PoX: Protecting Users from Malicious Facebook Applications

Manuel Egele*†, Andreas Moser*†, Christopher Kruegel†, and Engin Kirda‡

* Vienna University of Technology, Austria
{manuel,andy}@seclab.tuwien.ac.at

† University of California, Santa Barbara
{maeg,chris}@cs.ucsb.edu

‡ Northeastern University, Boston
ek@ccs.neu.edu

***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, existing mechanisms are not convincing. Therefore, we introduce PoX, an extension for Facebook that makes all 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 privacy-relevant improvements to be implemented by Facebook itself.

I. Introduction

Social networks have recently enjoyed tremendous success and growth. Statistics for Facebook, arguably the most popular social network, indicate that its user base now exceeds 500 million users [1]. 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.

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 [2]. 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 [3]. In fact, recent Facebook statistics [1] 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.

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 [4], [5], [6] 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 [5], 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. A more recent incident [6] involved some of the most popular Facebook applications transmitting user information to advertising and Internet tracking companies. Clearly, such incidents, which result in wide media coverage, make users aware of the need for improved access control for third-party applications. The latest incident even caught the attention of US political leaders who wrote a letter [7] to Facebook inquiring the company's privacy practices. Furthermore, Facebook recently outraged civil liberty campaigners by introducing new privacy settings that, while seemingly improving the users' privacy, dramatically

increase the amount of personal information users share publicly by default [8]. Additionally, in a recent interview with Facebook CEO Mark Zuckerberg, he stated that privacy is no longer a "social norm" [9]. As reaction to raised criticism, Facebook implemented coarse grained access control mechanisms for third party applications. However, we think that this approach is not far-reaching enough. First, profile information is grouped together too coarse grained. A user's "Basic Information", for example, includes the name, profile picture, all the user's networks, and friends. Thus, a user who is willing to share her name but does not want to expose her friend's information to an application could not use such an application. Second, all applications that were installed by the user before Facebook introduced their access-control changes continue to have unrestricted access to the user's profile data.

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. 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.

II. Background on Facebook applications

The first step to create a Facebook application requires the developer to register the application with Facebook. Each application is assigned an application-id and a private application key. All communication between the application and Facebook's servers has to be signed with this key.

A user can install an application by visiting the application's landing page, and accepting the dialog specifying the access rights of the application. However, the user can only accept or cancel the dialog. It is not possible to selectively grant or deny access to individual profile information.

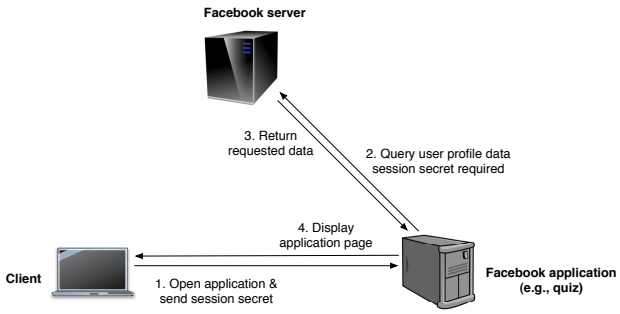


Figure 1: Data-flow in Facebook

Figure 1 illustrates how information is transmitted if a user authorized an application to access her profile. Note that even giving an application access to "Basic Information" exposes all of a user's friends to this application too.

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. 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).

III. PoX design

As mentioned previously, current Facebook applications communicate directly with the Facebook servers, and legacy applications even 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. Since such a proxy would have to be trusted by the user, such an approach would violate our second requirement that states that no additional trusted component can be introduced.

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. 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.

The remainder of this section elaborates on some of the implementation decisions we made during the development of our PoX prototype.

A. 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

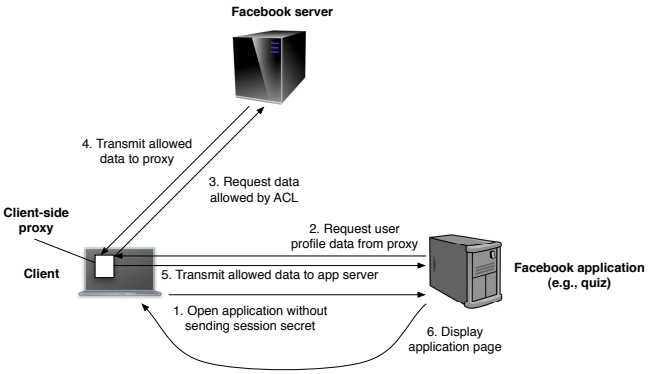


Figure 2: Modified data-flow with PoX

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 (i.e., the missing secret will prevent the application from retrieving 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.

B. 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.

Initially, the proxy retrieves the current access control list (ACL) for the user. 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.

Once loaded, the proxy waits for requests from Facebook applications that require access to profile 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.

After sanitizing the request, the proxy forwards the modified request to the Facebook server, and the result is relayed back to the application.

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.

Note that the user is not required to trust this proxy application more than any other Facebook 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.

C. Server to client communication

The PoX proxy executes entirely in the web browser of the Facebook user. It is thus 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).

To keep the proxy simple and independent of proprietary protocols (e.g., Flash), 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 (e.g., 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. To request profile data from the PoX client, the server resumes the script and outputs its request. The request is transferred to the client where it is subsequently processed.

D. 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. To seamlessly integrate PoX-aware and non-PoX-aware clients, 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 unmodified code paths where profile data requests are sent to the Facebook servers directly.

IV. 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 library, the system proposed in this paper can actually outperform the method currently used by Facebook for subsequent data requests.

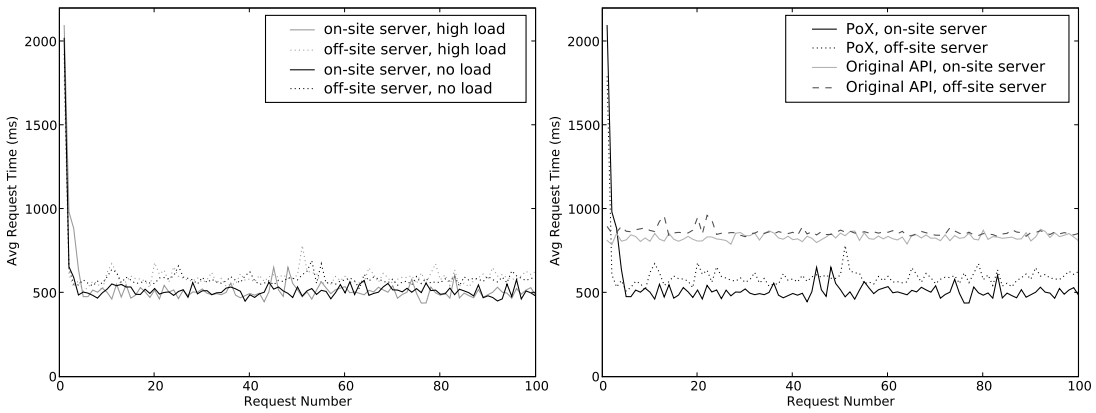
A. Performance of PoX

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 a 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 3a 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



(a) Request times for high / low load

(b) Request times for PoX and the original FB API

Figure 3: PoX Performance.

applications. Furthermore, results will show that the PoX library can handle the load of 60 concurrent users with ease.

Figure 3a 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 to parse Facebook profile data. 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.

B. Comparing PoX to the original Facebook library

Figure 3b 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 IV-A). 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 IV-A. 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 3b 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. 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).

V. Related work

Users and their profile information stored on social networks are at risk. For example, Bilge et al. [10] 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. [11] 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 [12], 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 application framework introduced in [13] is designed to keep all personal profile data confined. To this end, the xBook framework provides a restricted JavaScript environment based on ADSafe [14], 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. [15] introduce an three step approach for Facebook application access control. First, upon registration, each application has to submit a so-called *application sheet*, specifying the data needs for this applications. The second step consists of 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 extensive support from any social network that decides to implement it (e.g., filtering requests with regard to the user sheet). 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. [16] propose flyByNight, a cryptographic system that encrypts all communication between users on the Facebook Platform. 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. 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. [17]. 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.

VI. 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, current access controls for Facebook applications are too coarse grained and even non existent for legacy applications. 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] "Facebook statistics," http://www.facebook.com/press/info.php?statistics.

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

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

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

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

[6] E. Steel and G. A. Fowler, "Facebook in online privacy breach; applications transmitting identifying information," http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html, 2010.

[7] E. J. Markey and J. Barton, "Letter to mr. zuckerberg," http://markey.house.gov/docs/letter_-_facebook_-_post_wsj_-_10-18-10.pdf, 2010.

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

[9] "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.

[10] 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*. New York, NY, USA: ACM, 2009, pp. 551–560.

[11] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Commun. ACM*, vol. 50, no. 10, pp. 94–100, 2007.

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

[13] 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. [Online]. Available: http://www.cc.gatech.edu/grads/k/ksingh/publication/sec09-xbook.pdf

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

[15] 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*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 174–189. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88625-9_12

[16] 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*. New York, NY, USA: ACM, 2008, pp. 1–8.

[17] 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.