Privacy-Preserving Social Plugins

Georgios Kontaxis,[†] Michalis Polychronakis,[†] Angelos D. Keromytis,[†] Evangelos P. Markatos[*]
[†]*Columbia University,* [*]*FORTH-ICS*
{kontaxis,mikepo,angelos}@cs.columbia.edu, markatos@ics.forth.gr

Abstract

The widespread adoption of social plugins, such as Facebook's Like and Google's +1 buttons, has raised concerns about their implications to user privacy, as they enable social networking services to track a growing part of their members' browsing activity. Existing mitigations in the form of browser extensions can prevent social plugins from tracking user visits, but inevitably disable any kind of content personalization, ruining the user experience.

In this paper we propose a novel design for *privacy-preserving social plugins* that decouples the retrieval of user-specific content from the loading of a social plugin. In contrast to existing solutions, this design preserves the functionality of existing social plugins by delivering the same *personalized* content, while it protects user privacy by avoiding the transmission of user-identifying information at load time. We have implemented our design in *SafeButton*, an add-on for Firefox that fully supports seven out of the nine social plugins currently provided by Facebook, including the Like button, and partially due to API restrictions the other two. As privacy-preserving social plugins maintain the functionality of existing social plugins, we envisage that they could be adopted by social networking services themselves for the benefit of their members. To that end, we also present a pure JavaScript design that can be offered transparently as a service without the need to install any browser add-ons.

1 Introduction

Social plugins enable third-party websites to offer personalized content by leveraging the social graph, and allow their visitors to seamlessly share, comment, and interact with their social circles [12]. For example, Facebook's Like button, probably the most widely deployed social plugin [33], enables users to leave positive feedback for the web page in which it has been embedded, share the page with their social circle, and view their like-minded friends. Google's "+1" button [16] offers almost identical features to the Like button, while similar widgets are also available from other popular social networking services (SNSs) such as Twitter and LinkedIn.

Social plugins offer multifaceted benefits to both content providers and members of SNSs, a fact that is reflected by the tremendous growth in their adoption. Indicatively, as of June 2012, more than two million websites have incorporated some of Facebook's social plugins, while more than 35% of the top 10,000 websites include Like buttons—a percentage three times higher than just one year ago [33]. Unfortunately, as the number of websites that incorporate social plugins increases, so does the portion of their visitor's browsing history that gets exposed.

To personalize the content of third-party web pages, social plugins connect to the SNS and transmit a unique user identifier—usually contained in an HTTP cookie—along with the URL of the visited page. Consequently, the SNS receives detailed information about every visit of its members to any page with embedded social plugins. Considering the increasing adoption rate of social plugins, a constantly growing part of its members' browsing history can be precisely tracked.

More importantly, the cookies used in social plugins are linked to user profiles that typically contain the person's name, email address, and other private information. Although third-party tracking cookies as used by advertising networks and traffic analytics services also aim to track the pages visited by a specific user [43], in essence they track the pages opened using a particular browser instance running on a device with a given IP address. While this can already be considered as personally identifying information to some extent, in addition to that information, social plugins reveal much more: the browsing history of *individuals*.

The important implications of social plugins to user privacy were identified soon after their release [34, 54], and concerns have been intensifying [34, 37]. As avoid-

ing becoming a member of any SNS is often rather difficult (even users that are not interested in the social aspects of a service can be affected, e.g., Gmail users can still be tracked through Google's "+1" buttons), privacy-conscious users can resort to browser extensions that block user-identifying information from reaching the SNS through social plugins [27, 15, 7, 4, 28, 45].

Depending on the subtlety of their approach, ranging from stripping cookies and headers from the plugin's requests to preventing the plugin from loading, some or none of its user interaction functionality may be preserved. However, as user-identifying information never reaches the SNS, all these solutions completely disable any kind of content personalization. As an example, for a Like button, even logged in members will be viewing just the total number of "likes" for the page (Fig. 1a), instead of the names and pictures of their friends who have liked the page (Fig. 1c).

We believe that the majority of users are not even aware of the privacy issues stemming from the prevalence of social plugins. For this reason, we argue that any solution can be effective only if it can be deployed by SNSs themselves, so as to protect *all* users without requiring any action on their behalf. Crucially, content personalization and user interaction are two key features of existing social plugins. Any solution that lacks either of them, or introduces even a slight compromise in user experience, is not likely to be adopted by SNSs.

Driven by these two observations, in this paper we propose a novel design for *privacy-preserving social plugins*, which fulfills two seemingly contradicting goals: it protects user privacy by avoiding the transmission of user-identifying information at load time, while it offers identical functionality to existing social plugins by providing the same personalized content. The main idea is to decouple the retrieval of private information from the loading of a social plugin by prefetching all data from the user's social circle that might be needed in the context of a social plugin. Any missing non-private data is retrieved on demand without revealing the identity of the user to the SNS. Local (private) and server-side (public) data are then combined to render a pixel-by-pixel identical version of the same personalized content that would have been rendered by existing social plugins.

To demonstrate the feasibility of our design, we have implemented *SafeButton*, an add-on for Firefox that provides privacy-preserving versions of existing social plugins, as they are provided by the major SNSs. Based on our experimental evaluation, the local disk space consumed by SafeButton for storing the private data required for handling the nine different social plugins currently provided by Facebook is in the order of a few megabytes for typical users, and 145MB for the extreme case of a user with 5,000 friends. At the same time, due to reduced network overhead, SafeButton renders social plugins 64% faster compared to their original versions. Our design can be readily adopted by existing SNSs, and be offered transparently as a service to their members without the need to install any additional software.

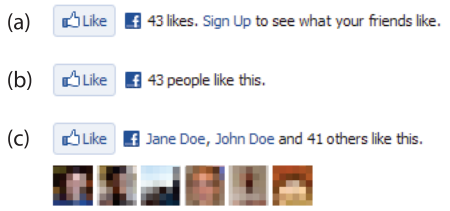

Figure 1: Different states of Facebook's Like button for a user that (a) has never logged in on Facebook from this particular browser or is not a member of Facebook at all, (b) has previously logged in but is currently logged out, (c) is currently logged in (personalized view).

Our work makes the following main contributions:

- We propose a novel design for privacy-preserving social plugins that i) prevents the SNS from tracking its members' browsing activities, and ii) provides the same functionality as existing social plugins with no compromises in content personalization.
- We have implemented SafeButton, a Firefox extension that currently provides privacy-preserving versions of Facebook's social plugins.
- We evaluate our implementation and demonstrate the feasibility of the proposed design in terms of functionality, effectiveness, and performance.
- We describe in detail a pure JavaScript implementation of our design that can be offered by existing SNSs as a transparent service to their members.

2 User Tracking through Social Plugins

2.1 Social Plugins

Social plugins are provided by the major social networking services in the form of "widgets" that can be embedded in any web page, usually in the form of an IFRAME element. After downloading the page, the browser issues a subsequent request to fetch and load the content of the plugin, as shown in Fig. 2 (step 2). The domain that serves the social plugins is the same as the one that hosts the SNS itself, and thus any state that the browser maintains for the SNS in the form of HTTP cookies [17] is transmitted along with the request for the social plugin.

Assuming the user has an active session with the SNS, the site will associate the request with the user's profile,

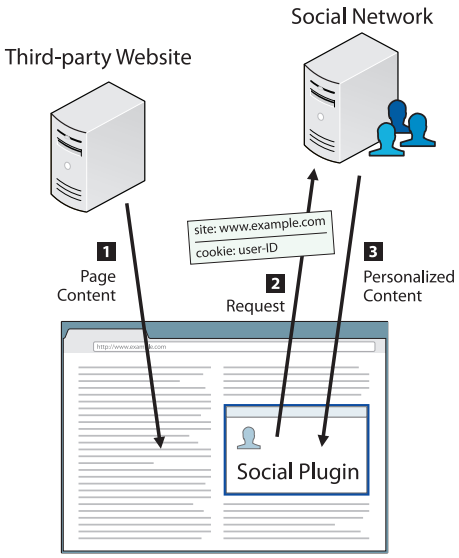

Figure 2: Loading phase of social plugins. After a page is fetched (1), the browser loads the IFRAME of the social plugin (2). If the user is logged in on the SNS, the plugin receives and displays personalized information (3). Users are identified (and can be tracked) through the HTTP cookies included in the request.

and respond with personalized content tailored to that particular user and visited web page (step 3 in Fig. 2). Otherwise, if the user has not logged in on the SNS from that particular browser before, or has never registered at all, the social plugin will display only generic, publicly accessible information for that page.

For instance, Fig. 1 shows the different modes of the Like button depending on the browser's cookies for `facebook.com`. If a user does not have an account or has not logged in on Facebook using that browser, the plugin displays only the total number of "likes" and prompts the user to sign up (a). If a user is currently logged in, the plugin displays personalized information, including some of the names and pictures of the user's friends that have liked the page (c). Interestingly, while a user is logged out (b), the plugin does not prompt for sign-up; depending on how cookies are cleared, some user-identifying information may persist even upon user exit.

2.2 Privacy Issues

With publishers reporting multifold increases in traffic [35], and the continuous addition of new gestures and social features by the major social networking services [36], it is expected that the explosive popularity of social plugins will only continue to grow. As more sites employ social plugins, the potential for broader user tracking increases. With more than 35% of the top 10,000 most visited websites having Like buttons in their pages (as of June 2012) [33], a good part of the daily browsing history of 901 million active Facebook users [8] is technically available to Facebook. We should stress that the same issue holds for all other major social networking services that provide social plugins, including Google and Twitter.

The privacy issues related to the use of HTTP cookies are a well-known problem. Since their introduction in 1995, cookies have been extensively used by advertising networks for building user profiles and tracking the browsing activity of users across the web [51]. Although user tracking through social plugins resembles this kind of cross-site tracking through third-party cookies [43], there is one key difference.

An advertising network uses cookies to track the same user across all affiliate sites that host the network's advertisements, but cannot easily link the derived activity pattern to the actual identity of the user. In contrast, social plugins use cookies associated with real user profiles on the respective social networking site, which typically contain an abundance of personally identifiable information [47]. In essence, instead of tracking anonymous users, social plugins enable tracking of named *persons*.

Advertising agencies can also potentially associate a user profile with a person's identity by combining information from other sources, e.g., in cooperation with one or more affiliate websites on which users provide contact information for registration. Social networking services, though, do not have to collude with another party because they already have access to both extensive personally identifiable information, as well as to a broad network of sites that host social plugins.

2.3 Preventing Privacy Leaks

One might think that if users diligently log out of the social networking site, they will be safe from the privacy leaks caused by its social plugins. Unfortunately, this seems a rather daunting task for users that rely daily on Google, Facebook, Twitter, and other popular SNSs for their personal and professional communication and social interaction activities. To provide convenience for frequent use, these sites follow a single sign-on approach for all offered services, and prompt users to stay logged in indefinitely through "keep me logged in" features. Consequently, users typically remain logged in throughout the whole duration of their online presence.

In some cases, even after a user logs out, the cookies of the SNS might not be cleared completely, and personally identifiable information may still persist [9]. For example, even after logging out of Facebook, a cookie with a user identifier remains in the browser, enabling features such as pre-filling a returning user's email address in the

log in form, or avoiding to unnecessarily prompt existing members to sign up, as shown in Fig. 1(b).

Blocking of third-party cookies could be considered a mitigation to this problem, since most of the major web browsers (Chrome, Firefox, Internet Explorer) have adapted their security policy to prevent third parties from reading (in addition to writing) cookies. Therefore, even though the SNS's domain appears both as a first party (when a user visits the site directly) and as a third party (when a social plugin is embedded in a page), in the latter case the SNS no longer receives any cookies. However, with the exception of Internet Explorer, blocking of third-party cookies is not enabled by default. Internet Explorer will do so, but white-lists same-domain cookies set by first parties that return a P3P header [30] (even a dummy one), which both Facebook and Google [25] appear to be doing. Moreover, even if a user chooses explicitly to block third-party cookies, there are known bypass techniques [2], such as faking an interaction with the embedded page through a script-initiated form submission in Safari, or opening the embedded page in a pop-up window that gets treated by the browser as a first party [53], which interestingly in Chrome is not hindered by pop-up blocking [3].

The Do Not Track HTTP header [5] is an encouraging recent initiative that allows users to opt out of tracking by advertising networks and analytics services. Although currently not supported by any SNS, if it were adopted, Do Not Track could allow users to choose whether they want to opt in for the personalized versions of social plugins or not. However, users who would opt in for the personalized versions (or who would not opt out, depending on the default setting) could still be tracked.

This situation drives privacy-conscious users towards browser extensions that block the transmission of user-identifying information through social plugins [27, 15, 7, 4, 28, 45]. For instance, Facebook Blocker [7] removes completely the IFRAME elements of social plugins from visited web pages. Instead of blocking social plugins completely, ShareMeNot [28] simply removes the sensitive cookies from the social plugin's requests at load time. When a user explicitly interacts with a plugin, the cookies are then allowed to go through, enabling the action to complete normally. Although this approach strikes a balance between usability and privacy, it still completely disables any content personalization.

3 Design

3.1 Requirements

The design of privacy-preserving social plugins is driven by two key requirements: *i) provide identical functionality to existing social plugins in terms of content personalization and user interaction*, and *ii) avoid the transmission of user-identifying information to the social networking service before any user interaction takes place*. The first requirement is necessary for ensuring that users receive the full experience of social plugins, as currently offered by the major SNSs. Existing solutions against user tracking do not provide support for content personalization, and thus are unlikely to be embraced by SNSs and content providers. The second requirement is mandatory for preventing SNSs from receiving user-identifying information whenever users merely view a page and do not interact with a social plugin.

We consider as user-identifying information any piece of information that can be used to *directly* associate a social plugin instance with a user profile on the SNS, such as a cookie containing a unique user identifier. The IP address of a device or a browser fingerprint can also be considered personally identifying information, and could be used by a shady provider for user tracking. However, the accuracy of such signals cannot be compared with the ability of directly associating a visit to a page with the actual *person* that visits the page, due to factors that introduce uncertainty [52], such as DHCP churn, NAT, proxies, multiple users using the same browser, and other aspects that obscure the association of a device with the actual person behind it. Users can mitigate the effect of these signals to their privacy by browsing through an anonymous communication network [38], and ensuring that their browser has a non-unique fingerprint [39].

When viewed in conjunction, the two requirements seem contradicting. Content personalization presumes knowledge of the person for whom the content will be personalized. Nevertheless, the approach we propose satisfies both requirements, and enables a social plugin instance to render personalized content without revealing any user-identifying information to the SNS.

3.2 Overall Approach

Social plugins present the user with two different types of content: *private* information, such as the names and pictures of friends who like a page, and *public* information, such as the total number of "likes." The main idea behind our approach is to maintain a local copy of all private information that can possibly be needed for rendering any personalized content for a particular user, and query the social networking service only for public information that can be requested anonymously.

This approach satisfies our first requirement, since all the required private information for synthesizing and presenting personalized content is still available to the social plugin locally, while any missing public information can be fetched on demand. User interaction is not hindered in any way, as user actions are handled in the same way

as in existing social plugins. Our second requirement is also accomplished, because all communication of a privacy-preserving social plugin with the SNS for loading its content *does not* include any user-identifying information. Only public information about the page might be requested, which can be retrieved anonymously.

The whole process is coordinated by the *Social Plugin Agent*, which runs in the context of the browser and has three main tasks: i) upon first run, gathers all private data that might be needed through the user's profile and social circle, and stores it in a local *DataStore*, ii) periodically, synchronizes the DataStore with the information available online by adding or deleting any new or stale entries, and iii) whenever a social plugin is encountered, synthesizes and presents the appropriate content by combining private, personalized information from the local DataStore and public, non-personalized information through the SNS. Maintaining a local copy of the user's social information is a continuous process, and takes place transparently in the background. Once all necessary information has been mirrored during the bootstrapping phase, the DataStore is kept up to date periodically.

Going back to the example of the Like button, the private information that must be stored locally for its privacy-preserving version should suffice for properly rendering *any possible* instance of its personalized content for *any third-party page the user might encounter*. This can be achieved by storing locally all the "likes" that all of the user's friends have ever made, as well as the names and thumbnail pictures of the user's friends. Note that all the above information is available through the profile history of the user's friends, which is always accessible while the user is logged in.

Although keeping all this state locally might seem daunting at first, as we demonstrate in Sec. 5.2, the required space for storing all the necessary private information for privacy-preserving versions of *all* Facebook's existing social plugins is just 5.4MB for the typical case of a user with 190 friends, and 145MB for an extreme case of a user with 5,000 friends. No information that is not accessible under the user's credentials is ever needed, and daily synchronization typically requires the transmission of a few kilobytes of data.

Continuing with the Like button as an example, Fig. 3 illustrates the process of rendering its privacy-preserving version. Upon visiting a third-party page, the Social Plugin Agent requests from the SNS the total number of "likes" for that particular page, without providing any user-identifying information (step 3). In parallel, it looks up the URL of the page in the DataStore and retrieves the names and pictures of the friends that have liked the page (if any). Once the total number of "likes" arrives (step 4), it is combined with the local information and the unified personalized content is presented to the user (5).

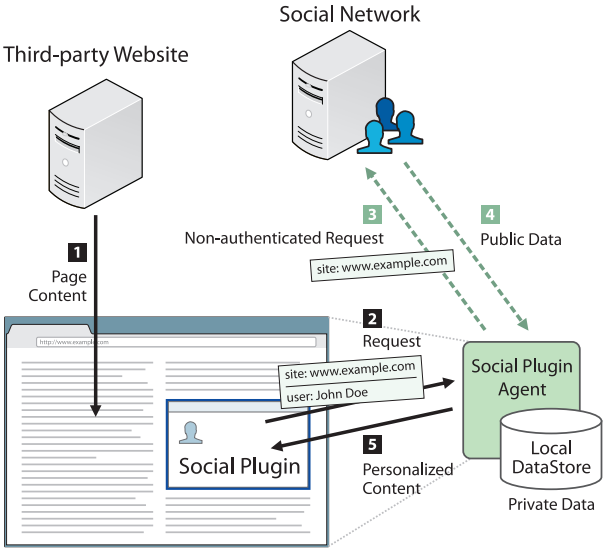

Figure 3: The loading phase of privacy-preserving social plugins. When a social plugin is encountered (1), the Social Plugin Agent intervenes between the plugin and the SNS (2). The agent requests (3) and receives (4) only publicly accessible content, e.g., the page's total number of "likes," without revealing any user-identifying information to the SNS. The agent then combines this data with personalized information that is maintained locally, and presents the unified content to the user (5).

Further optimizations are possible for avoiding querying for non-personalized content at load time. Depending on the plugin and the kind of information it provides, public information for frequently visited pages can be cached, while public information for highly popular pages can be prefetched. For example, information such as the total number of "likes" for a page that a user visits several times a day can be updated only once per day without introducing a significant inconsistency, allowing the Social Plugin Agent to occasionally serve the Like button using *solely* local information. Similarly, the SNS can regularly push to the agent the total number of "likes" for the top 10K most "liked" pages. In both cases, the elimination of any network communication on every cache hit not only reduces the rendering time, but also protects the user's browsing pattern even further.

4 Implementation

To explore the feasibility of our approach we have implemented *SafeButton*, an add-on for Firefox (version 7.0.1) that provides privacy-preserving versions of existing social plugins. SafeButton is written in JavaScript and XUL [23], and relies on the XPCOM interfaces of Firefox to interact with the internals of the browser. Figure 4

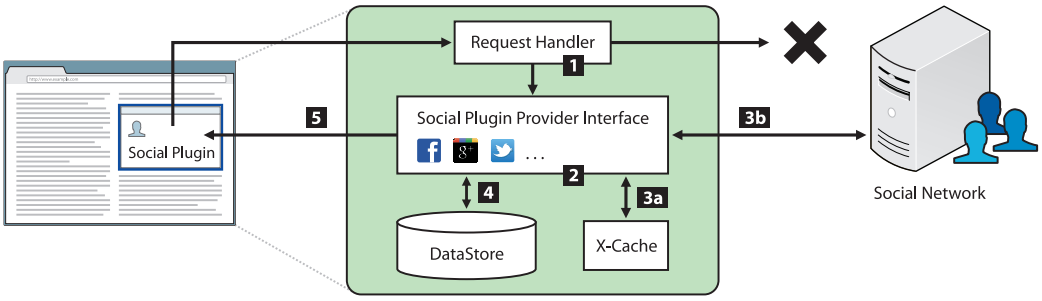

Figure 4: Overall architecture of SafeButton. A Request Handler (1) intercepts the HTTP requests of social plugins. Privacy-preserving implementations of the supported plugins (2) combine public remote data (3b), which can be cached in the X-Cache for improving network performance (3a), and private data from the user's social circle, which are maintained locally in the DataStore (4), and deliver the same personalized content (5) as the original plugins.

provides an overview of SafeButton's main components, which are described below. A detailed description of how the components are put together to handle a Like button is provided at the end of this section.

Request Handler The main task of the Request Handler is to intercept the HTTP requests of a social plugin at load time, and hand off the event to an appropriate callback handler function. The requests are intercepted using a set of filters based on signatures that capture the target URL of each plugin. These signatures are received from the Social Plugin Provider Interface, along with the callback handlers that should be invoked whenever a filter is triggered. The Request Handler provides as an argument to these callbacks a reference to the DOM of the page that contains the social plugin that triggered the filter.

We have implemented the Request Handler by registering an observer for HTTP requests (`http-on-modify-request` notification) using XPCOM's `nsIObserverService`. This allows the inspection code to lie inline in the HTTP request creation process, and either intercept and modify requests (e.g., by stripping HTTP cookies or other sensitive headers), or drop them entirely when necessary.

Social Plugin Provider Interface The Social Plugin Provider Interface serves as an abstraction between the Request Handler and different *Provider Modules* that support the social plugins offered by different social networking services. This extensible design enables more networks and plugins to be supported in the future. In the current version of SafeButton, we have implemented a Provider Module for the social plugins offered by Facebook. We take advantage of the Graph API [10] to download the user's private social information that needs to be stored locally, and access any other public content on demand. We should stress that, although an option, we do not employ any kind of web scraping to acquire information from pages accessible through the user's profile.

A Provider Module for a SNS consists of: i) the signatures that will be used by the Request Handler for intercepting the HTTP requests of the platform's social plugins, ii) the callback handler functions that implement the core functionality of each social plugin based on local and remote social information, and iii) the necessary logic for initializing the DataStore and keeping it up to date with the information that is available online.

Each callback function implements the core functionality for rendering a particular social plugin. Its main task is to retrieve the appropriate private social data from the DataStore, request any missing public data from the SNS (without revealing any user-identifying information), and compile the two into the personalized content that will be displayed. The function then updates the DOM of the web page through the page reference that was passed by the Request Handler.

DataStore The DataStore keeps locally all the private social data that might be required for rendering personalized versions of any of the supported social plugins. All information is organized in a SQLite database that is stored in the browser's profile folder for the user that has installed SafeButton. Upon first invocation, SafeButton begins the process of prefetching the necessary data. This process takes place in the background, and relies on the detection of browser idle time and event scheduling to operate opportunistically without interfering with the user's browsing activity.

In our implementation for Facebook, data retrieval begins with information about the user's friends, including each friend's name, thumbnail picture, and unique identifier in Facebook's social graph. Then, for each friend, SafeButton retrieves events of social activity such as the pages that a friend has liked or shared, starting with the oldest available event and moving onward. In case the

download process is interrupted, e.g., if the users turns off the computer, it continues from where it left off the next time the browser is started.

Updating the DataStore is an incremental process that takes place periodically. Fortunately, the current version of the Graph API offers support for incremental updates. As we need to query for any new activity using a separate request for each friend (a Graph API function for multiple user updates would be welcome), we do so gracefully for each friend every two hours, or, if the browser is not idle, in the next idle period. We have empirically found the above interval to strike a good balance between the timeliness of the locally stored information and the incurred network overhead. In our future work, we plan to employ a more elaborate approach based on an exponential backoff algorithm, so that a separate adaptive update interval can be maintained for different friend groups according to their "chattiness."

Note that we also need to address the consistency of the locally stored data with the corresponding data that is available online. For instance, friends may "like" a page and later on "unlike" it, thereby deleting this activity from their profile. Unfortunately, the Graph API currently does not offer support for retrieving any kind of removal events. Nevertheless, SafeButton periodically fetches the entire set of activities for each friend (at a much slower pace than the incremental updates), and removes any stale entries from the DataStore.

X-Cache The *X-Cache* holds frequently used public information and meta-information, such as the total number of "likes" for a page or the mapping between page URLs and objects in the Facebook graph. A hit in the X-Cache means that no request towards the social networking service is necessary for rendering a social plugin. This improves significantly the time it takes for the rendering process to complete, and at the same time does not reveal the IP address of the user to the SNS.

Use Case: Facebook Like Button Here we enrich the running case of the Facebook Like button from Sec. 3 with the technical details of the behavior of SafeButton's components, as shown by the relevant steps in Fig. 4.

Upon visiting a web page with an embedded Like button in the form of an IFRAME, the browser will issue an HTTP request towards Facebook to load and subsequently render the contents of that IFRAME. The Request Handler intercepts this request and attempts to match its URL against the set of signatures of the supported social plugins, which will trigger a match for the regular expression `http[s]?:\/\/www\.facebook\.com\/plugins\/like\.php`. Subsequently, the handler invokes the callback associated with this signature and pass as an argument the plugin's URL and a reference to the DOM of the page that contains the social plugin (step 1).

The first action of the callback function is to query X-Cache for any cached non-personalized information about the button and the page it is referring to. This includes the mapping between the page's URL and its ID in the Facebook graph, along with the global count of users who have "liked" the page (step 3a). In case of a miss, a request made through the Graph API retrieves that information (step 3b). The request is stripped from any Facebook cookies that the browser unavoidably appends to it. The response is then added to X-Cache for future reference. After retrieving the global count of users, the names (and if the developer has chosen so, the thumbnail pictures) of the user's friends that have liked the page are retrieved from the LocalStore (step 4).

Finally, the reference to the DOM of the embedding page (passed by the handler in step 1), is used to update the IFRAME where the original Like button would have been with exactly the same content (step 5).

5 Experimental Evaluation

5.1 Supported Facebook Plugins

In this section we discuss the social plugins offered by Facebook and evaluate the extend to which SafeButton can support them in respect to two requirements: i) user privacy, and ii) support for personalized content. Table 1 lists the nine social plugins currently offered by Facebook. For each plugin, we provide a brief categorization of its "view" functionality, i.e., the content presented to the user according to whether it is based on public (non-personalized) or private (personalized) information, as well as its "on-click" functionality, i.e., the type of action that a user can take.

Although SafeButton interferes with the "view" functionality of existing social plugins, it does not affect their "on-click" functionality, allowing users to interact normally as with the original plugins. As shown in Table 2, SafeButton currently provides complete support for seven out of the nine social plugins currently offered by Facebook.

The Like button and its variation, the Like Box, are fully functional; the count, names, and pictures of the user's friends are retrieved from the DataStore, while the total number "likes" is requested on demand anonymously. The Recommendations plugin presents a list of recommendations for pages from the same site, with those made by friends appearing first. Recommendations from the user's friends are stored locally, so SafeButton can render those that are relevant to the visited site on top. The list is then completed with public recommendations by others, which are retrieved on demand.

Facebook Social Plugin	Public Content	Personalized Content	User Action
Like Button	Total number of people that have liked the page	Names and pictures of friends that have liked the page	Like page
Send Button	-	-	Send content/page URL
Comments	List of user comments	Friends' comments appear on top	Post comment
Activity Feed	List of user activities (likes, comments, shared pages)	Friends' activities appear on top	-
Recommendations	List of user recommendations (likes)	Friends' recommendations appear on top	-
Like Box	Total number of people that have liked the Facebook Page, names and pictures of some of them, list of recent posts from the Page	Names and pictures of friends that have liked the page are shown first	Like page
Registration	-	User's Name, picture, birthday, gender, location, email (prefilled in registration form)	Register
Facepile	-	Names and pictures of friends that have liked the page	-
Live Stream	User messages	-	Post message

Table 1: Public vs. Personalized content in Facebook's social plugins [12].

Facebook Social Plugin	Exposed information during loading		Personalized Content with SafeButton
	Original	SafeButton	
Like Button	IP addr. + cookies	IP addr.	Complete
Send Button	IP addr. + cookies	None	Complete
Comments	IP addr. + cookies	IP addr.	Partial[1]
Activity Feed	IP addr. + cookies	IP addr.	Partial[2]
Recommendations	IP addr. + cookies	IP addr.	Complete
Like Box	IP addr. + cookies	IP addr.	Complete
Registration	IP addr. + cookies	None	Complete
Facepile	IP addr. + cookies	IP addr.	Complete
Live Stream	IP addr. + cookies	IP addr.	Complete

[1] *When all comments are loaded at once, all personalized content is complete. In case they are loaded in a paginated form, some of the friends' comments (if any) might not be shown in the first page.*

[2] *Some of the friends' comments (if any) might be omitted (access to comments is currently not supported by Facebook's APIs).*

Table 2: For 7 out of the 9 Facebook social plugins, SafeButton provides exactly the same personalized content without exposing any user-identifying information.

Similarly to the Like button, Facepile presents pictures of friends who have liked a page, and that information is already present in the DataStore. The Send, Register, and Login buttons do not present any kind of dynamic information, and thus can be rendered instantly without issuing any network request.

Similarly to the Recommendations plugin, content personalization in the Comments plugin consists of giving priority to comments made by friends. SafeButton retrieves the non-personalized version of the plugin, and reorders the received comments so that friends' comments are placed on top. When all comments for a page are fetched at once, the personalized information presented by SafeButton is *fully consistent* with the original version of the plugin. However, when comments are presented in a paginated form, only the first sub-page is loaded. The current version of the Graph API does not support the retrieval of comments (e.g., in contrast to "likes"), and thus in case friends' comments appear deeper than the first sub-page, SafeButton will not show them on top (a workaround would be to download all subsequent comment sub-pages, but for popular pages this would result in a prohibitive amount of data).

The Activity Feed plugin is essentially a wrapper for showing a mix of "likes" and comments by friends, and thus again SafeButton's output lacks any friends' comments. Note that our implementation is based solely on the functionality provided by the Graph API [10], and we refrain from scraping of web content for any missing information. Ideally, future extensions of the Graph API will allow SafeButton to fully support the personalized content of all plugins. We discuss this and other missing functionality that would facilitate SafeButton in Sec. 7.

5.2 Space Requirements

To explore the local space requirements of SafeButton, we gathered a data set that simulates the friends a user may have. Starting with a set of friends from the authors' Facebook profiles, we crawled the social graph and identified about 300,000 profiles with relaxed privacy settings that allow unrestricted access to all profile information, including the pages that person has liked or shared in the past. From these profiles, we randomly selected a set of 5,000—the maximum number of friends a person can have on Facebook [6].

Data	190 Friends	5,000 Friends
Names, IDs of Friends	10.5KB	204.8KB
Photos of Friends	463.4KB	11.8MB
Likes of Friends	4.6MB	126.7MB
Shares of Friends	318.4KB	7.0MB
Total	5.4MB	145.7MB
Average (per friend)	29.2KB	29.7KB

Table 3: Storage space requirements for the average case of 190 friends and the borderline case of 5,000 friends.

To quantify the space needed for storing the required data from a user's social circle, we initialized SafeButton using the above 5,000 profiles. In detail, SafeButton prefetches the names, IDs, and photos of all friends, and the URLs of all pages they have liked or shared. Although we have employed a slow-paced data retrieval process (5sec delay between consecutive requests), the entire process for all 5,000 friends took less than 10 hours. For typical users with a few hundred friends, bootstrapping completes in less than a hour. As already mentioned, users are free to use the browser during that time or shut it down and resume the process later.

Table 3 shows a breakdown of the consumed space for the average case of a user with 190 friends [58] and the extreme case of a user with 5,000 friends, which totals 5.4MB and 145.7MB, respectively. Evidently, consumed space is dominated by "likes," an observation consistent with the prevailing popularity of the Like button compared to the other social plugins. To gain a better understanding of storage requirements for different users, Fig. 5 shows the consumed space as a function of the number of friends, which as expected increases linearly.

We should note that the above results are specific for the particular data set, and the storage space might increase for users with more "verbose" friends. Furthermore, the profile history of current members will only continue to grow as time passes by, and the storage space for older users in the future will probably be larger. Nevertheless, these results are indicative for the overall magnitude of SafeButton's storage requirements, which can be considered reasonable even for current smartphones, while the storage space of future devices can only be expected to increase.

To further investigate the distribution of "likes," the factor that dominates local space, we plot in Fig. 6 the CDF of the number of "likes" of each user in our data set. The median user has 122 "likes," while there are some users with much heavier interaction: about 10% of the users have more than 504 "likes." The total number of "likes" was 1,110,000, i.e., 222 per user on average. This number falls within the same order of magnitude as previously reported statistics, which suggest that there are about 381,861 "likes" per minute on Facebook [31]. With a total population of about 901 million active users [8], this results in about 217 "likes" per user per year. These results indicate that our data set is not particularly biased towards excessively active or inactive profiles.

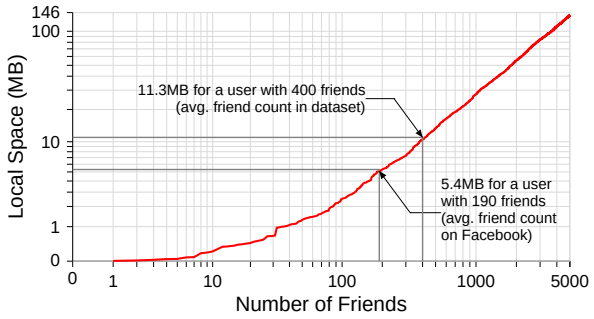

Figure 5: Local space consumption for the required information from a user's social circle as a function of the number of friends. For the average case of a user with 190 friends, SafeButton needs just 5.4MB.

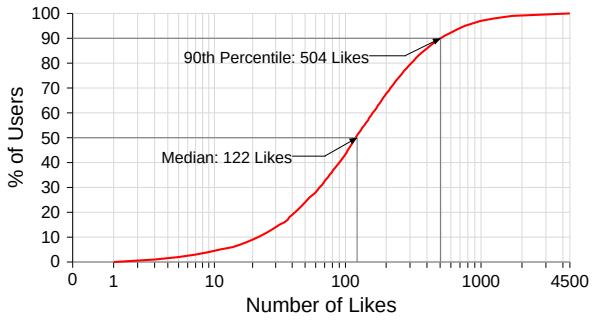

Figure 6: CDF of the number of "likes" of each user.

Besides the storage of social information, SafeButton maintains the X-Cache for quick access to frequently used non-personalized information about a social plugin. To get an estimate about its size requirements, we visited the home pages of the first 1,000 of the top websites according to `alexa.com` that contained at least one Facebook social plugin. About 82.4% of the identified plugins corresponded to a Like Button or Like Box, 14% to Facebook Connect, 3% to Recommendations, 0.5% to Send Button, and 0.1% to Facepile and Activity Box. After visiting all above sites, X-Cache grew to no more than 850KB, for more than 2,500 entries.

5.3 Speed

In this experiment, we explore the rendering time of social plugins with and without SafeButton. Specif-

ically, we measured the time from the moment the HTTP request for loading the IFRAME of a Like button is sent by the browser, until its content is fully rendered in the browser window. To do so, we instrumented Firefox with measurement code triggered by `http-on-modify-request` notifications [20] and `pageshow` events [21]. We chose to measure the rendering time for the IFRAME instead of the entire page to eliminate measurement variations due to other remote elements in the page. This is consistent with the way a browser renders a page, since IFRAMEs are loaded in parallel with the rest of its elements.

We consider the following three scenarios: i) Firefox rendering a Like button unobstructed, and Firefox with SafeButton rendering a Like button when there is ii) an X-Cache miss or iii) an X-Cache hit. For the original Like button, we used a hot browser cache to cancel out loading times for any required external elements, such as CSS and JavaScipt files. Using SafeButton, visiting a newly or infrequently accessed webpage will result in a miss in the X-Cache. For a Like button, this means that besides looking up the relevant information in the local DataStore, SafeButton must (anonymously) query Facebook to retrieve the total number of "likes." For frequently accessed pages, such personalized information will likely already exist in the X-Cache, and thus SafeButton does not place any network request at all.

Using a set of the first 100 among the top websites according to `alexa.com` that contain a Like button, we measured the loading time of the Like button's IFRAME for each site (each measurement was repeated 1,000 times). Figure 7 shows the median loading time across all sites for each scenario, as well as its breakdown according to the events that take place during loading. The rendering time for the original Like button is 351ms, most of which is spent for communication with Facebook. In particular, it takes 130ms from the moment the browser issues the request for the IFRAME until the first byte of the response is received, and another 204ms for the completion of the transfer. In contrast, SafeButton is much faster, as it needs 127ms for rendering the Like button in case of an X-Cache miss (2.8 times faster than the original), and just 24ms in case of an X-Cache hit (14.6 times faster), due to the absence of any network communication.

The difference in the response times for the network requests placed by the original Like button and SafeButton in case of an X-Cache miss can be associated with the different API used and amount of data returned in each case. SafeButton uses the Graph API to retrieve just the total number of "likes," which is returned as a raw ASCII value that is just a few bytes long. In contrast, the original plugin communicates with a different endpoint from the side of Facebook, and fetches a full HTML page with embedded CSS and JavaScript content. While these two

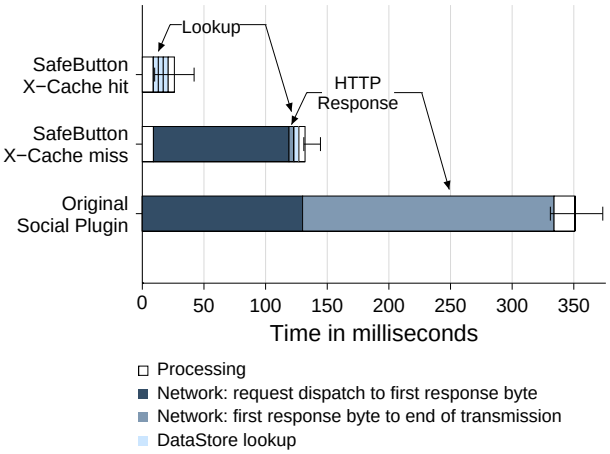

Figure 7: Loading time for Like button with and without SafeButton. Even when the total number of "likes" is not available in the X-Cache, SafeButton is 2.8 times faster.

requests need a similar amount of time from the moment they are placed until the first response byte is received from the server, they differ by two orders of magnitude in terms of the time required to complete the transfer. Even if Facebook optimizes its own plugins in the future, we expect the rendering speed of SafeButton to be comparable in case of an X-Cache miss, and still much faster in case of an X-Cache hit.

5.4 Effectiveness

As presented in Sec. 3, we rely on a set of heuristics that match the target URL of each supported social plugin to intercept and treat them accordingly so as to protect the user's privacy. To evaluate the effectiveness and accuracy of our approach, we carried out the following experiment. Using `tcpdump`, we captured a network trace of all outgoing communication of a test PC in our lab while surfing the web for a week through Firefox equipped with SafeButton. We then inspected the trace and found that no cookie was ever transmitted in any HTTP communication with `facebook.com` or any of its sub-domains.

This was a result of the following "fail-safe" approach. Besides the signatures of the supported social plugins, SafeButton inspects all communication with `facebook.com` and strips any cookies from requests initiated by third-party pages. Next, we performed the reverse experiment: using the same browser equipped with SafeButton, we surfed `www.facebook.com` and interacted with the site's functionality without any issues for a long period. Careful inspection of the log generated by SafeButton proved that no in-Facebook communication was hindered at any time.

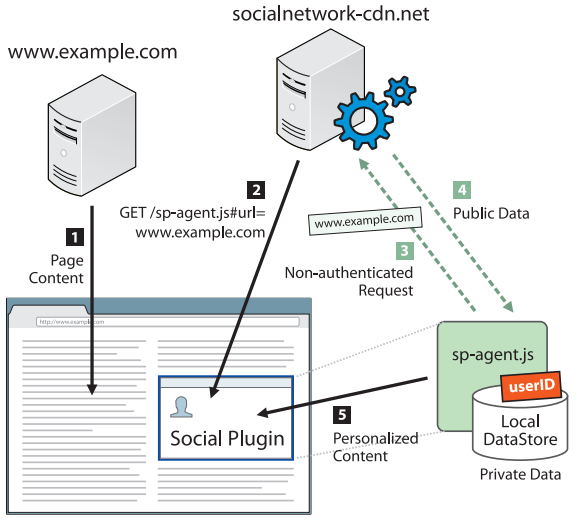

Figure 8: Privacy-preserving social plugins serviced by a SNS. Here: the loading of a social plugin in a third-party page. The code of the social plugin agent is always fetched from a secondary domain to avoid leaking cookies set by the primary domain of the SNS. The URL of the target page is passed via a fragment identifier, so it is never transmitted to the SNS. The agent synthesizes and renders the personalized content of the social plugin.

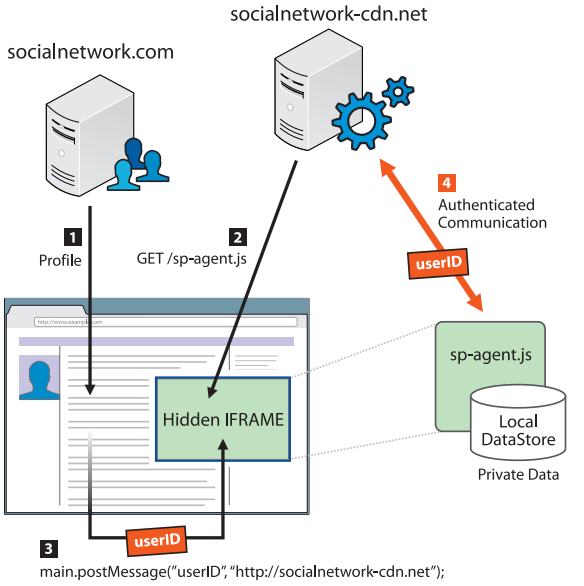

Figure 9: Privacy-preserving social plugins serviced by a SNS. Here: securely communicating the user's session identifier to the social plugin agent when logging in on the SNS. Although the agent is hosted on a secondary domain, it receives and stores the identifier from the primary domain through the `postMessage` function, allowing it to place asynchronous authenticated requests for accessing the user's profile information.

6 Privacy-preserving Social Plugins as a Service: A Pure JavaScript Design

As many users are typically not aware of the privacy issues of social plugins, they are not likely to install any browser extension for their protection. For instance, NoScript [27], a Firefox add-on which blocks untrusted JavaScript code from being executed, has roughly just 2 million downloads, and AdBlock [1], an add-on which prevents advertisement domains from loading as third parties in a web page, has been downloaded 14 million times. At the same time, Firefox has 450 million active users [24], which brings the adoption rate of the above security add-ons to 0.4% and 3.1%, respectively. For this reason, in this section we present a pure JavaScript implementation of privacy-preserving social plugins that could be employed by social networking services themselves for the protection of their members.

The use case would not be much different from now: web developers would still embed an IFRAME element that loads the social plugin from the SNS. However, instead of serving a traditional social plugin, the SNS serves a JavaScript implementation of a social plugin agent in respect to the design presented in Sec. 3. The agent then fetches personalized information from the browser's local storage, requests non-personalized information from the SNS, and renders the synthesized content according to the specified social plugin. The feasibility of the above design is supported by existing web technologies such as IndexedDB [19], which provide a JavaScript API for managing a local database, similar to the DataStore used in SafeButton.

The most challenging aspect of this implementation is to prevent the leakage of user-identifying information during the loading of a social plugin. If the IFRAME of the social plugin agent is hosted on the same (sub)domain as the SNS itself (e.g., `socialnetwork.com`), then the request for fetching its JavaScript code would also transmit the user's cookies for the SNS. At the same time, the agent would need to know the URL of the embedding page for which it has personalized the social plugin's content. If the URL is passed as a parameter to that initial request, the situation is obviously as problematic as in current social plugins.

A solution would be to leave out the URL of the page from the request for loading the social plugin agent. However, there should be a way to communicate this information to the agent once its JavaScript code has been loaded by the browser. This can be achieved through a fragment identifier [32] in the URL from which the agent is loaded. Fragment identifiers come as the last part of a URL, and begin with a hash mark (#) char-

acter. According to the HTTP specification [18], fragment identifiers are never transmitted as part of a request to a server. Thus, during the loading of a social plugin in a third-party page, instead of passing an explicit parameter with the URL of the embedding page, as in `www.socialnetwork.com/sp-agent.js?url=<URL>`, it can be passed through a fragment identifier, as in `www.socialnetwork.com/sp-agent.js#<URL>`. The information about the URL of the visited page never leaves the browser, and remains accessible to the JavaScript code of the agent, which can then parse the hypertext reference of its container and extract the fragment identifier.

Unfortunately, this approach is still not secure in practice. The URL of the embedding page is usually also transmitted as part of the HTTP Referer [sic] header by most browsers. Therefore, even if we omit the target URL from the HTTP parameters of the request, the server will receive it anyway, allowing the SNS to correlate this information with the user's cookies that are transmitted as part of the same request.

To overcome this issue, the social plugin agent can be hosted on a secondary domain, different than the primary domain of the SNS, as also proposed by Do Not Track [5]. For instance, in this design the agent could be hosted under `socialnetwork-cdn.net` instead of `socialnetwork.com`, as shown in Fig. 8. This prevents the browser from appending the user's cookies whenever a social plugin is encountered (step 2), since its IFRAME will be served from a different domain than the one for which the cookies were set. The rest of the steps are analogous to Fig. 3.

Still, the social plugin agent must be able to issue authenticated requests towards the SNS for accessing the user's profile and retrieving the necessary private social information that must be maintained locally. This requires access to the user's cookies, and specifically to the identifier of the authenticated session that the user has with the SNS.

A solution to this problem can be achieved by taking advantage of the `windows.postMessage` [22] JavaScript API, which allows two different origins to communicate. When the user logs in on the SNS, the login page contains a hidden IFRAME loaded through HTTPS from the secondary domain on which the social plugin agent is hosted, as shown in Fig. 9 (step 2). The login page then communicates to the agent's IFRAME the session identifier of the user through `postMessage` (step 3). The IFRAME executes JavaScript code that stores locally the user identifier under its own domain, making it accessible to the plugin agent. The agent can then read the session identifier from its own local storage, and place authenticated requests towards the SNS for accessing the user's profile (step 4) and synchronizing the required information with the locally stored data. When the user explicitly logs out from the social networking site, the log out page follows a similar process to erase the identifier from the local storage of the agent.

In respect to supporting multiple users per browser instance and protecting the personal information stored locally, encryption can be employed to shield any sensitive information, such as the names or identifiers of a user's friends. In accordance with the communication of the session identifier described above, a user-specific cryptographic key can be communicated from the SNS to the social plugin agent. The plugin can then use this key to encrypt sensitive information locally. The key is kept only in memory. Each time the plugin agent loads, it spawns a child IFRAME towards the social networking site. The request for the child IFRAME will normally have the user's cookies appended. Finally, that child IFRAME, once loaded, can communicate via `postMessage` the encryption key back to the plugin agent.

7 Discussion

Strict Mode of Operation Although SafeButton does not send any cookies to the social networking service, it still needs to make non-authenticated requests towards the SNS to fetch public information for some social plugins (e.g., for Facebook plugins, the information shown in column "Public Content" in Table 1). These requests unavoidably expose the user's IP address to the SNS.

Some users might not feel comfortable with exposing their IP address to the SNS (even when no cookies are sent), as this information could be correlated by the SNS with other sources of information, and could eventually lead to the exposure of the users' true identity. For such privacy-savvy users, we consider a "paranoid" mode of operation in which SafeButton does not reveal the user's IP address to the social networking service when encountering a social plugin in a third-party page, by simply not retrieving any public information about the page. Unavoidably, some social plugins are then rendered using solely the locally available personalized information, e.g., for the Like button, the total number of "likes" for the page will be missing.

Alternatively, given the very low traffic incurred by SafeButton's non-authenticated queries to the SNS, these can be carried out transparently by SafeButton through an anonymous communication network such as Tor [38]. Given that social plugins are loaded in parallel with the rest of the page's elements, this would minimally affect the browsing experience (compared to browsing solely through Tor).

Potential Challenges with Future Social Plugins. Although SafeButton currently supports all social plugins

offered by Facebook, and our approach is extensible so as to handle the plugins of other social networking services, we consider two potential challenges with future plugins [44]. First, future personalization functionality could include social information from a user's second degree friends, i.e., the friends of his friends, or rely on the analysis of data from the entire user population of the social network. Second, this type of personalization could involve proprietary algorithms not available to the client-side at run-time.

We believe that our approach could be adapted to support such developments. We find it realistic that such extended analysis will take place offline, and result in the calculation of a product that will be stored and taken into account in real-time during content personalization. Therefore, it will not be necessary to have at the client side neither the analysis algorithms nor the entire dataset. The stored outcome of the analysis, e.g., some extra weight on the social graph or additional meta-data, could be available to through the developer's API, and be taken into account by SafeButton during content personalization. At the same time, the social networking service is not deprived of the data necessary to carry out such analysis. Our approach protects user privacy when accessing the "view" functionality of social plugins, but when users explicitly interact with them, their actions and any corresponding data are transmitted to the SNS.

Profile Management As users may access the web via more than one devices, it reasonable to assume that they will require a practical way to use SafeButton in all of them. Although installing SafeButton on each browser should be enough, this will result to the synchronization of the locally stored information with the SNS for each instance separately. In our future work, we will consider the use of cloud storage for keeping fully-encrypted copies of the local DataStore and X-Cache, and synchronizing them across all the user's browser instances, in the same spirit as existing settings and bookmark synchronization features of popular browsers [29, 14].

Keeping a local copy of private information that is normally accessible only through the social networking service might be considered a security risk, as it would be made readily available to an attacker that gains unauthorized access to the user's system. At that point, though, the attacker would already have access to the user's credentials (or could steal them by installing a keylogger on the compromised host) and could easily gather this information from the SNS anyway.

In any case, users could opt-in for keeping the DataStore encrypted, although this would require them to provide a password to SafeButton (similarly to the above mentioned settings synchronization features). For the pure JavaScript implementation, though, as discussed in Section 6, the cryptographic key can be supplied by the SNS upon user login, making the process completely transparent to the user.

Security in Multi-user Environments We now consider the operation of SafeButton in a multi-user environment where more than one users share the same browser instance. In general, sharing the same browser instance is a bad security practice, because after users are done with a browsing session they may leave sensitive information behind, such as stored passwords, cookies, and browsing history. Ideally, users should maintain their own browser instance or accounts in the operating system.

SafeButton retrieves private information when users are logged in the SNS, and stores it locally even after they log out, as it would be inefficient to erase it every single time. Multiple users are supported by monitoring the current cookies for that domain of the SNS, and serving personalized content only for the user that is currently logged in. Local entries that belong to a user ID that does not match the one currently logged in are never returned. Obviously, users that share the same OS account can access each other's locally stored data, since they are contained in the same DataStore instance, unless they have opted in for keeping their data encrypted, as discussed earlier.

Shortcomings of the Graph API Throughout this paper we have briefly mentioned some obstacles we have encountered, namely shortcomings in the developer API provided by Facebook, in respect to our objective of protecting the user's privacy while maintaining full functionality for the social plugins. We summarize these issues here and discuss how the social networks in general could support us.

User Activity Updates through the API. Currently the Facebook API [10] offers access to the social graph but there is no way to receive updates or "diffs" when something changes. For instance, we retrieve a friend's "likes" through the API, we are also able to fetch only new "likes" from a point forward, but are unable to receive notice when that friend "unlikes." A friend "activity" or "history" function could significantly aid our implementation in keeping an accurate local store.

Accuracy of the Provided Information. Sometimes, the API calls and documentation offered to developers differ slightly from the actual behavior of a plugin when it is offered by the SNS itself [11]. This creates a predicament for developers wishing to replicate the functionality.

Support for All Social Information that is Otherwise Accessible. We consider it reasonable for the API to provide access to information that is accessible via the social plugins offered by the SNS itself or via the profile pages of its users. For instance, there is no API call to

access the comments of a specific user, although they appear in the user's profile page. Scrapping could retrieve them, but this practice is not ideal. Therefore, in our case, we have to resolve to practices that result in reduced accuracy, such as anonymously retrieving a sample of the comments of a page and placing the comments of a user's friends at the top, if present in the sample. Retrieving the entire set of comments could be inefficient for pages with too many comments.

Alternatively, Facebook could provide a call for retrieving just the user IDs of all the commenters, and another call for specifying a set of IDs for which to retrieve the actual comments. In that case, we could hide the IDs of a user's friends among a group of k strangers and request their comments for that page [56].

8 Related Work

Do Not Track [5] is a browser technology which enables users to signal, via an HTTP header, that they do not wish to be tracked by websites they do not explicitly visit. Unfortunately there are no guarantees that such a request will be honored by the receiving site or not.

Krishnamurthy et al. [46] studied privacy leaks in online social networking services. They identified the presence of embedded content from third-party domains in the interactions of a user with the SNS itself and stress that the combination with personal information inside an SNS could pose a significant threat to user privacy.

There has been significant work in the interplay between SNSs and privacy. For example, there has been some focus on protecting privacy in SNS against third-party applications installed in a user's profile within the social network [41, 40, 55]. Facecloak [49] shields a user's personal information from a SNS and any third-party interaction, by providing fake information to the SNS and storing actual, sensitive information in an encrypted form on a separate server. The authors in FlyByNight [48] propose the use of public key cryptography among friends in a SNS so as to protect their information from a curious social provider and potential data leaks.

Recent work has focused on how to support personalized advertisements without revealing the user's personal information to the providing party. Adnostic [57] offers targeted advertising while preserving the user's privacy by having the web browser profile the user, through the monitoring of his browsing history, and inferring his interests. It then downloads diverse content from the advertising server and selects which part of it to display to the user. Similarly, RePriv [42] enables the browser to mine a user's web behavior to infer guidelines for content personalization, which are ultimately communicated to interested sites. Our approach differs in principle as the model of these previous systems prevents a web site from building a profile for the user while we decouple the identification step the user undergoes, to access his already existing social profile, with his subsequent requests for content personalization.

Mayer et al. [50] highlight the threats against user privacy by the cross-site tracking capabilities of third-party web services. The authors detail a plethora of tracking technologies used by the embedded pages of advertisement, analytics, and social networking services. Their work demonstrates the high level of sophistication in web tracking technologies, and their resiliency against browser countermeasures.

Roesner et al. [53] study the tracking ecosystem of third-party web services and discuss current defenses, including third-party cookie blocking. They identify cases where tracking services actively try to evade such restrictions by bringing themselves in a first party position, e.g., by spawning pop-up windows. Moreover, the authors present cases in which services are treated as first parties when visited directly and intentionally by the users, and at the same time appear embedded as third parties in web sites, as is the case with social networking services and their social plugins. Overall, they conclude that current restrictions imposed by browsers against third-party tracking are not fool-proof, and at the same time find more than 500 tracking services, some with the capability to capture more than 20% of a user's browsing behavior.

A series of browser add-ons exist [7, 26] that block social plugins from the web pages a user visits by removing them or preventing them from loading, in a manner similar to what Adblock [1] does for advertisements. However, they come at the cost of full loss of functionality as social plugins are completely removed from a page. Note that some of these add-ons are poorly implemented and naively remove the social plugins only after they have appeared on a page, meaning that the corresponding HTTP request containing user-identifying information has already been issued towards the server.

ShareMeNot [28, 53] is a Firefox add-on that strips user cookies from a series of HTTP requests that the web browser issues to load social plugins. As a result, no user-identifying information is sent to the social networking service until the user explicitly interacts with the social plugin. The downside of this approach is that users are deprived of any personalized information offered by the plugin, e.g., the number and names of any of their friends that might have already interacted with a page. In other words, users view these social plugins as if they were logged out from the respective SNS (or browsing in "incognito" mode). Our approach differs from ShareMeNot in that it focuses on providing the full content personalization of existing social plugins while protecting user privacy.

9 Conclusion

Concerns about the interplay between social plugins and privacy are mounting rapidly. Tensions have reached the point that even governments consider to outlaw Facebook's Like button [13]. Recently, in an official response to questions regarding user privacy asked by the government of Norway, it was stated that *"Facebook does not use cookies to track people visiting websites using the Like button"* [37]. The current design of social plugins, as provided by all major social networking services, combined with empirical evidence [9], stresses the need for changes so that words align with actions. We want to believe that SNSs treat the privacy of their members as an issue of the utmost importance, and we hope that they are willing to ensure it through technical means.

In this paper, we have presented a novel design for privacy-preserving social plugins, which provide exactly the same user experience as existing plugins, and at the same time prevent SNSs from being able to track the browsing activities of their users. We have described in detail how this design can be offered transparently as a service to users of existing SNSs without the need to install any additional software, and thus envisage that it could be adopted for the protection of their member's privacy. SafeButton, our proof-of-concept implementation of this design as a browser add-on for Firefox, demonstrates the practicality of our approach. SafeButton is publicly available, and currently supports full content personalization in a privacy-preserving way and with minimal space overhead for seven out of the nine social plugins offered by Facebook, while it loads them 2.8 times faster compared to their original versions.

Availability

SafeButton is publicly available as an open source project at `http://www.cs.columbia.edu/~kontaxis/safebutton/`

Acknowledgements

This work was supported in part by the FP7-PEOPLE-2009-IOF project MALCODE and the FP7 project SysSec, funded by the European Commission under Grant Agreements No. 254116 and No. 257007. This work was also supported by the National Science Foundation through Grant CNS-09-14312, with additional support from Google. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or the NSF.

References

[1] AdBlock Plus. `https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/`.

[2] Browser Security Handbook - Third-party cookie rules. `http://code.google.com/p/browsersec/wiki/Part2#Third-party_cookie_rules`.

[3] Chromium - Don't play plugin instances inside suppressed popups? `http://code.google.com/p/chromium/issues/detail?id=3477`.

[4] Disconnect. `http://disconnect.me/`.

[5] Do Not Track - Universal Web Tracking Opt Out. `http://donottrack.us/`.

[6] Facebook - How many Pages can I like? `https://www.facebook.com/help/?faq=116603848424794`.

[7] Facebook Blocker. `http://webgraph.com/resources/facebookblocker/`.

[8] Facebook Fact Sheet. `http://newsroom.fb.com/content/default.aspx?NewsAreaId=22`.

[9] Facebook fixes logout issue, explains cookies. `http://nikcub.appspot.com/facebook-fixes-logout-issue-explains-cookies`.

[10] Facebook Graph API. `http://developers.facebook.com/docs/reference/api/`.

[11] Facebook Like Button Count Inaccuracies. `http://faso.com/fineartviews/21028/facebook-like-button-count-inaccuracies`.

[12] Facebook Plugins. `http://developers.facebook.com/docs/plugins/`.

[13] Facebook's Like button illegal in German state. `http://news.cnet.com/8301-1023_3-20094866-93/facebooks-like-button-illegal-in-german-state/`.

[14] Firefox Sync. `http://www.mozilla.org/en-US/mobile/sync/`.

[15] Ghostery. `http://www.ghostery.com/`.

[16] Google +1 button. `http://www.google.com/+1/button/`.

[17] HTTP state management. `http://www.ietf.org/rfc/rfc2109.txt`.

[18] Hypertext Transfer Protocol 1.1. `http://www.ietf.org/rfc/rfc2616.txt`.

[19] Indexed Database API. `http://www.w3.org/TR/IndexedDB/`.

[20] MDN - Intercepting Page Loads. `https://developer.mozilla.org/en/XUL_School/Intercepting_Page_Loads`.

[21] MDN - Pageshow Event. `https://developer.mozilla.org/en/using_firefox_1.5_caching#pageshow`.

[22] MDN - window.postMessage. `https://developer.mozilla.org/en/DOM/window.postMessage`.

[23] MDN - XML User Interface Language. `https://developer.mozilla.org/En/XUL`.

[24] Mozilla At a Glance. `http://blog.mozilla.org/press/ataglance/`.

[25] MSDN Blogs - Google Bypassing User Privacy Settings.

http://blogs.msdn.com/b/ie/archive/2012/02/20/google-bypassing-user-privacy-settings.aspx.

[26] No Likie. https://chrome.google.com/webstore/detail/pockodjapmojcccdpgfhkjldcnbhenjm.

[27] NoScript. https://addons.mozilla.org/en-US/firefox/addon/noscript/.

[28] ShareMeNot. http://sharemenot.cs.washington.edu/.

[29] The Chromium projects - Sync. http://www.chromium.org/developers/design-documents/sync.

[30] The Platform for Privacy Preferences Specification. http://www.w3.org/TR/P3P/.

[31] Time Magazine - One Minute on Facebook. http://www.time.com/time/video/player/0,32068,711054024001_2037229,00.html.

[32] Uniform Resource Identifier. http://www.ietf.org/rfc/rfc3986.txt.

[33] Widgets Distribution. http://trends.builtwith.com/widgets.

[34] An Open Letter to Facebook CEO Mark Zuckerberg, June 2010. https://www.eff.org/files/filenode/social_networks/OpenLettertoFacebook.pdf.

[35] Facebook + Media - The Value of a Liker, Sept. 2010. https://www.facebook.com/note.php?note_id=150630338305797.

[36] 5 ways Facebook's new features will fuel social shopping, Sept. 2011. http://mashable.com/2011/09/29/facebook-social-shopping/.

[37] Facebook's response to questions from the Data Inspectorate of Norway, Sept. 2011. http://www.datatilsynet.no/upload/Dokumenter/utredningeravDatatilsynet/FromFacebook-Norway-DPA.pdf.

[38] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320. USENIX Association, 2004.

[39] P. Eckersley. How unique is your web browser? In *Proceedings of the 10th international conference on Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[40] M. Egele, A. Moser, C. Kruegel, and E. Kirda. PoX: Protecting users from malicious facebook applications. In *Proceedings of the 9th Annual IEEE international conference on Pervasive Computing and Communications (PerCom), Workshop Proceedings*, pages 288–294. IEEE Computer Society, 2011.

[41] A. Felt and D. Evans. Privacy protection for social networking platforms. In *Proceedings of the 2008 IEEE Workshop on Web 2.0 Security and Privacy*, 2008.

[42] M. Fredrikson and B. Livshits. RePriv: Re-envisioning in-browser privacy. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pages 131–146. IEEE Computer Society, 2011.

[43] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international World Wide Web Conference (WWW)*, pages 737–744. ACM, 2006.

[44] A. Kobsa. Privacy-enhanced personalization. *Communications of the ACM*, 50:24–33, August 2007.

[45] G. Kontaxis, M. Polychronakis, and E. P. Markatos. SudoWeb: Minimizing information disclosure to third parties in single sign-on platforms. In *Proceedings of the 14th Information Security Conference*, pages 197–212. Springer, 2011.

[46] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *Proceeedings of the 1st Workshop on Online Social Networks*, pages 37–42. ACM, 2008.

[47] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. *SIGCOMM Computer Communication Review*, 40, 2010.

[48] M. M. Lucas and N. Borisov. FlyByNight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the ElecTronic Society (PETS)*, pages 1–8. ACM, 2008.

[49] W. Luo, Q. Xie, and U. Hengartner. FaceCloak: An architecture for user privacy on social networking sites. In *Proceedings of the international conference on computational science and engineering*, pages 26–33. IEEE Computer Society, 2009.

[50] J. R. Mayer and J. C. Mitchell. Third-Party Web Tracking: Policy and Technology. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012.

[51] L. I. Millett, B. Friedman, and E. Felten. Cookies and web browser design: toward realizing informed consent online. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2001.

[52] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *Proceedings of the first workshop on Hot topics in understanding Botnets (HotBots)*. USENIX Association, 2007.

[53] F. Roesner, T. Kohno, and D. 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)*. USENIX Association, 2012.

[54] A. Roosendaal. Facebook tracks and traces everyone: Like this! http://ssrn.com/abstract=1717563.

[55] K. Singh, S. Bhola, and W. Lee. xbook: Redesigning privacy control in social networking platforms. In *Proceedings of the 18th USENIX Security Symposium*, pages 249–266. USENIX Association, 2009.

[56] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10:557–570, October 2002.

[57] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*. IEEE Internet Society, 2010.

[58] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.