [7] are the most effective blacklists [26]. Easy List is used for blocking advertisement-related tracking, including behaviours such as loading advertisement images, while such behaviours do not collect users' information [2]. Easy Privacy is used for blocking all kinds of tracking, though the number of trackers in Easy Privacy is fewer than Ghostery. In comparison, the Ghostery list covers most trackers, and we compared the efficiency of the list generated by DMTrackerDetector with the Ghostery list.

For the comparison, we firstly obtained the tracking JavaScript through the classifier using the test set in Sect. 4.3. Then, we obtained the HTTP requests generated by the third-party tracking JavaScript, and added all third-party tracking JavaScript and the trackers' generated HTTP requests to the blacklist. Then we labelled all HTTP requests in the 100 websites by using the Ghostery list.

As introduced in Sect. 3.4, Ghostery labels third-party trackers without distinguishing tracking JavaScript libraries and tracking callers, while DMTrackerDetector does not label JavaScript tracking libraries as trackers. Thus, we grouped trackers by domains to compare them. Ghostery labels 283 group trackers in the test set and 243 groups can also be identified by DMTrackerDetector. 40 groups are not considered to be trackers by DMTrackerDetector: 11 groups are used to present advertisements which do not have tracking behaviours; 2 groups are not JavaScript-based trackers which may get users' information such as IP address, city and country by executing background code and send this information to third-party websites via json files; 21 groups are 1*1 pixel images with fewer than two parameters in their URL parameters (15 of them are only images without any information); and only 6 groups are JavaScript-based trackers. Moreover, 35 unrevealed group trackers are detected by DMTrackerDetector, as listed in Table 6.

In conclusion, the results show that our list not only covers almost all JavaScript-based tracking records in the Ghostery list but also contains more trackers than Ghostery.

Table 6. The trackers detected by DMTrackerDetector

ID	The website which embed the tracker	Trackers
1	http://enter.ru	http://adforce.ru/rtcode/p2.php?rp=actionpay&uvid=56d86fb1b0007c910ea39335&source=enter
2	http://enter.ru	http://ads.admized.com/rtb/usermatch.php?umid=37&dataid=0&userid=5541794328814420293&redirecturl=http%3A%2F%2Feu-sonar.sociomantic.com%2Fimg%2F2010-07-01%2Faction%2Fmatch%3Faid%3Dadmized-eu%26fpc%3D5541794328814420293%26id%3D%25userid%25%26&call_type=redirect
3	http://fnac.es	http://ums.adtechjp.com/mapuser?providerid=1006;userid=8056922365943981727
4	http://szonline.net	http://jsonp.aid.alibaba.com/Umid/getDeviceInfo?_cbFunction=fn_qPPsje6I&tokenid=x1K6YmVvY0jbtleDiujejqGkV6OWQKDM&acookie=BmlfDxUpthECAXwQRha3xiwK&n=first
5	http://fnac.es	http://audienceinsights.net/cs/s?t=http%3A%2F%2Frtb-csync.smartadserver.com%2Fredir%2F%3Fpartnerid%3D53%26partneruserid%3D
6	http://komputronik.pl	http://prf.audiencemanager.de/log/profile/user-match?type=js&sec=7f2e41a33090712432b572a6eaf6b10b&advertiserId=56c58220b1a43dbd6323c052&pid=56c58220b1a43dbd6323&r=295053053
7	http://fnac.es	http://adserving.avazudsp.net/dspJS.js
		http://adserving.avazudsp.net/check_adv.php?r=0.7620577486231923&runid=NTg2aWVyX2ZnKzQ=_0&advid=MTgxNG1lX3QrNGlr&pid=0&gettype=0&httptype=1&pid=0&prunid=NTg2aWVyX2ZnKzQ=&k=&MastWeb=&loc=http%3A//www.fnac.es//&referer=&h=480&w=640
8	http://fnac.es	http://www.barilliance.net/data.js.php?a=pv&sid=46133&uid=332872599&pid=&enc=utf-8&cfp=1&lvt=null&ut=1457026759&ses=1&spv=1&1&ref=&br=Chrome&v=32&os=Linux&scw=640&sch=480&th=18&tdw=4&tdm=3&xtr1=%20&xtr2=%20&xtr3=%20&url=http%3A%2F%2Fwww.fnac.es%2F%2F&cm=1&pcm=0&abt=a&pt=U&pidu=1&attu[Member%20Status]=Non-Member&&attud[Member%20Status]=&&&ts=3226
9	http://theblaze.com	http://match.basebanner.com/match?excid=7&cijs=0
10	http://kenh14.vn	http://log1.channelvn.net/scripts/log.js?v=261115
		http://log1.channelvn.net/__ltm.gif?utmu=1286217190&ltma=54548145.194480496.162368097&ltmb=54548145.194480496&ltmc=54548145&ltmts=1457025349569&ltmk=192847022&ltmap=LA-54548145-1&ltmcs=UTF-8&ltmsr=640x480&ltmsc=8-bit&ltmul=en-us&ltmr=0&rf=http://kenh14.vn/
11	http://tv2.dk	https://go.flx1.com/uid?anuid=2552025266156320933&t=&m=84&_rdnr=1
		https://go.flx1.com/imp?id=6143&m=84&pl=39&pubid=TV2.dk&csiz=320x100&advid=Danske+Spil&plid=S%c3%a6rplacering&cpid=2016_DLO_Kontinuerlig_Performance+spor_uge+9-53&cid=EJP_Image_uge36&out=https://go.flx1.com/empty?&rnd=64084
12	http://drudgereport.com	http://dtm.gap.com/dmm/contextweb/match?fpc=3132&pnid=14200&trid=2668032597339262O5&fpctok=1

(Continued)

Table 6. (*Continued*)

ID	The website which embed the tracker	Trackers
13	http://povarenok.ru	http://idntfy.ru/token?e=base64&u=aHR0cDovL290Y2xpY2stYWR2LnJ1L2NvcmUvY29kZS5qcz9waWQ9NTg3JnJpZD01MjAwMTgmcmw9JmxsPWh0dHAlM0EvL3d3dy5wb3ZhcmVub2sucnUvJm93PTYyMCZvaD00NjAmc3c9NjQwJnNoPTQ4MCZwZD04JnZpZHNldHVwPTE=&p=idntfy&n=otclick
14	http://job.ru	http://imrk.net/services/nslookup?app=conv&referer-kw=poadcp+poadcp6116&setuid=1&f=1&test=1&random=8876190
		http://imrk.net/userbind?src=btw&pbf=1&gi=1
15	http://theblaze.com	http://sync.ipredictive.com/d/sync/cookie/generic?http://us-u.openx.net/w/1.0/sd?id=537073028&val=$ADELPHIC_CUID
16	http://bongda.com.vn	http://pub.lavanetwork.net/sites/bongda.com.vn/m.bongda.com.vn.js
17	http://mrporter.com	https://dc.ads.linkedin.com/collect/?pid=6883&opid=8868&fmt=gif&gtmcb=1663206603&ck=&3pc=true&an_user_id=3247088782189976807
18	http://bonprix.de	https://tracking.m6r.eu/sync/adscaleSyncDone?userBuyeruid=c2482089cc8add393ae5721d56e49283&userId=adscale-user:970211457027563663
		https://tracking.m6r.eu/pixel/container?pixelId=6ee465d6-190a-432b-8eac-d14a6d2771e8&pageType=view-home&userId=1937045090604115023&gender=&customerType=&shopCountry=DE
19	http://enter.ru	http://cm.marketgid.com/i.gif?dsp=actionpay&c=56d86fb1b0007c910ea39335
20	http://komputronik.pl	http://app.marketizator.com/mktzsave?event=view&uid=4940760341146969322&session=ses1904746168ion&id_website=2823&page_url=http%3A%2F%2Fwww.komputronik.pl%2F&time=2016-03-3—18:46:15&svo=0&browser=Chrome%2032&resolution=640x480&device_type=desktop&referer_type=direct&visitor_type=new&country=undefined&region=&city=&os=Linux
21	http://couriermail.com.au	http://tags.news.com.au/prod/utrack/utrack.js?cb=14570269227840.03295537573285401
		http://tags.news.com.au/prod/metrics/metrics.js
22	http://couriermail.com.au	http://pixel.newsdiscover.com.au/px1.gif?net_uid=d25dbaf8e951ebf16b279635498dd117&net_platform=web&net_site=tcm&pc_acclevel=none&pc_byref=none&pc_conttypegrant=none&pc_conttyperule=none&pc_memtype=anonymous&pc_pcsid=none&newskey_geo_network=unknown&cbd=11011&nk=d25dbaf8e951ebf16b279635498dd117&nk_src=generator.html&nk_ts=1457026871&nk_isnew=0&nk_chk=32466009-33981823013164103&nk_utl=1&l_nk_src=utrack.js&l_nk_ts=1488562924&ad_krux_segs=&ad_krux_user=&app_pi_ts=1457026966090&app_pi_tz=1&bhgt=460&bwdth=620
23	http://postaffiliatepro.com	https://support.qualityunit.com/scripts/track_visit.php?t=Y&C=Track&B=joxpv41uhy67nfpm5m1qo0liq9xyz&S=26olukdouf1azh3lkgmjwnoxbcokk&pt=Post%20Affiliate%20Pro%20-%20Affiliate%20Tracking%20%26%20Affiliate%20Program%20Software&url=__S__www.postaffiliatepro.com%2F&ref=&sr=640x480&ud=%7B%7D&vn=Y&ci=
24	http://enter.ru	http://cdn.retailrocket.ru/content/javascript/tracking.js

(*Continued*)

Table 6. (*Continued*)

ID	The website which embed the tracker	Trackers
25	http://povarenok.ru	http://adcode.rontar.com/SspCookieySync.axd?dspId=5&userId=3015197682235478616
26	http://rurubu.travel	http://rt.rtoaster.jp/t/?a=RTA-bc4a-1bb923b71630&l=http%3A%2F%2Frurubu.travel%2F&r=&m=&p=&i=0.08605761686339974&c=Shift_JIS
27	http://komputronik.pl	http://app2.salesmanago.pl/static/sm.js
		http://app2.salesmanago.pl/api/r.gif?uri=%2F&location=www.komputronik.pl&uuid=1533d96eb8a-52e9673b9f07-7637696d-a333f81e-7b7d56dd-52e382ca8b0e&referrer=&smid=paxk8cw3on87gvmw&time=2016-03-03T17%3A46%3A54Z&title=Sklep%20komputerowy%20-%20Komputronik.pl&cp=1457027214224
28	http://carsales.com.au	http://ads.stickyadstv.com/user-registering?dataProviderId=183&userId=f7ee56d8-6ff7-4200-869d-3522ea667792
29	http://drudgereport.com	http://cs.tekblue.net/u/cejeiejcfebegdhddhbfbhff.gif?redir=http%3A%2F%2Fr.openx.net%2Fset%3Fpid%3D342e478d-4d67-267b-f081-0eff5f6bc080%26rtb%3D__TEKBLUE_UUID
30	http://shopclues.com	http://st.targetix.net/match?id=4&burl=http%3A%2F%2Fsync.audtd.com%2Fmatch%2Ftargetix%3Fuid%3D%24%7BVID%7D%26fpd%3Dgetintent
31	http://tvtoday.de	http://i.tfag.de/js_ng/tvt_tfm_container.js
		http://i.tfag.de/js_ng/js_gpt_tvt.js
32	http://mediamarkt.de	http://widgets.trustedshops.com/js/X850885BCC73A48D1B290F0459CA39921.js
33	http://shopclues.com	http://tracker.unbxdapi.com/v2/1p.jpg?data=%7B%22url%22%3A%22http%3A%2F%2Fwww.shopclues.com%2F%22%2C%22referrer%22%3A%22%22%2C%22visit_type%22%3A%22first_time%22%2C%22ver%22%3A%222.11.8%22%7D&UnbxdKey=shopclues_dev-u1447936138700&action=visitor&uid=uid-1457025422905-73509&t=1457025422957|0.03593841101974249
34	http://clicksor.com	http://support.yesup.net/visitor/index.php?_m=livesupport&_a=htmlcode&departmentid=4,2,69&custom=YToyOntpOjA7czo4NzoiPGltZyBzcmM9Imh0dHA6Ly93d3cuY2xpY2tzb3IuY29tL2ltYWdlcy9saXZlc3VwcG9ydC5wbmciIGFsdD0iQ2xpY2tzb3IgTGl2ZSBTdXBvcnQiIC8%2BIjtpOjE7czo4ODoiPGltZyBzcmM9Imh0dHA6Ly93d3cuY2xpY2tzb3IuY29tL2ltYWdlcy9saXZlc3VwcG9ydC5wbmciICBhbHQ9IkNsaWNrc29yIExpdmUgU3Vwb3J0IiAvPiI7fQ%3D%3D
35	http://mrporter.com	https://media.yoox.biz/newsl/rd/yorb.htm?yorbid=e254173b-26c0-4cc2-99b4-a137967d6ed7

5 Discussions

Firstly, since we labelled the training dataset manually, the training dataset may contain some bias. However, it is extremely challenging to obtain an ideal, unbiased dataset with perfect ground truth. To reduce possible data sampling bias, we labelled the training dataset by the popular blacklists first, then manually confirmed the JavaScript files labelled as trackers by these lists and manually relabelled the JavaScript files labelled as non-trackers by these lists. We believe that even though the effectiveness of classification in our work may vary a little bit when using different training sets, our major conclusions and insights will likely still hold.

Secondly, this work improves upon our former work in that the APIs correlated to being a tracker now play a key role in the classification, so it is difficult to evade our detection by adding or reducing irrelevant APIs in a tracking JavaScript file. However, it would be feasible to bypass the proposed detection mechanism if one can perform the splitting and merging of JavaScript files. If we split one tracking JavaScript file into different JavaScript files to weaken the impact of the APIs correlated to being a tracker, every JavaScript would likely call a similar API set to non-tracking JavaScript. Thus, trackers can evade our detection by merging pieces of users' fingerprint information in the server side to identify users. For example, a tracker can call screen.height in a.js, screen.width in b.js, location.hostname in c.js, document.title in d.js, etc., and also call some useless APIs in each files (such as document.createcommnet, etc.). The information the tracker obtains in each JavaScript can be used together to identify a user. However, we have not encountered such a situation so far.

Thirdly, another way for trackers to evade our detection is to generate the HTTP requests with some random strings in the URLs when executing the tracking JavaScript. In this way, it would be feasible to bypass the detection when the tracking JavaScript caches in the first-party server. To defend against third-party tracking in this situation, we can normalise the generated URLs to match the random string part; for example we can normalize the URLs by using regular expression. We will normalise the generated URLs to match the random string part in the future work.

6 Conclusion

Third-party tracking on the web has attracted much attention in recent years. The most effective method to defend against third-party tracking is based on blacklists. However, this method highly depends on the records in the database and these known blacklists need to be updated regularly. In this paper, we proposed an effective system named DMTrackerDetector, which can automatically detect third-party trackers offline and output a blacklist. Our system consists of preserving first-party files, extracting features, classification and identifying trackers' HTTP requests. To detect third-party trackers, the four parts work together, and high accuracy is obtained. We also compared the list generated by our system with the Ghostery list. The results showed that our list not only covers almost all JavaScript-based tracking records in the Ghostery list but also contains more trackers than Ghostery.

Acknowledgement. This work is supported in part by the National Natural Science Foundation of China (61272481, 61303239, 61572460), the National key research and development project (2016YFB0800703), the National Information Security Special Projects of National Development and Reform Commission of China [(2012)1424], open Project Program of the State Key Laboratory of Information Security(2015-MS-04). Peng Liu was supported by NSF SBE-1422215, ARO W911NF-13-1-0421 (MURI), and ARO W911NF-15-1-0576.

References

1. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: On the workings and current practices of web-based device fingerprinting. Secur. Priv. IEEE **12**(3), 28–36 (2014)
2. Qianru, W., Liu, Q., Zhang, Y., Wen, G.: Trackerdetector: A system to detect third-party trackers through machine learning. Comput. Netw. **91**, 164–173 (2015)
3. Bau, J., Mayer, J., Paskov, H., Mitchell, J.C.: A promising direction for web tracking countermeasures. In: Web, vol. 2 (2013)
4. Mayer, J.R., Mitchell, J.C.: Third-party web tracking: Policy and technology. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 413–427. IEEE (2012)
5. Adblock plus (2016). https://addons.mozilla.org/zh-cn/firefox/addon/adblock-plus/
6. Donottrackme: Online privacy protection (2016). https://addons.mozilla.org/zh-cn/firefox/addon/donottrackplus/
7. Ghostery (2016). https://addons.mozilla.org/zh-cn/firefox/addon/ghostery/
8. Pan, X., Cao, Y., Chen, Y.: I do not know what you visited last summer: Protecting users from third-party web tracking with trackingfree browser (2015)
9. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless monster: Exploring the ecosystem of web-based devicefingerprinting. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 541–555. IEEE (2013)
10. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: Fpdetective: dusting the web for fingerprinters. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 1129–1140. ACM (2013)
11. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: Persistent tracking mechanisms in the wild. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 674–689. ACM (2014)
12. Ayenson, M., Wambach, D.J., Soltani, A., Good, N., Hoofnagle, C.J.: Flash cookies and privacy ii: Now with html5 and etag respawning. Available at SSRN 1898390 (2011)
13. Soltani, A., Canty, S., Mayo, Q., Thomas, L., Hoofnagle, C.J.: Flash cookies and privacy. In: AAAI Spring Symposium: Intelligent Information Privacy Management (2010)
14. Eckersley, P.: How unique is your web browser? In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 1–18. Springer, Heidelberg (2010)

15. Doubleclick ad exchange real-time bidding protocol: cookie matching. https://developers.google.com/ad-exchange/rtb/cookie-guide
16. Kohno, T., Roesner, F.: University of Washington David Wetherall. Detecting and defending against third-party tracking on the web. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (2012)
17. Noscript (2016). https://addons.mozilla.org/zh-cn/firefox/addon/noscript/
18. Krishnamurthy, B., Naryshkin, K., Wills, C.: Privacy leakage vs. protection measures: the growing disconnect. Proc. Web **2**, 1–10 (2011)
19. Keep-my-opt-outs (2016). https://chrome.google.com/webstore/detail/keep-my-opt-outs/hhnjdplhmcnkiecampfdgfjilccfpfoe
20. Targeted advertising cookie opt-out (2016). https://addons.mozilla.org/zh-cn/firefox/addon/targeted-advertising-cookie-op/
21. Betterprivacy (2016). https://addons.mozilla.org/zh-cn/firefox/addon/better privacy/
22. Kushmerick, N.: Learning to remove internet advertisements. In: Proceedings of the Third Annual Conference on Autonomous Agents, pp. 175–181. ACM (1999)
23. Orr, C.R., Chauhan, A., Gupta, M., Frisz, C.J., Dunn, C.W.: An approach for identifying javascript-loaded advertisements through static program analysis. In: Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society, pp. 1–12. ACM (2012)
24. Bhagavatula, S., Dunn, C., Kanich, C., Gupta, M., Ziebart, B.: Leveraging machine learning to improve unwanted resource filtering. In: Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop, pp. 95–102. ACM (2014)
25. Yamada, A., Masanori, H., Miyake, Y.: Web tracking site detection based on temporal link analysis. In: 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 626–631. IEEE (2010)
26. Mayer, J.: Tracking the trackers: Self-help tools. http://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools, 9 2011
27. Witten, I.H., Frank, E., Trigg, L.E., Hall, M.A., Holmes, G., Cunningham, S.J., Weka: Practical machine learning tools and techniques with java implementations (1999)