PoX: Protecting Users from Malicious Facebook Applications

Manuel Egele[a,*], Andreas Moser[b], Christopher Kruegel[a], Engin Kirda[c]

[a] *University of California, Santa Barbara, Harold Frank Hall, Santa Barbara, CA 93106, USA*
[b] *Vienna University of Technology, Treitlstrasse 1, 1040 Vienna, Austria*
[c] *Northeastern University, 440 Huntington Avenue, 202 West Village H, Boston, MA 02115, USA*

Abstract

Online social networks such as Facebook, MySpace, and Orkut store large amounts of sensitive user data. While a user can legitimately assume that a social network provider adheres to strict privacy standards, we argue that it is unwise to trust third-party applications on these platforms in the same way.

Although the social network provider would be in the best position to implement fine-grained access control for third party applications directly into the platform, such mechanisms are still missing. Furthermore, recent press releases do not indicate that such mechanisms will be put in place in the near future. Therefore, we introduce PoX, an extension for Facebook that makes requests for private data explicit to the user and allows her to exert fine-grained access control over what profile data can be accessed by individual applications. By leveraging a client-side proxy that executes in the user's web browser, data requests can be relayed to Facebook without forcing the user to trust additional third parties. Of course, the presented system is backwards compatible and transparently falls back to the original behavior if a client does not support our system. Thus, we consider PoX to be a readily available alternative for privacy-aware users that do not want to wait for improvements implemented by Facebook itself.

1. Introduction

Social networks have recently enjoyed a tremendous success. Statistics for Facebook, arguably the most popular social network, indicate that its user base now exceeds 400 million users [3]. The amount and detail of private data stored in user profiles on these networks makes an attractive target for marketing companies, spammers, spear phishers, and identity thieves. The operators of social networking sites are very well-aware of the privacy implications of such a collection of personal data. Therefore, they provide a multitude of settings that allow users to control what parties have access to their profile data, and the content they create on the social network. Facebook, for example, provides facilities to arrange contacts, so-called friends, in multiple friend lists. Each piece of information shared on the network can then be assigned a list where only friends on that list have access to that information.

*Corresponding author
Email addresses: maeg@cs.ucsb.edu (Manuel Egele), andy@iseclab.org (Andreas Moser), chris@cs.ucsb.edu (Christopher Kruegel), ek@ccs.neu.edu (Engin Kirda)

Third-party applications. Many social networks also offer the possibility to create additional applications that extend the functionality of the network. The two major platforms for such applications are the Facebook Platform and Open Social [4]. While applications designed for the Facebook Platform can only be executed in Facebook, Open Social is a combined effort to allow developers to run their applications on any social network that supports the Open Social platform (e.g., MySpace and Orkut). The popularity of third-party social networking applications can be appreciated by looking at the ever-increasing number of active Facebook applications that are available since the Facebook Platform was launched in 2007 [5]. In fact, recent Facebook statistics [3] indicate that, at the time of writing, more than 550,000 third-party applications are available to Facebook's users.

To seamlessly embed an application into the social network, the platforms provide libraries to third-party developers. Those libraries contain the bindings for different programming languages to easily access the functionality and data of the social network. If an application, for example, needs to access the birthday of a user, the appropriate call to the library will return this value. All communication that happens between the application and the social network's servers is encapsulated by this library. Therefore, if the underlying protocol is modified or extended, application developers only need to replace the existing library with the newer version to take full advantage of any improvements.

With the tight integration between the social network and third-party applications, privacy issues arise, especially when it comes to the handling of sensitive profile data. Once an application obtains access to profile data, it is impossible for the social network to further enforce or asses how this data is used by the application. Lacking technical means to enforce profile data privacy, Facebook requires every application developer to agree to their terms of service (TOS). These terms state that an application must not store gathered profile data nor propagate that data further. However, reported incidents [14, 17] where applications violated these terms of service call for stronger means to protect the users' profile data from rogue Facebook applications. For example, in an incident involving the "Top Friends" application [17], everybody could access the birthday, relationship status, and gender of all Top Friends users, even in cases where this information was set to be private by those users. Once the issue was discovered, Facebook suspended this application from their platform. Clearly, such incidents, which result in wide media coverage, make users aware of the need for improved access control for third-party applications. This prompted Facebook to make changes with regard to their privacy settings, However, recent research [7] and news [12, 18] suggest that these changes are not enough. Furthermore, Facebook has a history of changing default privacy settings to the disadvantage of its users' privacy. For example, Facebook outraged civil liberty campaigners by introducing privacy settings that, while seemingly improving the users' privacy, dramatically increase the amount of personal information users share publicly by default [2]. Furthermore, in an interview with Facebook's CEO Mark Zuckerberg, he stated that privacy is no longer a "social norm" [6]. This gives additional reason to expect very little progress in that area.

To limit and control the access of third-party applications to user profile data, we propose a fine-grained access control scheme for Facebook applications. That is, we suggest that all requests for profile data are made explicit to the user. This provides the user with precise control over which information can be accessed by what application. For example, there is no need for popular, fun quiz-style applications to access any personal information. Of course, the basic idea of fine-grained access control is not novel. However, any system that wishes to introduce fine-grained access control for Facebook applications has to take into account the fact that there are hundreds of thousands of applications already out there. Moreover, we are aware that the best position to implement changes to the access control mechanism is in the Facebook Platform directly. However, Facebook did not implement such features in the past and is unlikely to immediately deploy a solution proposed in an academic research paper. Thus, the key requirement of a practical solution is *deployability* without the help of Facebook. With this, we mean that a solution should not require support from the social network, should keep modifications of existing applications to a minimum, and support a mixed mode in which the system simultaneously handles clients that already implement improved privacy measures along with legacy clients.

One way to work around the restrictions imposed by deployability would be a parallel system for third-party applications that operates independently of Facebook. We think that such solutions are not desirable, because they introduce an additional party that has to be trusted by many users to a significant extent. As a result, a second requirement is that a solution *does not introduce an additional, central party that needs to be trusted like Facebook*. The challenge, now, is to design a practical solution that meets the two requirements stated above.

In this paper, we advocate a solution to the access control problem that is in accordance with the above stated requirements, using client-side proxies. In our solution, a proxy executes entirely in the user's browser, and accepts profile data requests from Facebook Platform applications. The proxy scrutinizes each request and enforces access control decisions by verifying that the application sending the request is allowed to access the desired information. Requests that pass this check are forwarded to the Facebook servers by the client-side proxy, and the results are passed back to the application. This approach has the advantage that all relevant code is executed at the client side, where it can be trivially reviewed by the interested user. Therefore, no additional trusted entity is introduced to the system.

In this work, we present PoX, a **P**roxy **O**n the **C**lient-**S**ide system that provides a Facebook user with fine-grained access control capabilities over which parts of her private profile information can be accessed by third-party applications. This paper makes the following contributions:

- We summarize the current, non-satisfactory privacy situation regarding third-party Facebook applications.
- To remedy the shortcomings of the existing system, we propose PoX, a system that allows users to exert fine-grained access control over Facebook applications.

- We illustrate the design and prototype implementation of PoX.
- Finally, we present the results of our extensive evaluation of the prototype, showing that deploying PoX is simple and light-weight.

2. Background on Facebook applications

This section aims to make the reader familiar with the concept of Facebook applications, and the nuts and bolts of tying them into the Facebook Platform.

Register a Facebook application. Assuming that a developer already has an account on Facebook, the first step in the process of developing a Facebook application consists of registering this application with the Facebook Platform. This registration actually happens via a Facebook application (called the *developers* application), which the developer has to add to her profile. Each registered application is assigned an application-id and a private application key. All communication between the third party application server and the Facebook servers needs to be signed with the private application key.

Installing a Facebook application. For a Facebook user to install an application, all that is required is to visit the application's landing page. The first time the landing page is visited, Facebook displays a dialog, requesting the user's authorization that the application can access her profile data. In this step, Facebook enforces only a very coarse grained access control policy. An application can either request all private profile information from the user at once or use only information that is publicly available anyways. Thus, if the application needs only one piece of the user's private data, the user has to agree to grant access to all her profile information, or she cannot use the application at all.

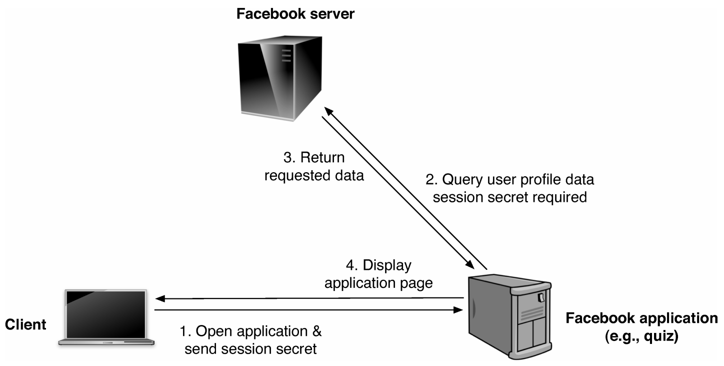

Facebook server
3. Return requested data
2. Query user profile data session secret required
4. Display application page
Client
1. Open application & send session secret
Facebook application (e.g., quiz)

Figure 1: Data-flow in Facebook

Accessing profile data. Accessing the user's profile data is a cornerstone of many Facebook applications. Figure 1 illustrates the flow of information when an application is requesting user profile data from the Facebook servers. Note that unless the user authorized an application during installation, the application can only retrieve those fields of a user's profile that are marked as public. However, an authorized application can, by default, also access all private information of the user, and even the profiles of the user's friends.

For an application to request profile information from the Facebook servers, it is necessary to transmit a valid session secret (Step 1 in Figure 1). This secret is created when the user visits the application landing page and transmitted to the application host as a URL parameter. The access library, in turn, automatically appends the session secret to all profile data requests (Step 2). Once the Facebook servers receive such a request, the session secret allows Facebook to determine whether the application is authorized to receive the requested information. More precisely, the Facebook servers will reject any data requests with a "session secret invalid or no longer valid" error message, unless the application transmits a valid session secret in the request. For requests that contain a valid session secret, the Facebook server responds with the requested profile information (Step 3). The application then continues processing this data, and answers the client request with the corresponding HTML output (Step 4).

Access control for Facebook applications. While Facebook provides rather elaborate privacy controls to govern what content is shared with a users' friends, the current privacy controls regarding third-party Facebook applications are limited at best. Even if the press releases after the recent changes for privacy settings for Facebook applications made it seem like the applications are now granted less access rights than before the changes, the situation essentially remains unchanged. Unless the application requires solely public profile data, the user has to either grant an application full access rights to all her profile information, or she cannot use the application at all. These are the only options available to the user, and they do not vary with the data needs of an application. This behavior is in direct contradiction with the "principle of least privilege“ [19], which states that every actor should only get the minimal set of access privileges that allow it to fulfill its task. The privacy problem in the current system exists because there are no means for a user to restrict or learn what information is accessed by a given application. An application that is in possession of a valid session secret may access any information in the user's and her friends' profiles without any notification of the user.

3. PoX design

As mentioned previously, current Facebook applications communicate directly with the Facebook servers, and have unrestricted access to their users' data. Limiting and controlling this access would require a reference monitor (e.g., a proxy) that sits between the Facebook application and the server. This reference monitor could then control and restrict the data that is requested and exchanged. For example, one could envision a trusted party that creates a proxy between third-party applications and the Facebook servers. Existing applications would need to be modified to send requests for profile data to the proxy instead of the Facebook servers. The proxy server holds a mapping between applications and profile data items that indicates which application is allowed to access what information. Following this approach, the individual applications do not get access to the profile data directly. All requests have to be funnelled through the proxy. However, the user would have to trust the proxy, as it needs unrestricted access to the profile data, and is responsible for enforcing access control. Even if the user had access to the source code of the proxy application, she cannot make sure that the published source is the program that is actually executed. Furthermore, the proxy would see all data requests by third-party applications, thus, being able to aggregate and create detailed behavioral profiles of all its users.

Introducing client-side proxies. To remove the need for a central, trusted party, PoX executes the proxy on the client side. That is, each user is basically running her own proxy locally. Thus, PoX still allows the user to exert fine-grained access control, but does not require additional trust relationships. Under the premise that we cannot change the behavior of the Facebook servers, PoX has to prevent the session secret from being transmitted to the third-party application. The reason for this is that a valid session secret would allow the application to retrieve profile data directly from the Facebook servers, without the user's knowledge. Therefore, a client-side component (i.e., browser plug-in) removes the session secret from all outgoing requests. This prevents the application from communicating directly with the Facebook server to request user profile data. Thus, whenever the application needs information about the user from Facebook, it sends a data request back to the proxy running in the client's browser. Once the proxy receives such a request, it performs the access control checks in accordance with the user-chosen settings. If the request passes the checks, the proxy signs the request with its own secret, forwards the request to the Facebook server, and relays the results back to the calling application. In this way, the application only has access to the data to which the user explicitly granted access. Once the application receives the data, it proceeds in creating the output (i.e., the HTML source describing the application page) as usual. The modified flow of data using the PoX system is depicted in Figure 2.

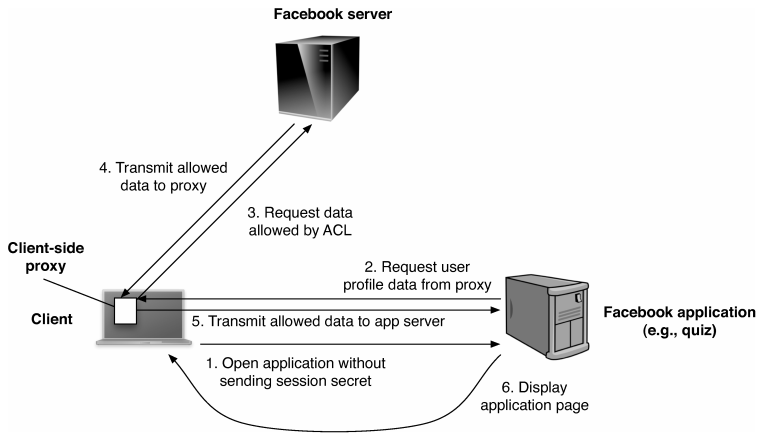

Facebook server
4. Transmit allowed data to proxy
3. Request data allowed by ACL
Client-side proxy
2. Request user profile data from proxy
Client
5. Transmit allowed data to app server
Facebook application (e.g., quiz)
1. Open application without sending session secret
6. Display application page

Figure 2: Modified data-flow with PoX

The remainder of this section elaborates on some of the implementation decisions we made during the development of our PoX prototype. Sections 3.1 and 3.2 show how the proxy is implemented on the client side. The communication method we use to pass data between the application server and the client-side proxy is described in Section 3.3. Finally, we illustrate how the PoX server library differs from the original Facebook library in Section 3.4.

3.1. Plug-ins

One of PoX' objectives is to make sure that the session secret is not transmitted to the third-party application. This is necessary to ensure that the application does not retrieve profile data from the Facebook servers directly. Instead, the application is forced to make all profile data requests explicit to the user by relaying the request via the client-side proxy. To prevent the session secret from being transmitted to the third-party application, we developed plug-ins for the Internet Explorer and Firefox browsers that filter the session secret tokens from the HTTP stream.

Note that using an application that is not PoX aware while the PoX browser plug-in is active, might lead to unexpected behavior. The application will connect to the Facebook server with an invalid session secret and, thus, it will not receive any profile data. While this behavior is a safe fall-back from a privacy point of view, a user might decide to fully trust such a legacy application to not violate her privacy. Therefore, the PoX plug-ins can be temporarily disabled from the browser's user interface.

To make a client PoX aware, it is sufficient for the user to install the PoX plug-in. However, to ensure that no data can be leaked to third-party Facebook applications, the Facebook privacy settings of the user have to be changed such that applications used by friends of the user cannot retrieve sensitive data. This can be easily done with the existing settings.

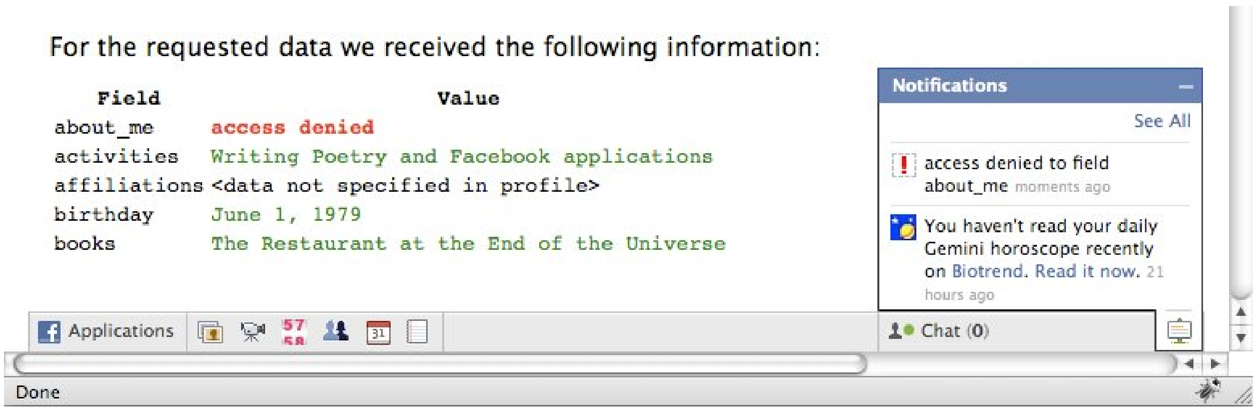
3.2. Client-side proxy

In PoX, the client-side proxy is automatically loaded if a PoX-aware Facebook application is used. However, the application itself is displayed to the end user exactly as without the PoX system as the proxy code resides in a hidden IFRAME and is not visible in the browser window.

Once the proxy starts execution, it retrieves the current access control list (ACL) for the user. In our current implementation, this list is stored by a dedicated Facebook application. This list indicates what application should be allowed to access what pieces of profile data. If an application-data mapping is not present, it is conservatively assumed that access is forbidden for the application in question. The ACL is fetched only once during the lifetime of the proxy. Apart from performance considerations (i.e., caching the ACL reduces load time), privacy concerns also played a role in this design decision. If the proxy would retrieve the ACL for the currently active application individually, this would allow the storage site of the ACL to build application usage profiles for its users. Storing the ACL locally would remedy the privacy concerns and increase performance. That is, the ACLs would not need to be retrieved from the Internet. However, a local ACL leaves the user with the task to keep the ACL accessible when she uses a different computer (e.g., from an Internet cafe) to access her Facebook profile.

Once loaded, the proxy waits for requests from Facebook applications that require access to user data. The Facebook libraries can create such requests in two different ways. The most common way for application developers to request user data is to call a library function that provides access to the requested data fields. In this case, a comma-separated list of the requested fields is generated by the library and sent to the Facebook servers or a PoX-enabled client. This list can easily be parsed by the PoX system, and thus, access control can be enforced. Alternatively, the developer can formulate his request as a Facebook Query Language (FQL) statement. FQL is a query language that syntactically resembles SQL, but contains additional restrictions such that queries cannot exhaust too many resources on the Facebook servers. Furthermore, all valid FQL queries need to explicitly list the data fields they want to access. Thus, PoX can enforce access control on FQL queries by performing simple string pattern matching.

PoX conservatively assumes that data fields that are requested, but do not have a corresponding "accept entry" in the ACL are not accessible to the application. Additionally, to make sure that the user knows that an application has requested data that was prohibited by the ACL, we also send a Facebook notification to the user indicating which fields have been blocked by the PoX system. A screen shot illustrating the notification can be seen in Figure 3. Once the request is sanitized, the proxy forwards the modified request to the Facebook server, and the result is relayed back to the application.


For the requested data we received the following information:

Field	Value
about_me	access denied
activities	Writing Poetry and Facebook applications
affiliations	<data not specified in profile>
birthday	June 1, 1979
books	The Restaurant at the End of the Universe

Notifications

See All

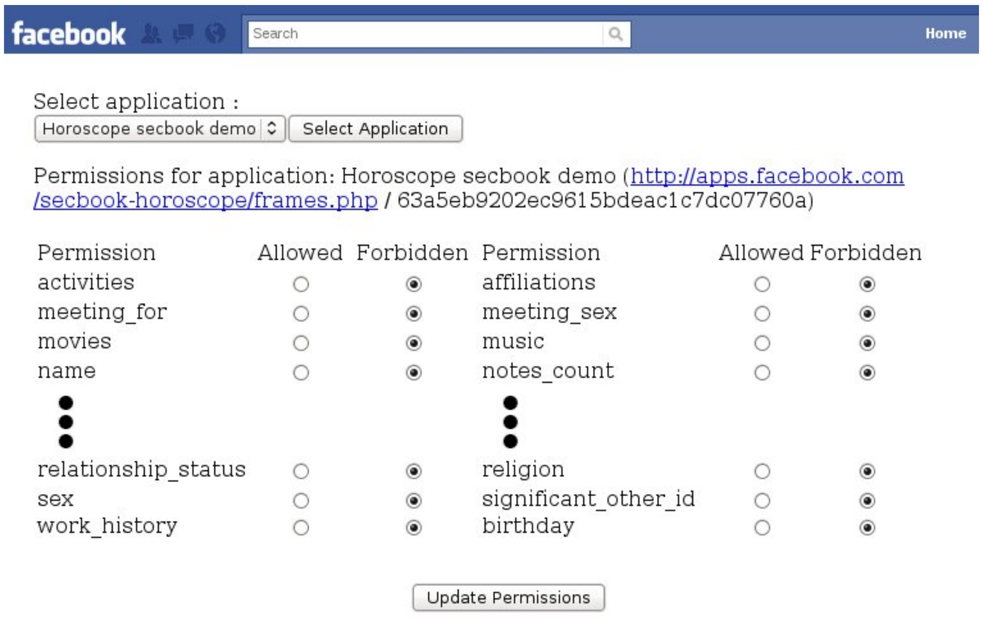
access denied to field about_me moments ago

You haven't read your daily Gemini horoscope recently on Biotrend. Read it now. 21 hours ago

Figure 3: Notification of denied access

The proxy Facebook application. Our current prototype implementation of the client-side proxy is realized as a Facebook Platform application. This application does not only host the JavaScript code for the client-side proxy, but it also provides an application ID and private key to the proxy that is needed in order to communicate with the Facebook server. Furthermore, this application provides the means to store and manipulate a user's access control list. To specify access control for an application, the user has to select the application for which she wants to create or modify the ACL. In a subsequent step, the user can decide for each of the data items stored in her profile whether or not the application is allowed to access this information (see Figure 4). This user interface is similar to the existing Facebook privacy settings which specify what information an application installed by a friend can access.

Note that the user is not required to trust this proxy application more than any other application. More precisely, even though the proxy is allowed to request data using the users' session secret *inside* the client browser, this application, just as any other, does not receive the session secret from its users. Thus, it cannot communicate with the Facebook servers directly.



Select application :

Horoscope secbook demo | Select Application

Permissions for application: Horoscope secbook demo (http://apps.facebook.com/secbook-horoscope/frames.php / 63a5eb9202ec9615bdeac1c7dc07760a)

Permission	Allowed	Forbidden	Permission	Allowed	Forbidden
activities	○	◉	affiliations	○	◉
meeting_for	○	◉	meeting_sex	○	◉
movies	○	◉	music	○	◉
name	○	◉	notes_count	○	◉
⋮			⋮		
relationship_status	○	◉	religion	○	◉
sex	○	◉	significant_other_id	○	◉
work_history	○	◉	birthday	○	◉

Update Permissions

Figure 4: Managing ACLs with PoX

3.3. Server to client communication

Using current standard web technologies, it is a rather tedious task to implement a communication channel that enables a web server to initiate a connection and push data to a client. However, to be able to run the proxy server entirely in the web browser of the Facebook user, it is necessary that the application server can initiate requests to the client (the proxy) whenever it needs to access profile data (shown as Step 2 in Figure 2). While such communication channels can be implemented using, for example, the Flash technology, we wanted our proxy to be as simple and independent of proprietary protocols as possible. Therefore, we use an approach known as "long polling" for the notification of the client proxy. In this approach, the client sends a standard HTTP request to the web server and tries to fetch a certain dynamic web page generated by a PHP script. As long as there is no request to process for the client, the web server stalls execution of this script, causing the client to wait for a response. As soon as data is available, the script is resumed, and the request is transferred to the client where it is subsequently processed.

The advantage of the "long polling" approach, besides the fact that it uses no additional software and, thus, runs in every web browser, is that only standard HTTP connections are used for the data transfer. This means that even when both client browser and the server hosting the Facebook application are firewalled (except for the HTTP port on the server), the application server can still talk to the client proxy. This further ensures that existing applications will not break if the developer switches to using the PoX library.

3.4. Server-side PoX library

To make an existing Facebook application use the PoX system, an application developer only needs to replace the original Facebook server-side library with the PoX server-side library. This modified version of the Facebook library performs an automated check for PoX-aware clients. If the connecting client is PoX-aware, the server automatically funnels all profile data requests through the client-side proxy. For non-PoX-aware clients, the library transparently falls back to the current behavior and sends profile data requests to the Facebook servers directly. This mechanism simply calls existing code from the Facebook library, thus, ensuring backwards compatibility. Furthermore, this scheme allows the simultaneous support for PoX-aware and traditional clients, and facilitates incremental deployment.

A motivation for developers of applications with profile data requirements to support PoX arises when users with PoX-enabled clients use their applications. A PoX-aware client does not transmit the session credentials that are required by an application to request profile data from the Facebook servers. Therefore, an application not supporting PoX cannot access any non-public profile information of users who run the PoX plug-in.

4. Evaluation

In this section, we show that the PoX system can deliver data from the Facebook database to the application server fast enough to be considered a practical, privacy-improving solution for real world Facebook applications. We will show that even if the initial request of an application is usually slower than a request sent by the original Facebook API, the system proposed in this paper can actually outperform the method currently used by Facebook for subsequent data requests.

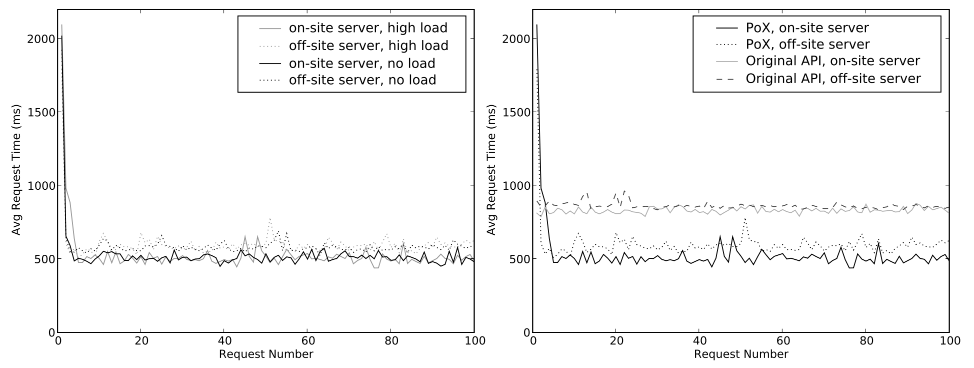
4.1. Performance of PoX

To show that the PoX system is capable of serving data fast enough to be considered also for large Facebook applications, we first analyzed how the PoX library performs under heavy load.

For this experiment, we set up five virtual machines running Ubuntu Linux 9.04. In each one of those virtual machines, we simulated two Facebook clients, for a total of ten users. To simulate clients, we implemented a Firefox extension that is able to automatically request pages from a Facebook application, including the steps to authenticate the application for the user and log the user into Facebook. For the server side of the experiments presented in this section, we set up two identical Facebook applications hosted on commodity desktop machines. One is equipped with an Intel Core 2 processor running at 2.4 GHz and 4GB of RAM, and it is located on the same local network as the client virtual machines (on-site setup). We used this machine to test the performance of the PoX system with very low network latency. The second machine has an Intel Pentium 4 CPU clocked at 3 GHz and has 2GB of memory installed. This machine is located remotely and the round trip time between the clients and the off-site machine is 24 ms on average (off-site setup).

During the experiment, we had each of the ten clients download 100 times the page that requests user information from the Facebook server. This was done twice, once for the on-site and once for the off-site application server. For each request, we measured how long it took the application to obtain user profile information by issuing an Facebook API call. As all simulated clients had the PoX plug-in installed, those requests were sent via the client proxy to Facebook and, thus, the time measured includes the proxy processing time, the time to transfer data between the proxy and the Facebook application, and the time to obtain the data from the Facebook server. The results for the two runs are shown in Figure 5a as the graphs marked with "no load."

In the next step, we repeated the experiment, but this time, the goal is to show that our proxy server approach can cope with heavy load. To this end, we started 50 additional clients in five virtual machines hosted on another server. These clients repeatedly requested the same page as fast as possible. Combined, this simulates a load of 60 concurrent users. Conservatively assuming that each simulated user issues one request per second, this load would add up to more than 160 million requests per month. Thus, even if we take into account effects such as peak application usage times, we believe that a library that is able to serve 160 million requests per month is suited to be used for large Facebook applications. Furthermore, results will show that the PoX library can handle the load of 60 concurrent users with ease.



(a) Request times for high / low load (b) Request times for PoX and the original FB API

Figure 5: PoX Performance.

Figure 5a shows that, for both the on-site and the off-site application server, handling 60 concurrent clients does not increase the request time by much (graphs marked with "high load"). The overall average request time only increased from 526 ms to 531 ms for the on-site server, and from 593 ms to 601 ms for the off-site server. This clearly shows that the PoX system is suited for usage in real-world Facebook applications.

Note that it is immediately evident that the initial request using PoX takes significantly longer to process. This is due to the fact that it takes some time for the client browser to load the proxy. Furthermore, the client side proxy includes two additional scripts that have to be downloaded: The Facebook JavaScript client API to query the Facebook servers for profile information, and a JSON processing script. Profile data requests that are sent by a Facebook application are encoded as JSON objects. Thus, to perform access control, PoX needs to parse these objects. Downloading those scripts and initializing the Facebook session takes around

500 ms on average. The access control lists have to be downloaded only for the first request, which takes another 100 ms in our setup. Additionally, the first request to the application server takes a bit more time because of the necessary connection setup. During this proxy load time, the Facebook application is often already waiting for the requested data, which results in an overhead of up to one second. However, this overhead occurs only once and remains well within reasonable limits.

4.2. Comparing PoX to the original Facebook library

Figure 5b depicts the time needed to get information from the Facebook servers using the original Facebook library and compares it to the time required by the PoX library (as measured in the experiment in Section 4.1). In this experiment, we started a total of ten Facebook clients in five virtual machines. Those clients download 100 times a page from our Facebook application. Again, we measured the time it took the application to acquire the requested data. For this experiment, the PoX plug-in of the clients were disabled. Therefore, we measured the required time to directly connect to the Facebook server using the original Facebook library. To make the experiment more realistic, we again started another 50 clients in virtual machines hosted on another server that repeatedly requested the same page as fast as possible.

We then compared the data from this run to the data we gathered in the experiment conducted in Section 4.1. Note that the experiment setup was exactly the same as described in the previous section except for the disabled plug-in. Thus, the times are comparable. The graph in Figure 5b shows the request time for each of the 100 requests, averaged over all clients. For reasons described in the previous section, the initial requests are slower with the PoX system than with the original library. After the initial request, however, we see that the PoX library actually outperforms the original Facebook library. This speedup can be accounted to the fact that the client proxy just has to establish one persistent HTTP connection that is reused for all subsequent requests. For the Facebook library this is not possible, because the PHP script is terminated at the end of every request. Thus, even though multiple requests for Facebook data during the processing of a *single* application page can be submitted (pooled) over the same connection, for each new page, a new connection to the Facebook server has to be established. This adds another round trip time to the Facebook server to the overall request time. Therefore, using the method presented in this paper does not only improve the privacy of a user, but can also improve the performance of Facebook applications.

The results are very comparable for the two application servers we used. For the direct connection to Facebook, the overall average request time for the two servers only differs by 32.22 ms. For the run using the client proxy, the average request time is 70.02 ms higher for the off-site server. This shows that the proxy server has a constant low run time and only the round trip time is added twice for the off-site server (once for getting the request to the client and once for sending the results back to the application).

4.3. Performance measurements with mixed clients

To show that clients that have the PoX plug-in installed as well as normal, unmodified clients can use an application with the PoX library at the same time, we again use our simulated Facebook clients to request data from our test application. For this experiment, the application was hosted on the on-site application server. We set up ten virtual machines with two Facebook users each, for a total of 20 clients. During one run of the experiment, we had each client request the website displaying profile data 100 times. For each request, we measured the time as described in the previous sections. We started five different runs of this experiment. The first time, all clients used the original Facebook library. For each successive run, we activated the PoX browser plug-in in another 25% of the total clients. In the last run, all clients were using the client proxy system. We grouped the requests by their sequential numbers over all (PoX and original API) clients. The average request time for each request number is plotted in Figure 6.

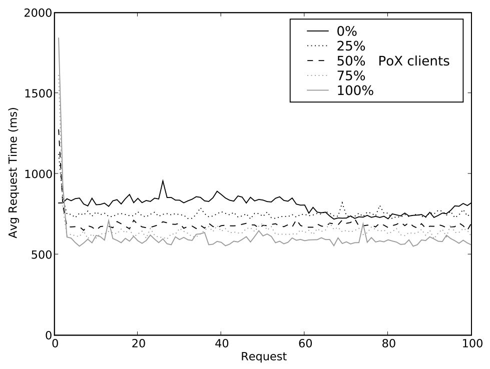


Figure 6: Data request time comparison for mixed PoX and original API clients.

The figure shows that the average request times increase linearly as more unmodified clients are added (except for the first request). This indicates that PoX clients and clients using the original Facebook library do not influence each other in any way. Thus, it is no problem to migrate existing Facebook applications to the PoX library even if most of the clients using the application are still unaware of the PoX system.

5. Limitations

In this section we elaborate on some limitations of the current PoX prototype implementation. Where appropriate, we also mention how these limitations can be removed.

Currently, whenever Facebook detects a rogue application, the social network suspends this application from its platform, preventing all users from accessing or installing this application. Although this blacklist approach protects future users from rogue applications, existing users need to assume (due to missing access controls) that the application in question has already harvested their profile information. In PoX, since all requests from proxied applications seem to originate from the proxy application itself, this blacklist approach would be ineffective. To protect PoX users from potential risks of this behavior, PoX could trivially be extended to support Facebook application blacklists. To this end, before PoX processes a data request for a third-party application, it verifies that the application can still be installed. For example, visiting the application's landing page ensures that the application has not been blacklisted, since a blocked application results in an error page being displayed.

Another small limitation of the PoX system is that every request to the Facebook REST (Representational state transfer) server appears to be coming from the PoX application instead of the actual Facebook application. This is because every request to the Facebook servers has to be signed by the application which is requesting the profile data. To create this signature, the session key and the secret application key are required. However, the application does not have the session key (because PoX prevents the key from being sent to the application), and the proxy does not have the application's secret key. As a result, this signature cannot be generated. Note that obtaining the application secret key from the application to act as an transparent proxy is not an option as this key is used to identify the application owner. Therefore, handing over the key would be equal to giving up control over the user base of the application, which no developer would consider. Thus, the client proxy extracts every request and creates a new signature, using the credentials of the proxy application.

While this is no problem for data requests (which make up the vast majority of Facebook API calls), a few procedures deliver different results when they are executed through the client proxy system. For example, whenever a proxied application attempts to post news on the user's dashboard (a function provided by Facebook to show all the application activities of a user to his friends), the post appears to be coming from the PoX application instead. That is, the name and icon next to the news post are of the PoX application, instead of the name and icon of the application actually posting the news. However, the proxied application still controls the message text that appears on the users' wall. We believe that this is a minor limitation that can be tolerated considering the privacy gain the PoX system provides.

Furthermore, in its current prototype implementation PoX does not authenticate third party applications reliably. That is, PoX performs all data requests that originate from applications (identified by their Facebook application ID) for which the user has a corresponding ACL entry. As application IDs are public, this opens the door to impersonation attacks. However, to thwart such attacks the PoX access library can be extended to perform reliable authentication of third party applications.

6. Related work

Users and their profile information stored on social networks are at risk. For example, Bilge et al. [9] performed identity fraud experiments in social networks. To this end, they initiated friendship requests to a set of victims. Once accepted, they cloned their victims' profiles in other social networks. By contacting a victims' friends in the new network, they were able to impersonate the victim in the new social network. As authorized applications already have access to the profile data of their users, developers of malicious applications could easily use the gathered information for similar attacks.

The study performed by Jagatic et al. [13] suggests that phishing campaigns that leverage data accessible from social networks have a four times higher probability to lure victims to disclose private information than common phishing campaigns. One has to assume that campaigns leveraging otherwise private profile data have an even higher success rate. Currently, one approach to access such data are rogue social network applications.

Privacy concerns with regard to online social networks applications attracted the attention of the research community. In [11], Felt et al. evaluated the requirements of personal data for 150 popular Facebook applications. They conclude that only 9% of the evaluated applications need to access personal profile data to work correctly. The remaining 91% would work without restriction if only public profile data was accessible.

The application framework introduced in [21] is specifically designed to keep all personal profile data confined. To this end, the xBook framework provides a restricted JavaScript environment based on ADSafe [1], extended with data storage capabilities. The authors envision that the user completely trusts their platform and require that all third-party applications are executed inside so-called xBook components. xBook enforces that applications can only transfer data to external entities that the user has explicitly agreed to. xBook solely supports JavaScript on both the client and server-side. That is, existing third-party applications written in other languages than server-side JavaScript would have to be ported to support xBook. For an application to support PoX, however, it is sufficient to substitute the existing client library with a PoX aware version. No further changes to the application code itself are necessary. Moreover, xBook introduces a trusted hosting platform and requires that application developers release their source codes over to this platform. This violates our second requirement which states that no additional trusted parties should be introduced to the existing system.

Shehab et al. [20] also address the problem of non-restricted access to profile data for third-party applications on social networks. Their approach to introduce a fine-grained access control system consists of three steps. First, upon registration, each application has to submit a so-called *application sheet*. This document describes the API calls that the application will make along with the data that is required to perform its task successfully. The second step results in a so-called *user sheet*, reflecting the access control decisions the user made for each element of the application sheet. Finally, the third step covers the necessary modifications the application has to undergo to cope with data that it cannot read because access is denied by the user sheet. The deployment of this approach would require substantial support from any social network that decides to implement it. In particular, the filtering of requests with the user sheet would need to be implemented at the social network. Furthermore, application developers would need to produce application sheets for existing applications. In our system, application developers do not need to modify their applications, but only need to replace the Facebook library with our Pox-aware version.

Lucas et al. [15] propose flyByNight, a cryptographic system that encrypts all communication between users on the Facebook Platform. The authors argue that, in order to minimize privacy breaches, all messages relayed through Facebook should be encrypted in a way that no message reaches the Facebook servers in the clear. Therefore, they implemented a proof of concept Facebook application that relies on asymmetric key cryptographic methods to encrypt messages with their respective receiver's public keys. To efficiently handle the one-to-many communication (which happens, for example, as a consequence of status updates), they leverage proxy cryptography. The purpose of flyByNight is to make communication in the social network unaccessible to the social network operator. The system does not, however, protect the data stored in a user's profile. Indeed, it is not clear whether the flyByNight approach could be adapted to support encrypted profile data and third-party applications simultaneously. In contrast, PoX assumes that the social network operator behaves in accordance with high privacy standards, and additional privacy protection is required only for third-party applications.

Another method to protect the data of Facebook users was presented by Lou et al. [16]. In their approach, they store fake information on the Facebook site, but keep the real data encrypted on a separate server. In this way, only trusted users who possess the appropriate decryption keys can view the stored information. This is done by installing a browser extension that looks up and decrypts the matching data set on the fly. This approach could be applied to very simple third-party applications that only retrieve data to display it unmodified on the application web site. However, even very simple applications that show different output depending on the data (for example, a horoscope application) would fail.

In their work, Athanasopoulos et al. [8] illustrate how third-party applications in social networks can be abused to perform distributed denial of service attacks. To this end, they implemented a malicious Facebook application that triggered requests for additional resources from the target of the DoS attack. Having only 480 daily users for their application resulted in up to 6 MBit of additional traffic to their target host.

Cutillo et al. [10] suggest that the central storage of all user profile data at the social network provider is a privacy risk that should be avoided. They propose to create a social network on top of a peer-to-peer network to allow sensitive user data to be stored in a decentralized manner. Nevertheless, in their proposed system, most core functionality of existing social networks, such as profile publication or private messaging, is present as well.

7. Conclusions

The amount of personal and sensitive data stored on social networks attracts the attention of people with questionable intentions. This information allows an attacker, for example, to create highly customized spear phishing emails. Furthermore, identity thieves can leverage the additional knowledge they can retrieve from such online sources.

Unfortunately, using an application on Facebook is only possible if the user gives up on privacy and grants that application full access to her and her friends' profile information. That is, once authorized, even the simplest application can query a user's profile in its entirety, without the user noticing. To remedy this privacy problem, this paper introduced PoX. By forcing applications to make profile data requests explicit to the user and funnel such requests through client-side proxies, PoX can exert fine-grained access control on profile data before it is transmitted to the application. PoX is fully backwards-compatible and simultaneously supports a mix of PoX-aware and traditional clients. Moreover, deploying PoX for existing applications is trivially accomplished by substituting the Facebook access library. By installing the PoX plug-in in their browser, users can protect their profile data from malicious applications, and can take full advantage of PoX compliant third-party Facebook applications. Thus, our system can be deployed today. In addition, the system uses distributed proxies that do not require a user to trust any other third-party. Our evaluation of PoX demonstrates that it is possible and feasible to have fine-grained access control over profile data for Facebook applications.

References

[1] Adsafe - making javascript safe for advertising. http://www.adsafe.org/.

[2] Facebook privacy change angers campaigners. http://www.guardian.co.uk/technology/2009/dec/10/facebook-privacy.

[3] Facebook statistics. http://www.facebook.com/press/info.php?statistics.

[4] Opensocial - the web is better when it's social. http://code.google.com/apis/opensocial/.

[5] Facebook platform launches with 65 developer partners and over 85 applications for facebook. http://www.facebook.com/press/releases.php?p=1319, 2007.

[6] Facebook's Zuckerberg: Privacy no longer a "social norm". http://www.pamorama.net/2010/01/11/facebooks-zuckerberg-privacy-no-longer-a-social-norm/, 2010.

[7] A. Acquisti, R. Gross, and F. Stutzman. Faces of Facebook: Privacy in the age of augmented reality. BlackHat USA, 2011.

[8] E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. G. Anagnostakis, and E. P. Markatos. Antisocial networks: Turning a social network into a botnet. In *ISC*, pages 146–160, 2008.

[9] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 551–560, New York, NY, USA, 2009. ACM.

[10] L. A. Cutillo, R. Molva, and T. Strufe. Privacy preserving social networking through decentralization. In *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, pages 145–152, 2009.

[11] A. Felt and D. Evans. Privacy protection for social networking platforms. In *Web 2.0 Security and Privacy, (W2SP 2008)*, 2008.

[12] K. Hill. Mark Zuckerberg's Private Photos Exposed Thanks To Facebook Flaw. http://www.forbes.com/sites/kashmirhill/2011/12/06/mark-zuckerbergs-private-photos-exposed-thanks-to-facebook-flaw/, 2011.

[13] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, 2007.

[14] S. Kelly. Identity 'at risk' on facebook. http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm, 2008.

[15] M. M. Lucas and N. Borisov. Flybynight: mitigating the privacy risks of social networking. In *WPES '08: Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 1–8, New York, NY, USA, 2008. ACM.

[16] W. Luo, Q. Xie, and U. Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Proceedings of 2009 IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT-09)*, August 2009.

[17] E. Mills. Facebook suspends app that permitted peephole. http://news.cnet.com/8301-10784_3-9977762-7.html, 2008.

[18] C. Mui. Facebook's Privacy Issues Are Even Deeper Than We Knew. http://www.forbes.com/sites/chunkamui/2011/08/08/facebooks-privacy-issues-are-even-deeper-than-we-knew/, 2011.

[19] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[20] M. Shehab, A. C. Squicciarini, and G. J. Ahn. Beyond user-to-user access control for online social networks. In *ICICS '08: Proceedings of the 10th International Conference on Information and Communications Security*, pages 174–189, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] K. Singh, S. Bhola, and W. Lee. xbook: Redesigning privacy control in social networking platforms. In *Proceedings of the 18th Usenix Security Symposium*, August 2009.