I Do Not Know What You Visited Last Summer: Protecting Users from Third-party Web Tracking with TrackingFree Browser

Xiang Pan
Northwestern University
xiangpan2011@u.northwestern.edu

Yinzhi Cao
Columbia University
yzcao@cs.columbia.edu

Yan Chen
Northwestern University
ychen@northwestern.edu

***Abstract*—Stateful third-party web tracking has drawn the attention of public media given its popularity among top Alexa web sites. A tracking server can associate a unique identifier from the client side with the private information contained in the *referer* header of the request to the tracking server, thus recording the client's behavior. Faced with the significant problem, existing works either disable setting tracking identifiers or blacklist third-party requests to certain servers. However, neither of them can completely block stateful web tracking.**

In this paper, we propose *TrackingFree*, the first anti-tracking browser by mitigating unique identifiers. Instead of disabling those unique identifiers, we isolate them into different browser principals so that the identifiers still exist but are not unique among different web sites. By doing this, we fundamentally cut off the tracking chain for third-party web tracking. Our evaluation shows that *TrackingFree* can invalidate all the 647 trackers found in Alexa Top 500 web sites, and we formally verified that in *TrackingFree* browser, a single tracker can at most correlate user's activities on three web sites by *Alloy*.

I. Introduction

Stateful third-party web tracking, the practice by which third-party web sites collect private information about web users, has been adopted by more than 90% of Alexa Top 500 web sites [34]. To track a web user, a third-party tracking[1] site first needs to identify the user by a unique string stored in client-side state. Then, the tracking site associates the identifier of the user with the private information, such as first-party web site domain name, contained in the *referer* header of the third-party request.

Faced with the significance of third-party tracking in the wild, researchers have proposed solutions targeting the two steps of third-party tracking: mitigating the unique identifier like disabling third-party cookies, or cutting off requests with private information like blacklisting known tracking servers. However, neither of the approaches can completely protect users from third-party tracking: in addition to browser cookies, a tracker can store user's unique identifier into many other places on client-side state, such as Flash files [4] [26] [17] and browser caches [26] [4] [17]; meanwhile, blacklist tools highly depend on all the records in the database and a tracking company can always adopt new domains to track users.

To address the shortcomings of existing anti-tracking approaches, a third-party tracking defense system should achieve the following goals:

- *Complete blocking.* The system can completely block all existing stateful third-party tracking techniques, such as those tracked by browser cookies, caches, HTML5 localStorage and Flash files.
- *High function preservation.* While blocking third-party tracking, the system should be compatible to existing web sites and web services.
- *Low performance overhead.* The system should incur affordable overhead compared with that of normal browsing.

In this paper, we propose *TrackingFree*, the first anti-tracking browser that can completely protect users from stateful third-party tracking practice. Instead of disabling places that store third-party unique identifiers, such as browser cookies and flash files, *TrackingFree* automatically partitions client-side state into multiple isolation units (*a.k.a.*, browser principals) so that the identifiers still exist but are not unique any more. Therefore, third-party tracking web sites cannot correlate a user's requests sent from different principals with those identifiers. As a comparison, all existing multi-principal browsers [5], [8], [10], [15], [28], [38] aim to protect users from memory attacks and cannot defend against tracking practices. To summarize, we make the following contributions:

- *Anti-tracking Content Allocation Mechanism.* To obtain the completeness of anti-tracking capability, *TrackingFree* partitions client-side state into different browser principals through a novel content allocation mechanism. *TrackingFree* first allocates initial server-side contents based on the registered domain names, and then allocates derivative server-side contents, such as user-navigated windows/frames and pop-up windows, based on a dynamically generated in-degree-bounded graph with its nodes as browser principals.
- *Privacy-preserving Communication. TrackingFree* is the first multi-principal browser with isolated client-side state that enables communication among different principals. The

[1] In this paper, unless otherwise stated, third-party tracking refers to stateful third-party tracking.

NDSS '15, 8-11 February 2015, San Diego, CA, USA
Copyright 2015 Internet Society, ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23163

communication channels offered by *TrackingFree* are privacy-preserving: they cannot be abused by trackers to bypass the browser's anti-tracking capability.

- *Evaluation using Formal Analysis and Real-world Data.* First, *Alloy* verifies that a single tracker can at most correlate user's activities on three web sites, and further the tracking ability of multiple collaborated trackers is bounded by a factor of the number of involved trackers. On the contrary, *Alloy* also verifies in traditional browsers, even a single tracker can correlate user's activities on unlimited number of web sites. Our evaluation in real world environment shows that *TrackingFree* can block all the 647 cross-site trackers found in Alexa Top 500 web sites. Secondly, in terms of compatibility, *TrackingFree* can correctly render all Alexa Top 50 web sites and run 67 out of 68 third-party services on them. Finally, for performance, *TrackingFree* incurs average latency overhead of 8.29% for opening a new site and 19.43% for cross-site navigation.

The rest of the paper is organized as follows. Section II and Section III present the threat model and system design of *TrackingFree*. Then we analyze how *TrackingFree* preserves existing web site's functionality with real world examples in Section IV. We discuss *TrackingFree*'s implementation and evaluate it in terms of its anti-tracking capability, performance and compatibility in Section V and Section VI respectively. Finally, we present related work in Section VII.

II. Threat Model

In a typical third-party web tracking scenario, three parties are involved, the host web server (the first party), the user (the second party), and the tracking web server (the third party). In our threat model, host web servers are benign. They may put third-party's tracking elements in their web pages, but they do not participate in distributing user's identifiers assigned by third-party trackers among other web sites. The tracking web server, on the other hand, could be malicious in the way that it passively collects the user's private information without exploiting any browser vulnerabilities.

The tracking process happens as follows. First, a user visits a web site on a host web server including first-party contents from the host server and third-party contents from the third-party server. When the user's browser requests the third-party contents, *referer* and *origin* headers containing the host web server's domain name will be sent to the third-party server. Then, the third-party server can correlate such domain name information with the third-party identifier[2] which is set by itself before. After the user visits many host web sites with third-party contents from the same server, the third-party server is able to collect a history of web sites that the user has visited.

Out-of-Scope Goals. We list threats that are *out of scope* below. They are either not as important or popular as stateful cross-site third-party tracking or have been prevented by existing solutions.

- *Within-Site Tracking.* Within-site tracking refers to the practices that trackers are able to monitor user's behaviors within a specific site, but they can not link this user's behaviors in different sites together. Therefore, within-site trackers do not need cross-site unique identifiers. Compared with cross-site tracking, the information that a within-site tracker could acquire is limited within that specific site.
- *Tracking by Exploiting Browser Vulnerabilities.* A third-party web site can utilize a vulnerability in the client browser to actively steal user's browser history, as shown by Liverani et al [24]. In this case, the browser vendor can fix the vulnerability to prevent private information leakage.
- *Stateless Tracking [41] [32].* Stateless (fingerprinting) tracking identifies a user with the passive properties of the client machine and browser, such as IP address, browser feature and plugin version. Tor browser [33] has addressed stateless tracking, so we only focus on stateful tracking. Our architecture can also be deployed to Tor browser to achieve both anti-stateful and anti-stateless tracking capability.

[2]Examples of places to store such third-party identifiers are cookies, HTML5 storage, cache, or a combination of them such as EverCookie [17].

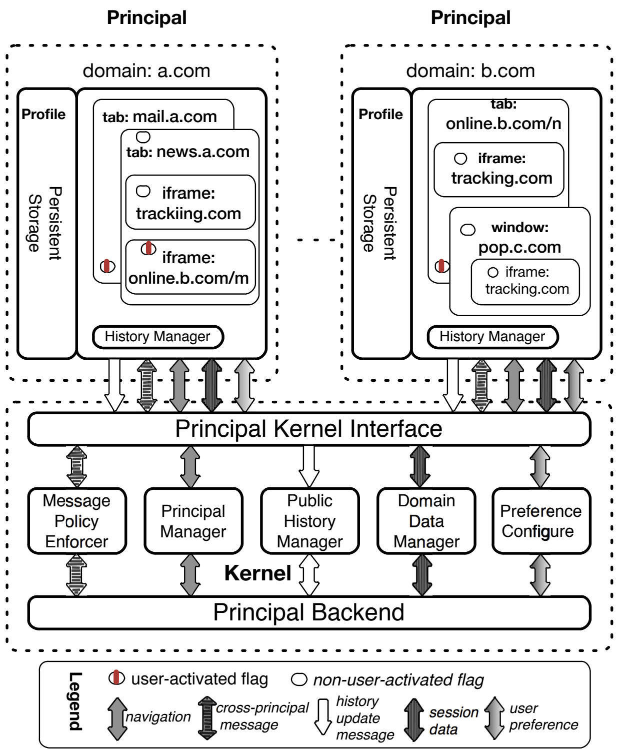


Figure 1: TrackingFree Architecture. TrackingFree adopts profile mechanism to isolate client-side state. All the navigation and communication related events will be permitted and determined by principal manager and message policy enforcer resided in TrackingFree's kernel. Another component in kernel is public history manager, which securely handles all the browsing history, bookmark history and download history of each principal. Preference configure synchronizes all user-initiated preferences among all principals. Domain data manager gives user the flexibility of controlling the balance between anti-tracking capability and user experience. Principal backend records the whole principal organization. Each frame has a flag, indicating whether there are user activities on it.

III. System Design

A. Overview

TrackingFree, illustrated in Fig. 1, is composed of two parts: isolation unit (web principal) and kernel. Each principal, running on the kernel, has isolated persistent storage so the third-party contents in different principals cannot share the same identifiers. We adopt profile based isolation mechanism [9] [31] because of its completeness in isolating client-side state and minimum overhead. Profile will also isolate user preferences, so we propose *preference configure* to synchronize all user-initiated preferences among all principals. See Section III-D for details.

In addition to isolation mechanism, another key factor for *TrackingFree* is content allocation mechanism. In *TrackingFree*, the *principal manager* handles how to allocate contents from the server into different browser principals. Specifically, it dynamically determines how to put different frames into different principals based on user activities, frame properties and principal organization. Principal organization is maintained by *principal backend* as a directed graph with maximum in-degree set as two. We claim that *TrackingFree*'s content allocation mechanism can strike a good balance between privacy and compatibility. We will discuss it in details in Section III-B.

Principal communication also plays an important role for browser's privacy and compatibility. They can be classified as two categories: explicit communication (*e.g.,* postMessage) and implicit communication (*e.g.,* history information sharing). In *TrackingFree*, *message policy enforcer* and *public history manager* handle all the principal communication: the former restricts the range of explicit communication for privacy-preserving purpose and the latter proposes a secure history sharing channel. We will discuss principal communication in Section III-C.

TrackingFree also gives user the flexibility of controlling the balance between anti-tracking capability and user experience. Specifically, the two components of *domain data manager* can decrease the number of principals and share specified domains' sessions among multiple principals, while still achieving expected privacy. *Domain data manager* will be discussed in Section III-E.

B. Content Allocation

TrackingFree's content allocation mechanism is composed of two parts: initial content allocation and derivative content allocation. Initial content allocation handles top frames that are navigated by user directly, that is, the web pages opened through address bar and command line parameters. Derivative content allocation handles frames that are generated due to other frames' contents, such as pop-up window and iframe. We refer to these frames as child frame.

1) Initial Content Allocation: Initial content allocation is based on the registered domain name (*e.g.,* google.com and sina.com.cn) of the top frames directly navigated by users. For these frames, *TrackingFree* will group them in the same principal if and only if they have the same registered domain. For example, in Fig. 1, web pages mail.a.com and news.a.com, though belonging to different subdomains, are located in the same principal. Isolating subdomains will bring little extra privacy benefits, but can break the functionality of a large amount of web sites: two subdomains, say, www.google.com and gmail.google.com, with the same parent domain would not be able to set cookie for each other, which, however, is a very common practice for many web sites.

Plugin objects, such as Flash or SilverLight files, are embedded in web pages and can also be utilized by trackers to store identifiers [35] [4]. We treat plugin objects like other HTML objects: putting them in the same principal as their embedding frames to cut off the identifier sharing channels provided by plugin objects.

Algorithm 1 Principal Switch Determination Algorithm

```
Input:
    $target : Frame$                              ▷ target frame
    $source : Frame$                              ▷ source frame
    $domain : Domain$                             ▷ the domain of the principal
    $isUserTriggered : Boolean$
Output:
 1: $isCrossSiteReq \Leftarrow (\textbf{not}\ equal(target.domain, domain))$
 2: $isMainFrameNav \Leftarrow$
 3:   $(source.isMainFrame()\ \textbf{and}\ isNavigation(source, target))$
 4: if $isMainFrameNav\ \&\ isCrossSiteReq$ then
 5:       return switch-principal
 6: else if $isCrossSiteReq\ \&\ isUserTriggered$ then
 7:       return switch-principal
 8: else
 9:       return non-switch
10: end if
```

2) Derivative Content Allocation: Once a frame is placed in a principal, *TrackingFree* needs to decide how to allocate its child frames. There are two steps in allocating those child frames. First, *TrackingFree* decides whether those child frames should stay in the same principal as their parent frames. A short answer is that *TrackingFree* keeps all non-user-triggered child frames in the principals that their parent frames reside in, and moves all user-triggered cross-site frames to other principals. This process is defined as principal switch. Each frame can at most be switched once in its creation phase. Second, for those frames that need to be moved out of current principal, *TrackingFree* selects an existing principal or creates a new one to render them. This process is defined as principal selection.

Principal Switch. There are two intuitive yet extreme principal switch algorithms: keeping all child frames in current principal (no switch) and making a switch for every child frame. However, keeping all child frames in the same principal allows trackers to collect user's browsing history; making a switch all the time causes serious compatibility issues.

Therefore, *TrackingFree* should make principal switch as little as possible, while still preserving user's privacy. We propose a switching algorithm in Algorithm 1. Whenever a new child frame is about to generate, *TrackingFree* will run Algorithm 1 to determine if it needs to be switched out. For convenience, we refer to the newly-created child frame as target frame, its URL as target URL and the frame that generates target frame as source frame. We also denote the first frame of each principal as main frame.

Main frame's domain should always be consistent with principal's domain. Therefore, if target frame is about to replace main frame, which we denote as main frame navigation, *TrackingFree* will move the target principal out as long as it is a cross-site navigation. In other cases (*e.g.,* navigation on child frame, new popup window), whether or not to switch principal also depends on if the target principal is user-triggered. For example, in Fig. 1, iframe online.b.com/m resides in principal a.com because it is not generated from user's activity; if a user clicks a link toward online.b.com/n in this frame, *TrackingFree* will put the target frame in a new page and render them in a different principal (principal b.com in Fig. 1). The reason we handle these two frames differently is that third-party trackers in this principal do not know a user's real intention for online.b.com/m: it will be automatically visited if the user visits news.a.com. On the other hand, child-frame online.b.com/n is generated by the user and thus shows the user's preferences. Keeping it in the original principal would leak out user's privacy at least to trackers embedded in frames news.a.com and online.b.com/n.

A key for Algorithm 1 is to dynamically monitor whether a *target frame* is user-triggered. In *TrackingFree*, we adopt an approximation approach to do this efficiently: *TrackingFree* hooks up all the click events and keyboard events in any frames. Once an event is triggered, it will label the frame on which the event happens as *user-activated frame*. In Fig. 1, every frame has a property to indicate whether it is a *user-activated frame*. When a new frame gets created, *TrackingFree* can determine if it is user-triggered by checking its *source frame*'s user-activated flag. This approach can result in false positive (mistakenly label an automatically generated *target frame* as user-triggered), but *TrackingFree*'s privacy-preserving capability is still guaranteed because of no false negative.

Principal Selection. We propose an in-degree-bounded graph approach for principal selection to strike a balance between compatibility and privacy-preserving ability. In *TrackingFree*, the *principal backend* (Fig. 1) maintains a graph to organize principals, in which each principal is a node. A directed edge from principal A to principal B exists if and only if at least one principal switch from A to B has occurred before. The principal graph may contain several mutually disconnected sub-graphs. Each sub-graph contains a starting point principal, which is created based on initial content allocation mechanism, and possibly many derivative principals, based on derivative content allocation mechanism. *TrackingFree* is configured with a maximum in-degree value[3] and enforces the number of any principal's parents never exceeding this value. The value determines tracker's tracking capability. If we let P denote the number of principals a tracker can correlate (tracking range) in worst case, k denotes the maximum in-degree value of principal graph, their relationship is given by:

$$P = (k + 1) * (c + 1), \tag{1}$$

where c denotes the number of non-tracking sites willing to relay tracking identifiers for trackers. We have not found such non-tracking web sites in practice, which will be discussed in details in Section VI-A4. Currently, it is safe to say in most cases, c equals to zero. We choose the value of k as two because it can achieve a good balance between privacy and compatibility: setting it to one leads to serious compatibility issues, while any number larger than two extends tracker's tracking range without making further contributions to compatibility. When k equals to two and c equals to zero, we formally verify that *TrackingFree* can limit tracker's tracking range no more than three principal instances using *Alloy* and experimentally demonstrate 67 out of 68 third-party services from Alexa Top 50 web sites work

[3]In directed graph, the number of head endpoints adjacent to a node is called the in-degree of the node. The maximum in-degree of a graph is the in-degree of the node with largest in-degree in the graph.

Algorithm 2 Target Principal Selection Algorithm

```
Input:
    $manager : PrincipalManager$
    $cause : EventCause$
    $source : Principal$
    $request : HTTPRequest$
Output:
 1: $target \Leftarrow NULL$
 2: if $equal(cause, AddressBar)$ or $equal(cause, Argument)$ then
 3:     $target \Leftarrow$
 4:       $manager.findStartingPointPrincipal(request.domain)$
 5:     if not $equal(target, NULL)$ then
 6:         return $target$
 7:     end if
 8:     $target \Leftarrow manager.buildPrincipal(request.domain)$
 9:     $manager.appendStartingPointPrincipal(target)$
10:     return $target$
11: end if
12: $candidates \Leftarrow$
13:     $manager.breadthFirstSearchPrincipalGraphReversely($
14:       $source, request.domain)$
15: loop
16:     $current \Leftarrow candidates.getNext()$
17:     if $equal(current, NULL)$ then
18:         break
19:     end if
20:     $parents\_num \Leftarrow manager.getParentsNumber(current)$
21:     if $less(parents\_num, 2)$ then
22:         $target \Leftarrow current$
23:         break
24:     end if
25: end loop
26: if $equal(target, NULL)$ then
27:     $target \Leftarrow$
28:       $manager.findChildPrincipal(source, request.domain)$
29:     if not $equal(target, NULL)$ then
30:         return $target$
31:     end if
32:     $target \Leftarrow manager.buildPrincipal(request.domain)$
33:     $manager.appendChildPrincipal(source, target)$
34: end if
35: manager.appendParentPrincipal(target, source)
36: return $target$
```

properly in *TrackingFree*. Fig. 2 shows an example of principal graph.

Algorithm 2 illustrates how *TrackingFree* selects target principals. Line 2-11 shows that if a web page is opened through address bar or command line argument, *TrackingFree* will put it in a starting point principal. Line 12-36 indicates when a principal switch occurs, *TrackingFree* first looks up appropriate target principal from source principal's ancestors. If there exists an ancestor principal that meets the following two conditions: 1). its *domain* is same as that of the target URL, and 2). the number of the principal's parents (the node's in-degree) is less than two, *TrackingFree* will set it as the target principal (line 12-25). Only if *TrackingFree* fails to find target principal from ancestors, would a child principal with the appropriate *domain* be selected. If there is no such child principal, *TrackingFree* will construct a new one. After selecting the target principal, *TrackingFree* needs to update the parent-child relationship on principal organization graph maintained by *principal backend* (line 35).

C. Principal Communication

Another important topic for any multi-principal browser is principal communication, which can be classified into two categories: explicit communication and implicit communication. On one hand, principal communication can be abused by

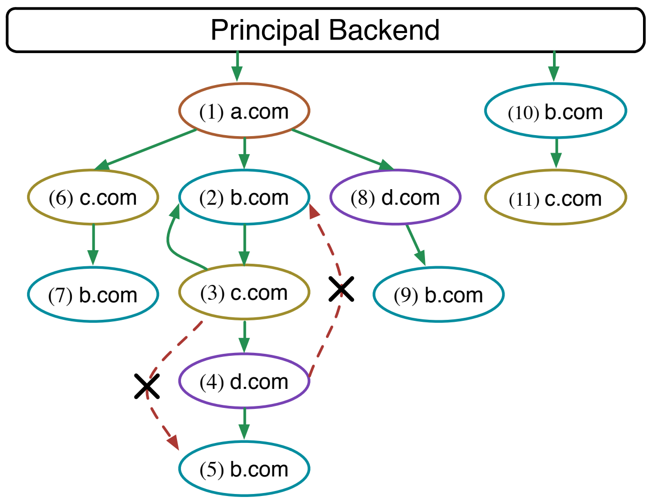


Figure 2: Principal Organization. This figure shows an example of principal graph, in which there exists two starting point principals (1,10). In principal (3), if a principal switch toward b.com occurs, target frame will be placed in principal (2), making principal (3) become one of principal (2)'s parents. However, if another switch toward b.com occurs in principal (4) afterward, principal (2) cannot be the target principal since the number of its parents has achieved the upper bound. In this case, TrackingFree will create a new principal (5) with domain b.com as the target principal.

trackers to bypass principal boundaries and thus leak out user's private information. On the other hand, completely disabling communication severely breaks the functionalities of existing web services. Therefore, to achieve a balance between privacy-preserving capability and functionality, we adopt different policies for different communication channels.

1) Explicit Communication: Explicit communication refers to the message channels that allow cross-origin messaging among different frames through browser APIs. The most common one is postMessage provided by HTML5, which enables asynchronous message passing among a "unit of related browsing contexts" [36]. Obviously, explicit communication can break the isolation mechanism. For example, all of the principals in the same path can be mutually reachable through the *opener* and *frames* properties of *Window* objects. In that case, the web pages placed in these principals can explicitly send tracking identifiers to each other, rendering isolation mechanism useless. To achieve better privacy, we restrict the use of explicit communication in *TrackingFree* as follows:

- Third-party elements in one principal can only explicitly communicate with the first-party elements in its principal;
- First-party elements can only explicitly communicate with first-party elements placed in its neighbor principals[4];

All the other explicit communication channels are not allowed. For example, third-party elements in different principals cannot communicate with each other; first-party elements cannot send messages to third-party elements placed in other principals. As we allow the most common cross-site communication practices (communication between first-party elements in two neighbor sites and communication between iframes with their embedding frames), the compatibility cost caused by these restrictions is negligible. To demonstrate that, we conduct two experiments to show the rarity of violation cases in real word. The results show that among the total of 7,971 message events we found in the two experiments, only six of them will be forbidden by *TrackingFree*. Moreover, these six violation cases are all communicated between first-party elements in one site and third-party iframes with domain doubleclick.net, one of the biggest tracking companies, embedded in other web sites. See Section VI for details.

[4]Two principals are neighbors if and only if they have parent-child relationship

2) Implicit Communication:

Secure History Sharing. The isolation mechanism we adopt will also isolate user's history-related data, such as browsing history, download history and bookmark history. This will affect user experience because history managers located in different principals can only record and manage the history in their principals, causing inconveniences to the user. For example, it cannot provide a user her overall browsing history through browser's history menu and user cannot go back to her previous web page if a principal switch just occurred. One straightforward solution is to allow different principals to share the same history manager, which, unfortunately, will break the isolation boundaries among different principals and might be abused by trackers to share identifiers.

In *TrackingFree*, we propose a secure history-sharing channel. Every time when we start *TrackingFree*, in addition to the history manager located in each loaded principal, an extra public history manager in *TrackingFree* kernel will also be started, which is the only manager that gets associated with browser's GUI. If any of the principals' local history managers gets updated (*e.g.,* a new history item gets added), the local history manager will inform the change, by sending messages, to the public history manager, which, after receiving the message, records the history item and source principal in a map, and then updates *TrackingFree*'s GUI to show the new history item. It is guaranteed that each history or bookmark item will be opened from the principal it is created and the back button can navigate to the correct principal. This secure history sharing channel allows user to use these browser history services without breaking isolation boundaries since all the principals can only write to, instead of reading from, the public service manager.

Communication through Navigation. Principal switch introduces a one-way communication channel that can be utilized by trackers: the tracker can put tracking identifier on cross-site URLs, which once clicked, may lead to principal switch and thus identifier transmission. If target principal belongs to a tracking domain, the tracker can directly correlate source principal and target principal's identifiers; if target principal not controlled by tracker, the third-party tracking elements can still correlate identifiers in the two principal through *referer* header.

TrackingFree does not modify cross-site navigation URLs because it is extremely hard to recognize and strip off the identifiers on URLs without breaking functionalities. But it clears the query strings on non-navigation third-party requests' *referer* headers as it incurs little compatibility cost.

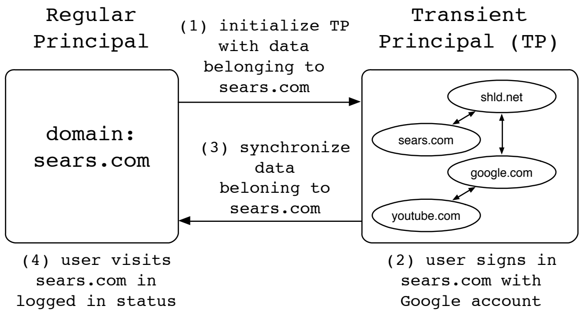

Regular Principal
(1) initialize TP with data belonging to sears.com
Transient Principal (TP)
domain: sears.com
shld.net
sears.com
google.com
youtube.com
(3) synchronize data beloning to sears.com
(4) user visits sears.com in logged in status
(2) user signs in sears.com with Google account

Figure 3: Domain Data Migration Example

D. Preference Configure

TrackingFree adopts profile based isolation mechanism. Profile encapsulates not only data storages (*e.g.,* cookie store, HTML5 local storage), but also user preferences. On one hand, isolating user preferences is necessary for anti-tracking purpose because trackers can abuse them to store identifiers [22]. But on the other hand, it also causes user experience issues: if a user changes her preference on one principal, the change will not be applied to other principals. To address this issue, we propose *preference configure*, which monitors the preference changes on each of the principals and only applies those user-initiated ones to all the other principals. Specifically, when a change occurs, *preference configure* determines if it is caused by browser GUI messages as they can only be generated by users. If so, *preference configure*, considering it as user-initiated, stores the change and applies it to all existing principals. When new principal gets created, *preference configure* will synchronize it with those saved changes. Site-initiated preferences, which might be abused to store identifiers, will never be transmitted among different principals. Therefore, *preference configure* can improve user's privacy without sacrificing privacy-preserving capability.

E. Domain Data Manager

TrackingFree can automatically allocate contents into correct principals and achieve formally-proven anti-tracking capability. However, not all users want such a complete anti-tracking capability and some are willing to trade specified privacy for better user experience. Therefore, we propose *domain data manager (DDM)* component to give users the flexibility of improving their user experience on specified domains, while still achieving privacy for all the other domains. *DDM* is composed of two parts: *domain data migration* and *domain data synchronization*. The former focuses on decreasing the number of principals and the latter aims to improve user experience.

1) Domain Data Migration: Domain data migration allows user to create one transient principal for each regular principal. For convenience, we refer to regular principal as parent principal and assume its domain as parent-dom.com in this section. Transient principal does not have domain attribute and it is neither allowed to switch principal nor to communicate with other principals. The storage of transient principal is initialized with its parent principal's first-party data (*e.g.,* cookies with domain of parent-dom.com). After initialization, *TrackingFree* monitors the transient principal's client-side data and unidirectionally synchronize the changed data belonging to domain parent-domain.com to parent principal to ensure the session state of domain parent-dom.com in transient principal migrated to its parent principal.

Domain data migration can be utilized in, but not limit to, the scenario of complicated single sign-on (SSO) service. A motivating example is logging sears.com with the the user's Google account, which results in a sequence of redirections ranging over four different domains (sears.com, shld.net, goog le.com and youtube.com), leading to four principals in default *TrackingFree*. With *domain data migration* enabled, however, the number of principals involved can be decreased to two. Fig. 3 shows how it works: (1) a transient principal is created with empty storage and then all client-side data belonging to sears.com in parent principal will be copied to it. (2) Once the storage is initialized, the active page in parent principal (login page for sears.com) is opened again in transient principal, where the user then signs in using her Google account. (3) Meanwhile, newly generated client-side data with domain of sears.com in transient principal is unidirectionally synchronized to parent principal. (4) Once the user finishes logging in, she can close the transient principal and visit sears.com in parent principal, where she is in logged in status.

If domain parent-dom.com is not a tracker, *domain data migration* can decrease the number of principals while still preserving privacy in all regular principals because only data with domain of parent-dom.com are synchronized; if parent-dom.com happens to be a tracker, its tracking scope is extended with one more principal: transient principal. As for transient principal, though not armed with anti-tracking capability, it is explicitly created by user so user can control the amount of data leaked out. It is usually short-lived and no data is stored permanently.

2) Domain Data Synchronization: Domain data synchronization allows multiple principals to synchronize client-side data (*e.g.,* cookies, HTML5 local storage) belonging to domains that the user specifies, while still achieving the same anti-tracking capability for all the other domains. It offers user more flexibility to choose between privacy and user experience. We discuss it in details in Section IV-C.

IV. CASE STUDY

In this section, we analyze several popular web service scenarios and show how *TrackingFree* preserves existing web site's functionalities as well as addresses user experience limitations.

A. Single Sign-on (SSO) Service

SSO service is a popular third-party service, allowing user to login a web site using her account on another web site. We first discuss a popular SSO example of login Yahoo using user's Facebook account to demonstrate *TrackingFree*'s compatibility on popular SSO services. Then we analyze why *TrackingFree* can preserve its functionality.

The SSO service works in following steps: (1). a user visits yahoo.com and clicks "sign in with Facebook account" button; (2). then yahoo.com redirects the user to facebook.com with necessary parameters, such as client ID and redirect URL, appended to the URL; (3). after receiving the navigation request, facebook.com authenticates the user through several server-client communications; (4). if authentication succeeds, the user will be redirected back to yahoo.com with access token and several other parameters from facebook.com and SSO process finishes.

When using *TrackingFree*, principal switch occurs in step (2) and (4). As we adopt two-in-degree-bounded graph algorithm, two different principals, one for Yahoo and the other for Facebook, get involved in the SSO scenario and the user will finally be redirected to the original Yahoo principal. More importantly, all necessary information from one principal to another are all passed through URL parameters, which *TrackingFree* allows. These two conditions guarantee *TrackingFree* not breaking this SSO scenario. There exist a few SSO scenarios that are more complicated than Facebook SSO service. For example, login sears.com using user's Google account, which we have discussed in Section III-E1, involves four different principals, but since information are still exchanged through URL's parameters, *TrackingFree* does not break its functionality either.

We have analyzed all the 36 SSO scenarios found in Alexa Top 50 web sites and confirmed that 34 of them only use URL parameter to exchange information. The rest two cases, login ask.com with user's Facebook and LinkedIn accounts, also use postMessage API to pass data among involved principals. *TrackingFree* supports this communication channel, so theoretically, *TrackingFree* does not break any of the SSO scenarios. In practice, this version of *TrackingFree* did fail one SSO scenario (login pinterest.com with Facebook account) due to implementation issue, but user can still use this service with the help of *TrackingFree*'s *domain data migration* component (see Section VI-B3).

B. Cross-site Contents Sharing

With online social network (OSN) getting more and more popular, sharing one site's contents to the user's OSN accounts becomes increasingly popular. We will use the example of sharing a Youtube video to user's Twitter web page to show how *TrackingFree* works for these services.

When a user clicks "Share to Twitter" button below a Youtube video, a navigation request URL toward twitter.com, carrying all the information about the video on its parameters, is generated. Instead of directly sending the request, *TrackingFree* switches to another principal, where the request gets sent out. After receiving the request, Twitter authenticates the user and shares the video to user's account if authentication succeeds. In the process, the communication channel used by the two principals is navigation URL's parameters. As we allow this channel, *TrackingFree* does not break cross-site contents sharing's functionalities. Some other third-party services, including third-party online payment and OSN like/+1 button, have similar working scenarios, so they all work as expected on *TrackingFree*, which has been confirmed by our evaluation results on all 32 non-SSO third-party services found on Alexa Top 50 web sites.

Though not breaking functionalities of third-party services, we admit that *TrackingFree* does affect user experiences: when a user wants to share contents from several different web sites (*e.g.,* Pinterest and Youtube) to the same OSN (*e.g.,* Facebook), the user might need to login her OSN account several times if that has not been done before. We provide *domain data synchronization* component to give user the option of choosing between privacy and user experience, which will be discussed in Section IV-C. Moreover, as all the principals adopt persistent client-side storage, a user only needs to login once for each third-party service. For example, the user does not need to login again when she wants to share the second video from Youtube to her Facebook account.

C. Working on Multiple Principals of a Single Site

TrackingFree can create several different principals for a single web site in order to preserve user's privacy, which might lead to inconsistency. We use the example of shopping on Amazon to demonstrate the issue. When a user visits amazon.com through address bar and puts a book into shopping cart, *TrackingFree* uses amazon.com's starting point principal, whose cookie stores the cart's contents. Then, the user opens another web page of amazon.com by clicking a link on google.com. As the new page is opened through different visiting path, *TrackingFree* renders the page in another principal. This time, the user puts a chocolate into her shopping cart and the cookie records the product. Since the two shopping carts' contents are stored in different cookie storages, the user has to checkout in both principals to purchase the book and chocolate, which affects user experience.

TrackingFree takes two approaches to address the inconsistency. First, when a principal gets activated, *TrackingFree* checks if there exist other principals with the same domain and if so, pops up a notice reminding the user and encourages her to work on one of them. Secondly, the *domain data synchronization (DDS)* of *domain data manager* component allows users to decide the balance between privacy and user experience according to there own needs. *DDS* enables multiple principals to synchronize client-side data (*e.g.,* cookies, HTML5 local storage) belonging to domains that the user specifies, while still achieving the same anti-tracking capability for all the other domains. *DDS* is disabled by default as *TrackingFree* gives privacy and compatibility the highest priority. To enable it, a user needs to register 1). one or several synchronizing domains and 2). principals to be synchronized for each of the domains, which we refer to as synchronization scope. *DDS* monitors and synchronizes each specified domain' data among corresponding synchronization scope. In this example, synchronizing domain should be set as amazon.com while synchronization scope set as all principals with domain of amazon.com. In the contents sharing scenario discussed in Section IV-B, they should be set as facebook.com, then the user only needs to login her Facebook account once when sharing contents from various other web sites.

Once domain data synchronization is enabled, the overall formally-proven privacy-preserving capability of *TrackingFree* cannot be guaranteed. In the example discussed above, trackers belonging to amazon.com can correlate user's activities on at most $3 * N$ principals without collaborating with other sites, where N is the number of Amazon principals. In practice, this is hard to achieve because the condition for the worst case is not easy to meet. Moreover, since it only synchronizes the data belonging to amazon.com, *TrackingFree* still protects user against trackers from other domains.

V. Implementation

We implemented a proof-of-concept *TrackingFree* in Chromium with around 1,800 lines of codes. We changed

Chromium's profile mechanism to isolate different principal. Each principal resides in a unique directory, in which all the persistent storages (*e.g.,* cookie jars, cache storages), user preferences and plugin files are stored. *TrackingFree* maintains a table to map principals to these directories, which guarantees that no two principals correspond to the same directory. Moreover, we hooked *WebUIImpl::ProcessWebUIMessage* and *PrefService::Set* methods to monitor user-initiated preference change.

For better efficiency, we modified Chromium's render and asked *TrackingFree* to determine principal switch directly in render process. If a principal switch is required, *TrackingFree* passes the request information to browser kernel, where *principal manager* determines target principal. We hooked all the click events to record whether there are user activities on each frame.

All cross-site messages will be intercepted and sent back to browser kernel, where *TrackingFree*'s message policy enforcer checks if each message is permitted based on our policies and, if it is, sends these messages to target principal through IPC messages. We also implemented a history manager object for each principal (profile) and it monitors the principal's *HistoryService*, *DownloadManagerImpl* and *BookmarkModel* objects. Whenever there is a change, the history manager will inform it to public history manager object, which is running on an isolated daemon principal and responsible for updating browser UI.

As for domain data manager, the current implementation only supports cookie migration and synchronization, but it can be easily extended to support other storages. We hooked all cookie message handlers, such as *OnSetCookie* and *OnDeleteCookie* in class *RenderMessageFilter*, to monitor and update cookie changes in different principals.

VI. Evaluation

We evaluate *TrackingFree* in two ways. First, we perform a formal analysis for its anti-tracking capability. Then we experimentally assess the system's anti-tracking effectiveness, performance overhead and compatibility. In this section, unless otherwise stated, both domain data migration and domain data synchronization are disabled.

A. Formal Analysis & Discussion

We use model-checking tool to formally analyze *TrackingFree*'s anti-tracking ability. To compare *TrackingFree* with browsers adopting other content allocation mechanisms, we further evaluate browsers that adopt top-frame based and same-origin-policy (SOP) based content allocation mechanism. The model-checking framework we adopt is *Alloy* [16], a declarative modeling language based on first-order relational logic. *Alloy* has been used in a wide range of applications to find vulnerabilities in security mechanisms, among which, Akhawe et al. [3] have proposed a complete web model in *Alloy*. Our evaluation is based on the client/server model described in [3], whose client-side is redesigned according to *TrackingFree*'s architecture. In formal analysis, we assume that non-tracking web sites do not relay identifiers for tracking sites and different tracking sites do not exchange a user's identifier with each other. We will discuss tracker's tracking range in Section VI-A4 if the assumptions do not hold.

1) TrackingFree Evaluation:

Security Goal. The broader security goal of *TrackingFree* is to defend against cross-site web tracking. We distill two more concrete goals, both of which are modeled by *Alloy assertion*s.

- For any two HTTP requests toward tracking servers while sent from different principals, we denote the identifiers carried by each of them as $setA$ and $setB$, then the following formula should always hold:

$$(setA \cap setB \neq \emptyset)\, implies\, ((setA \subseteq setB) or (setB \subseteq setA))$$

- The tracking identifiers carried in any HTTP request toward tracking servers are at most stored in the client-side states of three different principals.

The first goal shows that if tracking servers can correlate several different identifiers, these must exist at least one request carrying all of these identifiers. The second indicates the maximum principals that can be correlated by tracking companies through one request. Therefore, the two goals, if met, illustrate the extent a *TrackingFree* user can be tracked in the worst case.

Modeling TrackingFree. In *Alloy*, we model *TrackingFree* through *Alloy signature*, which represents a set of objects. Every *Alloy signature* has zero to several properties, which, from the view of set theory, are relations. We also describe the two goals discussed above as two *Alloy assertion*s, which will be checked by *Alloy* in a specified scope. *Alloy* tries to find counter examples for these *assertion*s. If it does not find one, we say that the *assertion*s hold up to the scope size we specified since *Alloy* will not miss any counter examples in the specified space. When searching for counter examples, *Alloy* follows the *facts* described by users. In our model, we describe the behaviors of *TrackingFree* and tracking servers in twenty-three *facts*.

```
one sig TrackingFreeKernel extends Browser{
  startingPoints: Origin -> lone Principal
}
sig Principal{
  persistentStorage :
    set PersistentStorageUnit,
  domain : Origin,
  childPrincipal :
    Origin -> lone Principal,
  parent : set Principal,
  browser : one TrackingFreeKernel,
  mainFrame : one ScriptContext
}
```

Code 1: TrackingFree Signatures

Code 1 describes the signature of *TrackingFree*. Each *TrackingFree* is composed of one *TrackingFreeKernel* object and multiple *Principal* objects. The relation *startingPoints* defined in signature *TrackingFreeKernel* maps each *TrackingFreeKernel* object to a set of relations, which then relates each *Origin* to its *Principal*, if there is one. Through the set of relations, we can get all the *TrackingFree*'s starting point principals by their *Origin*s. The relations in *Principal* are also straightforward. The *persistentStorage* relates each principal to its client-side state, where tracking identifiers assigned to this principal are stored; *domain* illustrates the principal's *Origin*, which is the origin of main frame, accessed by relation *mainFrame*; *childPrincipal*

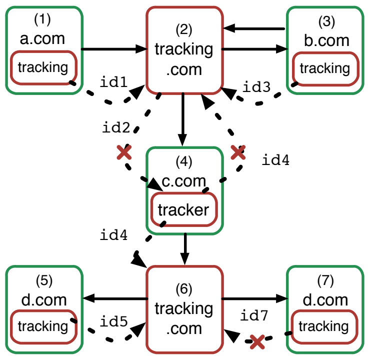


Figure 4: Worst Case Scenario without Site Collaboration. Tracking elements embedded in non-tracking web site principals (1) and (3) can send identifiers *id1* and *id2* in their principals to tracking principal (2). No other principals can send more identifiers to (2) because of two two-in-degree-bounded-graph algorithm and communication policy. Similarly, tracking principal (6) can collect identifiers from at most two other principals. The identifiers collected by principal (2) and (6) can not be linked together without collaborating with non-tracking web site *c.com*.

and *parent* map the current principal to its child and parent principals.

Codes 2 describe the security goals *Alloy* verifies. The first *assertion* asserts that for any tracking identifier stored in a principal, say *principal A*, no matter whether this identifier is directly assigned by tracking servers or passed from other principals, there must be a *HTTPResponse* from tracking servers carrying the identifier and then storing that identifier on *principal A* (this has also been verified by *Alloy*). For an identifier stored in a principal, we refer to the *HTTPResponse* that stores the identifier in that principal as *carrier response*. In second *assertion*, *setIndetifier* is a relation that relates each *carrier response* to its identifiers. Therefore, *getPrincipal[setIdentifier.(areq.carryData&TrackingIdentifier)]* can get all the principals that store at least one of the identifiers carried by *HTTPRequest areq*.

By running *Alloy*, we verify that both goals are upheld, up to thirteen NetworkEvents (including *HTTPRequest* and *HTTPResponse*), meaning that in worst case, trackers can correlate *TrackingFree* user's activities up to three site instances. Fig. 4 shows an example of *TrackingFree*'s worst case scenario without site collaboration. We will also use it to discuss site collaboration in later part of the section. Principal (2) and (6) are both trackers, belonging to the same domain tracking.com. Other green principals (1), (3), (4), (5) and (7) belong to different non-tracking domains, all of which embed third-party tracking iframes of tracking.com. The solid line shows principals' parent-child relationship (*e.g.,* principal (2) is opened through a cross-site navigation from principal (1)) and the dotted line shows communication channel. In the worst case, principal (2) can collect tracking identifiers *id1* and *id3*, sent by third-party iframes embedded in principal (1) and (3), so trackers can correlate user's activities in principal (1), (2) and (3). Note that due to the two-in-degree-bounded graph algorithm and *TrackingFree*'s communication policy, third-party iframe in principal (4) cannot send identifier *id4* to principal (2) neither explicitly (*e.g.,* postMessage API) nor implicitly (*e.g.,* URL parameters). Principal (2) can send its identifier *id2* to principal (4), but without site c.com's collaboration, third-party tracking elements in principal (4) cannot access *id2*, so tracker is not able to correlate tracking identifier *id4* with any of *id1*, *id2* and *id3*. Similarly, principal (6) can collect *id4* and *id5* from principal (4) and (5), but tracker is not able to correlate them with identifiers collected by principal (2).

2) Top-frame based Content Allocation Evaluation: Top-frame based allocation mechanism isolates principals based on web page's top frame. All the web pages and their contents, including third-party iframes, are kept in the same principal if and only if they are from the same site.

We first give the assumption that top-frame based content allocation mechanism cannot protect user's privacy and then use *Alloy predicate* to prove it. We give three principals and claim that there exists an *HTTPRequest* that contains the identifiers assigned to each of the principals and thus allow the tracker to correlate user's activities on all of these principals. *Alloy* will completely search the specified space to find an instance that satisfies the *predicate*. The evaluation result shows that *Alloy* find such an instance, illustrated in Fig. 5. We only specify the number of principals to be three because it is sufficient to demonstrate this issue and larger search space takes *Alloy* much longer time to evaluate.

```
assert IdentifiersCanBeFoundInOneRequest{
 all areq1, areq2 : HTTPRequest |
  (getPrincipal[areq1] !=  getPrincipal[areq2] and
   toTrackingHost[areq1]=TRUE and
   toTrackingHost[areq2]=TRUE and
   hasCancelled[areq1]=FALSE and
   hasCancelled[areq2]=FALSE and
   some areq1.carryData&TrackingIdentifier and
   some areq2.carryData&TrackingIdentifier)
  implies{
   (areq1.carryData&TrackingIdentifier in
     areq2.carryData&TrackingIdentifier) or
   (areq2.carryData&TrackingIdentifier in
     areq1.carryData&TrackingIdentifier)
 }
}
assert CorrelateLessEqualThanThree{
  all areq : HTTPRequest |
     (hasCancelled[areq]=FALSE  and
      toTrackingHost[areq]=TRUE) implies
       #(getPrincipal[
         setIdentifier.(
           areq.carryData&TrackingIdentifier)])<=3
}
```

Code 2: TrackingFree Privacy Goals

In this figure, a user first visits tracking.com, making the browser create a principal with domain tracking.com. tracking .com sets an identifier *id*1 to this principal (1). Then the user visits a.com and b.com. Browser creates two more principals for them. Both of these two sites contain third-party elements from tracking.com, leading identifiers *id*2 and *id*3 assigned to these two principals respectively (2, 3). After that, two cross-site navigations toward tracking.com, possibly due to link clicking or JavaScript execution, occur on the two principals. These two navigations will be first moved to principal tracking.com and then sent out. Tracking elements are able to append identifiers *id*2 and *id*3 to navigation requests and therefore pass them to target principals (4, 6). Finally, the principal *tracking.com* can send requests with all the three identifiers to tracking.c om and allow it to correlate user's activities on these three principals (7). Although it only shows three principals correlated, this scenario can be conveniently extended to any number of

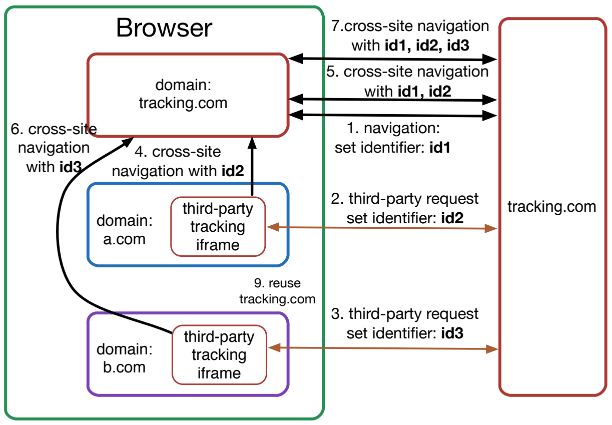

Browser
domain: tracking.com
7.cross-site navigation with id1, id2, id3
5. cross-site navigation with id1, id2
1. navigation: set identifier: id1
6. cross-site navigation with id3
4. cross-site navigation with id2
domain: a.com
third-party tracking iframe
2. third-party request set identifier: id2
tracking.com
9. reuse tracking.com
3. third-party request set identifier: id3
domain: b.com
third-party tracking iframe

Figure 5: Top-frame based Content Allocation Mechanism Counterexample. This figure shows a scenario found by *Alloy* that tracker can bypass top-frame based content allocation mechanism. Request 7 carries all the identifiers assigned to the three principals and makes tracker able to correlate user's activities on the three principals.

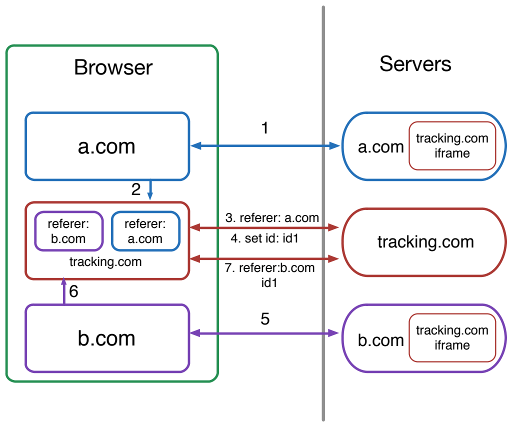


Figure 6: SOP based Content Allocation Mechanism Counterexample. This figure shows a scenario found by Alloy that tracker can bypass SOP based content allocation mechanism. All the frames belonging to tracking.com will be sent from the same principal, where they can access the same identifier. Requests 3 and 7 tell trackers the user has visited a.com and b.com

principals, indicating that *TrackingFree* with top-frame content allocation mechanism cannot completely defend against web tracking practice.

3) SOP based Content Allocation Evaluation: SOP based allocation mechanism strictly follows same-origin policy and puts all the child frames with different origins into different principals while loading the main page. Through *Alloy predicate*, we show that SOP based content allocation mechanism cannot protect users from web tracking. Fig. 6 illustrates the instance found by *Alloy*.

In Fig. 6, both web pages a.com and b.com contain iframe tracking.com. When a user visits these two pages, SOP based content allocation browser will assign the two embedded iframes the same principal, in which they will send requests to load the frames from tracking.com. However, although these two requests have been moved to principal tracking.com (2,6), their *referer* headers are still there and can inform tracking.com the user has visited a.com and b.com. This scenario can also be extended to unlimited number of web sites easily. From privacy-preserving ability's perspective, SOP-based allocation browser is worse than top-frame based browser.

In summary, without site collaboration, neither top-frame based or SOP based content allocation browsers can be used to prevent web tracking because user's activities on unlimited number of web sites can be tracked.

4) Site Collaboration: Without site collaboration, we have formally proved that *TrackingFree* can limit the tracking range within three principals. In this section, we discuss *TrackingFree*'s anti-tracking capability when multiple web sites collaborate to track users.

Non-tracking Site Relay Tracking Identifier From Fig. 4, tracking principal (2) can collect identifiers *id1*, *id2* and *id3*, while principal (6) gets identifiers *id4*, *id5* and *id6* in worst case. If non-tracking web site c.com collaborates with tracker and relays the tracking identifiers from principal (2) to (6), tracker can correlate the two sets of identifiers and correlate user's activities on the six principals. Generically, based on formula 1, each tracker T can at most correlate user's behaviors on $3 * (c + 1)$ principals, where c is the number of non-tracking sites that relay identifiers for tracker T. Among the $3 * (c + 1)$ principals, tracker T can track user on $2 * (c + 1)$ principals belonging to other sites in worst case because at least $(c + 1)$ principals are from tracker itself. In reality, we claim that tracker's tracking range is still limited to three principals without exchanging identifiers with other sites because the prerequisite for relaying identifiers is difficult to meet: the non-tracking site has to be navigated through a tracking principal and then open another tracking principal. Moreover, we have not found any of such collaboration scenarios.

Multiple Sites Exchange Identifiers Two sites, $T1$ and $T2$, can collaborate and send a user's identifier to each other so they can exchange tracking database on the back end. *TrackingFree* cannot prevent such collaborations. If $T1$ can track user on X principals and $T2$ on Y principals, their tracking ranges will be extended to $X + Y$ principals in worst case after collaboration. However, *TrackingFree* can effectively limit the tracking capability of each tracker, so even after they collaborate, user's privacy is still protected. For example, in vanilla browser, double-click.net can track user on 117 web sites and scorecardsearch.com can track user on 133 web sites among Alexa Top 500 web sites. After collaboration, user's activities on up to 250 web sites might be tracked. In *TrackingFree*, however, user's activities on at most 6 principal instances (4 non-tracking principal) can be linked together even after the two giant trackers exchange user's identifiers.

In practice, a more common example is that a third-party tracking site assists several other host web sites to track their users (*e.g.,* Google Analytics). These host web sites import a snippet of JavaScript code from the tracking site (*e.g.,* google.c om), through which, they send their first-party cookies to the tracking site, but they will not send their cookies to each other. The collaboration is unidirectional: the tracking site collects host web site's first-party cookie, but it will not send its own cookie back to the host web site. If we let P denote the number of principals tracking site can correlate in worst case, R_i denote

Tracking Host	Prevalance (# Domains)	Tracking Token(s)
b.scorecardresearch.com	133	UIDR
ad.doubleclick.ne	117	id, __gads
ib.adnxs.com	75	anj
p.twitter.com	70	__utma
cm.g.doubleclick.net	56	id
ad.yieldmanager.com	52	bx
bs.serving-sys.com	40	A4
cdn.api.twitter.com	40	__utmz
secure-us.imrworldwide.com	38	IMRID
adfarm.mediaplex.com	31	svid
d.adroll.com	30	__adroll
img.mediaplex.com	29	svid
ds.serving-sys.com	27	u2, __qca,A4
hm.baidu.com	27	BAIDUID
pixel.mathtag.com	23	uuid,HRL8

Table I: Top 15 Cross-site Trackers on Alexa Top 500 web sites

the tracking range of host web site i, N denote the number of host web sites, and functions max and $secondmax$ calculate the largest value and second largest value of a set respectively, tracking site's tracking range is given by:

$$P = max\{R_1 \ldots R_N\} + secondmax\{R_1 \ldots R_N\} + 1 \quad (2)$$

So the tracking site's tracking range is the sum of the top two host sites' tracking ranges plus one. Host web sites' tracking ranges remain the same. If host sites are not trackers (only track users within their own domains), tracking site's tracking range is still limited to three. If host web sites also serve as third-party trackers, tracking site's ability is restricted within seven principals, were host sites not exchanging identifiers with others.

B. Experiments with Real World Web Sites

1) Anti-tracking Capability: In order to measure *TrackingFree*'s ability in defending against cross-site third-party tracking, we first gather trackers on Alexa Top web sites by strictly following the tracker detection methodology presented by existing work [34], and then evaluate how many of them can be invalidated by *TrackingFree*. This approach, involving both manual and automatic workloads, can only give a lower bound of trackers, but the evaluation can still be a good indicator showing *TrackingFree*'s anti-tracking ability.

Specifically, we conduct the experiment on Alexa Top 500 web sites. In each site, we randomly visit four more pages. As some links are broken or redirect to the same link, we finally get 2,032 valid URLs as our dataset. We visit these URLs and capture all the HTTP(S) traffics. This process is repeated twice: once starting with a clean browser and once more after priming the client-side state. After that, we analyze the captured traffics and identify cross-site trackers based on their behaviors summarized by work [34].

We first conduct the experiment in standard Chromium. We find a total of 647 trackers, appearing in 5,814 requests, which we refer to as *tracking requests*. Table I lists the top 15 prevalent cross-site tracking hosts ordered by the number of domains they appear. Then we repeat this experiment with *TrackingFree*. The results show that we do not find any trackers. *TrackingFree* block all of the 647 trackers found in vanilla Chromium.

Source	Cost (ms)
Principal Construction	322.36
Persistent State Construction	98.03
In-memory State Construction	224.33
Extra IPC	349.06
Render/JS Engine Instrumentation	139.21

Table II: Cost of *TrackingFree*

We also find that blacklist-based tracking prevention products can not effectively protect users from tracking. We use the blacklist tool Ghostery [13] for comparison, which is one of the most effective privacy protection extensions based on open source measurement project "Are We Private Yet" [1], Through examining the 5,814 tracking requests on Ghostery's blacklist, we find the blacklist missing 1,057 requests, belongs to 53 tracking hosts.

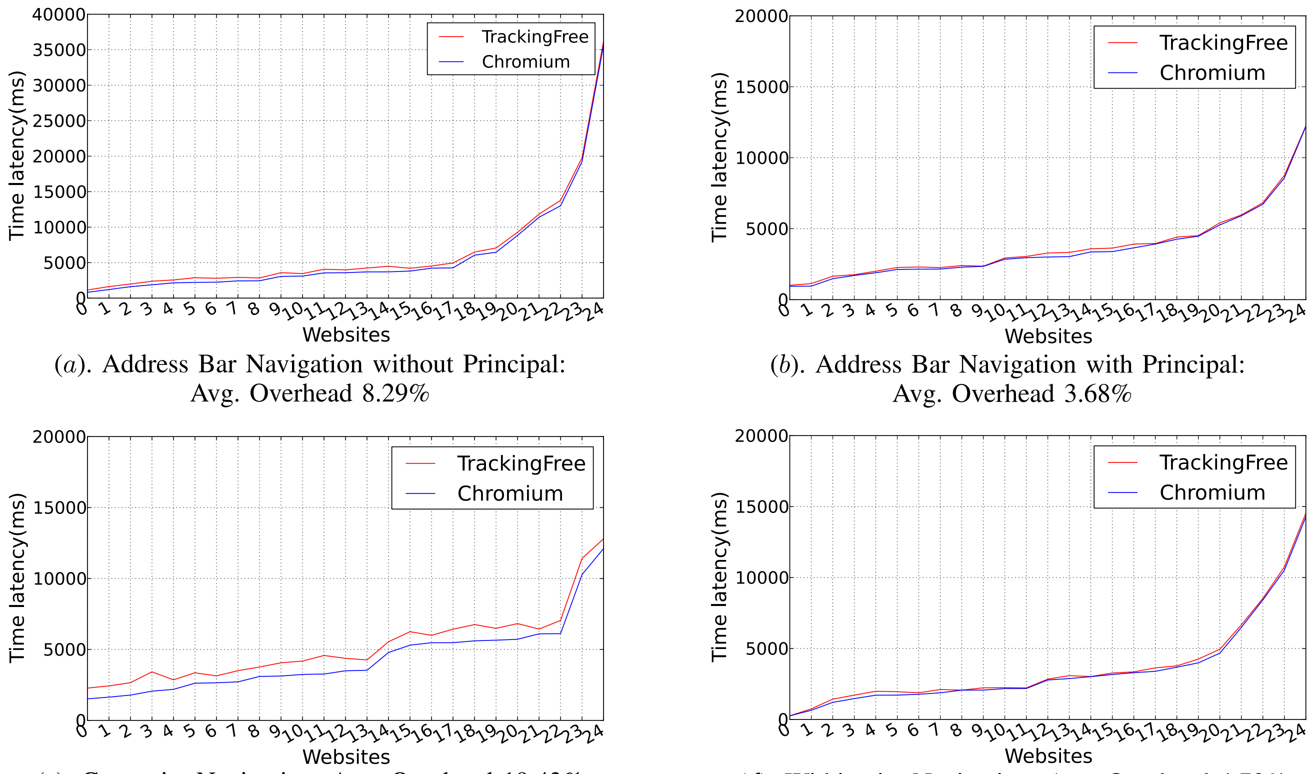
2) Performance: We measure the impact of our architecture on browser performance. All experiments are done in 1.3GHz dual-core Intel Core i5 processor, 4GB memory, 128GB flash storage, running OSX Mavericks operating system. To evaluate *TrackingFree*'s performance, we have measured page loading time, memory usage and disk usage.

Latency. The latency cost of *TrackingFree* comes from three sources: 1). principal construction, 2). extra inter-process communication (IPC) and 3). render and JavaScript engine (render/JS engine) instrumentation. Principal construction refers to the extra cost used to build a principal in *TrackingFree*, compared with the cost for creating a new render process in Chromium; extra IPC cost only exists in cross-site navigation, in which the navigation request will be passed to browser process from source render process and then sent back to target render process after new principal has been constructed; render/JS engine instrumentation cost results from extra instrumentations on WebKit engine and JS engine, such as hooking all mouse and message events, existing in all the navigation practices.

We evaluate the costs of principal construction (1) and render/JS engine instrumentation (3) by browsing Alexa Top 25 web sites with *TrackingFree* and calculating their average cost. To measure extra IPC cost (2), we visit Alexa Top 25 web sites and then randomly click a cross-site link from each of the site. Table II shows the average of each cost. Note that not all the browsing scenarios will suffer from all the overhead listed above. Specifically, cross-site navigation is the only scenario that might experience all the three costs, so on average, compared with vanilla browser, *TrackingFree* incurs ~810 ms extra overhead for cross-site navigation. In other scenarios, the average costs range from ~140 ms to ~460 ms.

To measure the overall impact of our architecture on browser performance, we further conduct the following four experiments, spanning browsing scenarios from lowest latency case (within-site navigation) to highest latency case (cross-site navigation). Each experiment is repeated five times to reduce the disturbance from networking delay. Fig. 7 is based on the median values of the five-round results.

- **Address Bar Navigation without Principal:** Experiment a is to evaluate the overall latency of visiting web pages whose principals have not been created. In this experiment, a web page's loading time is calculated from the time pressing



(a). Address Bar Navigation without Principal: Avg. Overhead 8.29%

(b). Address Bar Navigation with Principal: Avg. Overhead 3.68%

(c). Cross-site Navigation: Avg. Overhead 19.43%

(d). Within-site Navigation: Avg. Overhead 4.70%

Figure 7: Latency Overhead

the enter key on the address bar to the time the page's load event is triggered. This experiment aims to measure the overhead incurred by principal construction and render/JS engine instrumentation. We do the experiments on the main pages of Alexa Top 25 web sites. The average loading time is 6,523.63 ms for *TrackingFree* and 6,024.34 ms for Chromium. *TrackingFree* incurs an overhead of 8.29%. We also notice that the last web page takes much more time than others, significantly affecting the average time. Without the last point, the result becomes 5,286.89 ms for *TrackingFree*, 4,790.64 for Chromium. The overhead of *TrackingFree* increases to 10.36%.

- **Address Bar Navigation with Principal:** Experiment b is similar to experiment a, visiting a web page through address bar. But this time, the target principal has been built beforehand. This experiment aims to measure the overhead incurred by render/JS engine instrumentation. The average loading time is 3,798.11 ms for *TrackingFree* and 3,663.44 ms for Chromium. *TrackingFree* incurs an overhead of 3.68%.
- **Cross-site Navigation:** Experiment c is to evaluate the loading time for cross-site navigation in which the target principal has not been created. This experiment aims to measure the worst case of *TrackingFree*. The overhead is incurred by principal construction, extra IPC and render/JS engine instrumentation. We randomly click 25 cross-site links from Alexa Top web sites, at most one link from each of these sites. The loading time is calculated from the point of clicking the link to the point that the page's load event is triggered. On average, it takes 5,232.90 ms for *TrackingFree* and 4,381.46 ms for Chromium to fully load the cross-site navigation page. *TrackingFree* incurs an overhead of 19.43%.
- **Within-site Navigation** Experiment d is to evaluate the loading time for within-site navigation, the overhead of which is only incurred by render/JS engine instrumentation. We randomly click one within-site link from each of the Alexa Top 25 web sites. The loading time is calculated in the same way as experiment c. The average loading time is 3,289.93 ms for *TrackingFree* and 3,142.01 ms for Chromium. *TrackingFree* incurs an overhead of 4.70%.

Overall, the four experiments illustrate *TrackingFree*'s latency performance in the range of 3.68% to 19.43%.

Memory and Disk Overhead. To evaluate *TrackingFree*'s memory overhead, we conduct the following three experiments:

- We visit 12 web pages within *one* domain and monitor browser's memory overhead. The first rows in Table III and Table IV illustrate the results of memory usage and disk usage. The overhead incurred by *TrackingFree* is 27.9MB for memory and 0.5MB for disk.
- We visit 12 web pages within *four* domains. The second rows in Table III and Table IV illustrate the results of memory usage and disk usage. The overhead incurred by *TrackingFree* is 79.2MB for memory and 3.4MB for disk.
- We visit 12 web pages within *twelve* domains. The third rows in Table III and Table IV illustrate the result. The overhead incurred by *TrackingFree* is 207.9MB for memory and 5.7MB for disk.

Overall, the evaluation results show that each loaded and opened principal take about 20MB space in memory, while each created principal takes 0.5MB-1MB space in disk. The number of web pages in each principal does not significantly affect the memory overhead. *TrackingFree* records the usage history of each principal and accepts a threshold (*i.e.,* 1,000 MB) from

Memory	Chromium	TrackingFree	Increase
1 Domain	477.1(MB)	505(MB)	27.9(MB)
4 Domains	623.6(MB)	702.8(MB)	79.2(MB)
12 Domains	434.6(MB)	642.5(MB)	207.9(MB)

Table III: Memory Overhead

Disk	Chromium	TrackingFree	Increase
1 Domain	21.3(MB)	21.8(MB)	0.5(MB)
4 Domains	22.5(MB)	25.9(MB)	3.4(MB)
12 Domains	23.7(MB)	29.4(MB)	5.7(MB)

Table IV: Disk Overhead

user's configuration file as the maximum disk space that can be allocated. Once the threshold is achieved, *TrackingFree* will delete those least frequently used principals.

3) Compatibility: We evaluate *TrackingFree*'s compatibility through three experiments. The first experiment is to evaluate whether *TrackingFree* is backward compatible to existing first-party web sites. We manually run *TrackingFree* on Alexa Top 50 web sites. For each site, we open its main page through address bar. Once it has been fully loaded, we continue to perform a within-site navigation and a cross-site navigation, if there is one. In this period, we check the visual appearance of each web page and the stability of *TrackingFree*. The results show that *TrackingFree* is completely backward compatible to these 50 web sites.

The second experiment is to measure *TrackingFree*'s compatibility toward third-party services. In this experiment, we find 68 third-party services from Alexa Top 50 web sites and manually test them on *TrackingFree*. The third-party services we have found can be classified into three categories: 1). cross-site online payment; 2). cross-site content sharing; and 3). single sign-on (SSO). The evaluation results show that 67 out of 68 third-party services work properly on *TrackingFree*. We discuss in details why *TrackingFree* preserves these services' functionalities in Section IV. The failed case is a SSO service, logging pinterest.com through user's Facebook account. Through analysis, we believe it resulting from implementation issue: a JavaScript error leads to a post request not sent out, and thus not able to finish the last login step. User can still use this failed SSO service with little privacy cost and user burden through *TrackingFree*'s domain data migration component: performing SSO login activity in transient principal and switching back to regular principal after login succeeds.

The third experiment is to evaluate whether the two communication restriction rules are reasonable. The first rule regulates that in the scenario of user visiting web site A, then opening web site B from A, and then site C from B, communication between web sites A and C is not allowed. In order to measure whether the forbidden cases are rare, we find and test 70 browsing practices in this scenario ($A \Rightarrow B \Rightarrow C$) from Alexa Top 500 web sites and evaluate the number of communications among web site A, B and C. The results show that the number of message events between A and B, as well as B and C is 3,119, which *TrackingFree* allows, while the number of message events between A and C is zero. The second rule forbids third-party elements, embedded in A, communicating with elements in sites other than A. In this experiment, we evaluate the number of message events between site B and C, in the scenario that there exists a site A, which embeds an iframe of site B and a cross-site link toward site C. We find and test 58 browsing practices in this scenario from Alexa Top 500 web sites. The results show that the number of message events between site A and B, which *TrackingFree* allows is 4,852 and the number of message events between B and C is 6. All of the six messages are communicated between stubhub.com (C) and doubleclick.net (B), one of the largest tracking companies. In sum, among all the 7,971 communications allowed in regular browser, *TrackingFree* only forbids 6 of them.

VII. Related Work

Measurement of Web Tracking. The practice of web tracking has been studied extensively. Mayer and Mitchell give the most comprehensive discussion about third-party tracking, including tracking techniques, business models, defense choices and policy debates [26]. Another important measurement work is proposed by Roesner *et al.*, in which the authors propose a comprehensive classification framework for different web tracking practices [34]. Soltani *et al.* and Ayenson *et al.* measure the prevalence of non-cookie based stateful tracking and show how tracking companies use multiple client-side states to respawn deleted identifiers [35] [4]. Yen *et al.* and Nikiforakis *et al.* discuss the techniques of stateless tracking in their works [41] [32]. In addition to tracking behaviors and techniques, Krishnamurthy *et al.* [20] [19] [18] [21] focus on the risk of harm resulted from web tracking, showing that not only user's browsing history, but also other sensitive personal information, such as name and email, can be leaked out.

Existing Anti-tracking Mechanisms. Although web tracking has garnered much attention, no effective defense system has been proposed. Most commercial anti-tracking tools ([2], [13], [30]) are based on blacklists, which will leave user unprotected were the trackers not collected by their databases; Roesner *et al.* [34] proposed a tool called *ShareMeNot*, but it can only defeat against social media button tracking, a small subset of tracking practices. Private browsing mode [40] significantly affects user experience as user can not persistently save anything on client-side state and users can still be tracked before closing the browser. The *Do Not Track(DNT)* [39] header and legislation require tracker compliance and cannot effectively protect users from tracking in reality [34] [26]. Disabling third-party cookie, which is supported by most browsers [29] [14] [27], can be easily bypassed through non-cookie-based tracking approaches [17] [34] [35] [4] and also suffer from compatibility issues[5]. According to existing work [34], among all the 476 cross-site trackers found in Alexa Top 500 web sites, DNT can only block 27 (5.7%) of them, while disabling third-party cookie can block 367 (77.1%) of them.

Existing Multi-principal Browsers. A number of researchers and companies have proposed browsers with different isolation mechanisms [5] [28] [15] [38] [10] [15] [7] [6] [23] [8] [12] or utilities that facilitate browsing with multiple browsers [11] [25]. Gazelle [38], WebShield [23], Chromium [5], Internet Explorer 8 (IE8) [28], VirtualBrowser [6], COP [7] and OP browser [15] adopt different isolation policies, but

[5]For example, when disabling third-party cookies, user will fail login sears.com with Facebook.com account

Browser	Isolation Mechanism	Contents Allocation Mechanism	Stateful Anti-tracking Capability
Chromium [5]	In-memory Isolation	Top-frame based	No
COP [7]	In-memory Isolation	Server Configuration based	No
Gazelle [38], WebShield [23]	In-memory Isolation	Same-origin-policy (SOP) based	No
IE8 [28]	In-memory Isolation	Tab based	No
OP [15]	In-memory Isolation	Web Page based	No
VirtualBrowser [6]	In-memory Isolation	User Configuration based	No
AppIsolation [8]	Technique-specific Storage	User Configuration based	Incomplete
Fluid [11], MultiFirefox [25]	Profile	User Configuration based	Incomplete
Stainless [12]	Technique-specific Storage	User Configuration based	Incomplete
Tahoma [10]	Virtual Machine	User Configuration based	Incomplete
TrackingFree	Profile	In-degree-bounded Graph based	**Complete**

Table V: Multi-principal Browser Anti-tracking Capability Comparison

only isolate browser's in-memory state. AppIsolation [8] and Tahoma [10], aiming to protect sensitive web applications from untrusted ones, isolate sensitive application's in-memory and persistent states. AppIsolation adopts a light-weighted isolation mechanism implemented on Chromium, while Tahoma utilizes heavy-weighted virtual machines (VM) to isolate principals. Stainless [12] is another multi-process browser but with a feature of parallel sessions. Fluid [11] and MultiFirefox [25] are two popular site-specific browser utilities, facilitating user to visit different sites with different browsers. We compare these works' anti-tracking abilities with *TrackingFree* from two aspects: 1). isolation mechanism and 2). contents allocation policy. Table V lists the comparison details.

Tab based contents allocation policy is adopted by IE8. When a cross-site navigation occurs, this policy will not switch principal and allow multiple web sites using the same principal. Chromium's process-per-site mode adopts top-frame based policy, in which all the web pages and their contents, including third-party iframes, are kept in the same principal if and only if they are from the same site. SOP based policy, adopted by Gazelle and WebShield, strictly follows same-origin policy and puts all the child frames with different origins into different principals. We formally evaluate the anti-tracking ability of top-frame based and SOP based contents allocation policy with *Alloy* and the results show that they cannot protect user's privacy: in the worst case, tracker can correlate user's activities on unlimited number of web sites (see Section VI for details). User configuration based and server configuration based allocation mechanism require user or server to manually define allocation policy. The most privacy-preserving configuration a user can achieve is to assign each site a principal, similar to top-frame based allocation mechanism and therefore suffering from the same privacy issue. Server configuration based mechanism requires server-side collaboration, which is unlikely to happen. OP browser, adopting web page based policy, will always open a new web page instance whenever navigation occurs, which is not necessary and suffers from serious compatibility issues.

As for isolation mechanism, most browsers do not isolate client-side state and cannot prevent web tracking at all. Technique-specific storage (TSS) mechanism, adopted by AppIsolation and Stainless, can isolate some or even all of web data storages (*e.g.,* cookie jar, HTML local storage, cache store), but it can still be bypassed when trackers abuse those client-side state that do not belong to web data storages, (*e.g.*, user preference) [22]. Among the browsers or utilities that adopt complete isolation mechanisms, *i.e.,* profile and virtual machine, *TrackingFree* is the only one that allows principal communication.

VIII. Conclusion

In this paper, we design and implement *TrackingFree*, the first anti-tracking browser that can completely block stateful third-party tracking practice. To strike a balance between anti-tracking ability and compatibility, we propose a novel content allocation mechanism and a secure principal communication channel. Our evaluation shows that *TrackingFree* can block all the 647 trackers we found in Alexa Top 500 web sites with affordable overhead. Moreover, we formally prove *TrackingFree*'s anti-tracking ability with *Alloy*, theoretically showing that even in the worst scenario, trackers can at most correlate *TrackingFree* user's online behaviors on three sites.

Reference

[1] Are We Private Yet? An Open Source Project. http://www.areweprivateyet.com/.

[2] Abine Inc. Donottrackme. https://www.abine.com/dntdetail.php.

[3] Devdatta Akhawe, Adam Barth, Peifung E Lam, John Mitchell, and Dawn Song. Towards a formal foundation of web security. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, pages 290–304. IEEE, 2010.

[4] Mika Ayenson, Dietrich Wambach, Ashkan Soltani, Nathan Good, and Chris Hoofnagle. Flash cookies and privacy ii: Now with html5 and etag respawning. *Available at SSRN 1898390*, 2011.

[5] Adam Barth, Collin Jackson, Charles Reis, and TGC Team. The security architecture of the chromium browser, 2008.

[6] Yinzhi Cao, Zhichun Li, Vaibhav Rastogi, Yan Chen, and Xitao Wen. Virtual browser: a virtualized browser to sandbox third-party javascripts with enhanced security. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 8–9. ACM, 2012.

[7] Yinzhi Cao, Vaibhav Rastogi, Zhichun Li, Yan Chen, and Alexander Moshchuk. Redefining web browser principals with a configurable origin policy. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2013.

[8] Eric Yawei Chen, Jason Bau, Charles Reis, Adam Barth, and Collin Jackson. App isolation: get the security of multiple browsers with just one. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 227–238. ACM, 2011.

[9] Chromium. Creating and using profiles. http://www.chromium.org/developers/creating-and-using-profiles.

[10] R.S. Cox, J.G. Hansen, S.D. Gribble, and H.M. Levy. A safety-oriented platform for web applications. In *IEEE Symposium on Security and Privacy*. Citeseer, 2006.

[11] Todd Ditchendorf. Fluid: Turn your favorite web apps into real mac apps. http://fluidapp.com/.

[12] Danny Espinoza. Stainless: A multi-process browser alternative to google chrome. http://www.stainlessapp.com/.

[13] Ghostery. Ghostery. http://www.ghostery.com/.

[14] Google. Manage your cookies and site data. https://support.google.com/chrome/answer/95647?hl=en.

[15] Chris Grier, Shuo Tang, and Samuel T King. Secure web browsing with the op web browser. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 402–416. IEEE, 2008.

[16] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

[17] Samy Kamkar. Evercookie. http://samy.pl/evercookie/.

[18] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Web 2.0 Security and Privacy Workshop*, 2011.

[19] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web*, pages 541–550. ACM, 2009.

[20] Balachander Krishnamurthy and Craig E Wills. Generating a privacy footprint on the internet. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 65–70. ACM, 2006.

[21] Balachander Krishnamurthy and Craig E Wills. Characterizing privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, pages 37–42. ACM, 2008.

[22] Teddy Leva. Sts design questions around probing what sites a user has been to. http://code.google.com/p/chromium/issues/detail?id=33445.

[23] Zhichun Li, Yi Tang, Yinzhi Cao, Vaibhav Rastogi, Yan Chen, Bin Liu, and Clint Sbisa. Webshield: Enabling various web defense techniques without client side modifications. In *NDSS*, 2011.

[24] ROBERT S Liverani and Nick Freeman. Abusing Firefox Extensions. *Defcon17, July*, 2009.

[25] Dave Martorana. Multifirefox. http://davemartorana.com/multifirefox/.

[26] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 413–427. IEEE, 2012.

[27] Microsoft. Description of cookies. http://support.microsoft.com/kb/260971.

[28] Microsoft. IE8 Security Part V: Comprehensive Protection.

[28] Microsoft. IE8 Security Part V: Comprehensive Protection. http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-v-comprehensive-protection.aspx?Redirected=true.

[29] Mozilla. Disable third-party cookies in firefox to stop some types of tracking by advertisers. http://support.mozilla.org/en-US/kb/disable-third-party-cookies.

[30] Mozilla. Lightbeam for Firefox. http://www.mozilla.org/en-US/lightbeam/.

[31] Mozilla. Use the Profile Manager to Create and Remove Firefox Profiles. https://support.mozilla.org/en-US/kb/profile-manager-create-and-remove-firefox-profiles.

[32] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2013.

[33] M Perry, E Clark, and S Murdoch. The design and implementation of the tor browser [draft][online], united states, 2011.

[34] Franziska Roesner, Tadayoshi Kohno, and 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*, NSDI'12, pages 12–12, Berkeley, CA, USA, 2012. USENIX Association.

[35] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[36] The World Wide Web Consortium. W3C editor's draft: HTML5. http://www.w3.org/html/wg/drafts/html/master/browsers.html#unit-of-related-browsing-contexts.

[37] W3C. HTTP ETag. http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

[38] Helen J Wang, Chris Grier, Alexander Moshchuk, Samuel T King, Piali Choudhury, and Herman Venter. The multi-principal os construction of the gazelle web browser. In *USENIX Security Symposium*, pages 417–432, 2009.

[39] Wikipedia. Do Not Track Policy. http://en.wikipedia.org/wiki/Do_Not_Track_Policy.

[40] Wikipedia. Privacy Mode. http://en.wikipedia.org/wiki/Privacy_mode.

[41] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martın Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *Proceedings of NDSS*, 2012.