Workshop on Web 2.0 Security and Privacy. Oakland, CA. 22 May 2008.

Privacy Protection for Social Networking Platforms

Adrienne Felt
University of Virginia
Charlottesville, VA
felt@virginia.edu

David Evans
University of Virginia
Charlottesville, VA
evans@cs.virginia.edu

ABSTRACT

Social networking platforms integrate third-party content into social networking sites and give third-party developers access to user data. These open interfaces enable popular site enhancements but pose serious privacy risks by exposing user data to third-party developers. We address the privacy risks associated with social networking APIs by presenting a *privacy-by-proxy* design for a privacy-preserving API. Our design is motivated by an analysis of the data needs and uses of Facebook applications. We studied 150 popular Facebook applications and found that nearly all applications could maintain their functionality using a limited interface that only provides access to an anonymized social graph and placeholders for user data. Since the platform host can control the third party applications' output, privacy-by-proxy can be accomplished by using new tags and data transformations without major changes to either the platform architecture or applications.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Service—*Data Sharing*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*

General Terms

Management, Design, Security

Keywords

Social networking, privacy, data mining

1. INTRODUCTION

Content sharing services have made social networking sites immensely popular. Users view their profiles on social networking sites as a form of self-expression, but these profiles also have commercial value. To allay fears of privacy violations, social networking sites provide users with access control settings to place restrictions on who may view their personal information. The introduction of open APIs to social networks, however, has created a way to circumvent access control settings and increased the ease with which such attacks can be executed.

Facebook released the first social networking API for third-party development in May 2007. Applications built using this API pose serious privacy concerns: an "installed" application acquires the privileges of the profile owner and can query the API for personal information of the user and members of the user's network. The information available to developers includes hometowns, dating preferences, and music tastes. A Facebook demographics study of CMU students found that 89% disclosed their full birth date and gender to their network, and 46% also posted their current residence [7]. Thus, the information on most user profiles is sufficient to uniquely identify their owners, even with the name removed [15]. This loss of privacy directly impacts users. It is also of concern to social networking site operators: if advertisers can independently identify "desirable" users, the importance of the social networking site intermediary diminishes.

Since the release of the Facebook Platform, nineteen other sites have joined together to support Google's OpenSocial, a cross-site social network development platform [8]. Several sites have implemented restricted versions of OpenSocial, which is still in alpha release. Like Facebook, they support the distribution of social graph and personal user data. These new APIs suffer from the same privacy concerns as Facebook.

Due to the popularity of the Facebook Platform and the ongoing development of new APIs, it is important to address the privacy risks inherent in exposing user data to third-party applications. The next section provides background information on the structure of social networking platforms, and Section 3 discusses how platforms violate user privacy expectations. We then present an analysis of Facebook applications' information needs, finding that most applications use information in a predictable and limited way (Section 4).

Section 5 introduces *privacy-by-proxy*, an alternative platform design that prevents third parties from ob-

taining real user data. This design upholds users' privacy expectations while providing sufficient functionality for nearly all current Facebook applications. It hinges on two observations: social networking sites can manipulate application output, and applications need limited information. Section 6 analyzes our design from the perspective of an attacker, a developer, and a host site. Section 7 describes related work on privacy protection, and Section 8 concludes.

2. SOCIAL NETWORK PLATFORMS

This section describes the architectures of the Facebook Platform (Section 2.1) and OpenSocial (Section 2.2). The Facebook Platform is the more popular (and only complete) social networking site platform.

2.1 The Facebook Platform

Facebook is a popular social networking site with over thirty million users [1]. Users can see the profiles of their friends and network (e.g., college) members. Profiles include photos, dating preferences, birthdays, etc. Since the launch of the Facebook Platform, profiles can also display third-party gadgets.

Facebook applications have two core components: a homepage and a profile box. The homepage is located at a Facebook URL and appears to be part of the Facebook site. Developers may choose whether their homepage content is proxied through Facebook or isolated in an iframe. Proxied content is written in Facebook Markup Language (FBML), a "safe" subset of HTML and CSS extended with proprietary Facebook tags. Content in an iframe may contain arbitrary executable code but lags behind proxying in popularity due to the convenience of FBML. Profile box ("gadget") content must always be written in FBML.

Figure 1 shows how third party content is integrated into profiles. In Step 1, a user interacts with an application's homepage through the Facebook website. The Facebook server passes the user's ID and session key to the third party (Step 2). The application running on the third party's server requests data from Facebook servers using a Representational State Transfer (REST) interface. Querying the server requires a valid user session key and application secret. Applications can request detailed information about the user, the user's friends, and members of the user's networks.

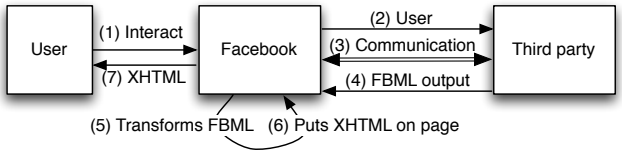

Figure 1: Application installation.

In Step 4, the application sends its output back to the Facebook server for display. The output is stripped of executable code to prevent cross-site scripting. Instead of JavaScript, FBML has "mock AJAX" FBML tags and Facebook JavaScript (FBJS), both of which are transformed by the Facebook server into JavaScript. The post-processing phase also reduces the server load, provides useful scripts to developers, and hides the identity of the viewer from the third party. API calls can be avoided with tags like `<fb:name id="[$id]" />`, which displays the name of a user and a link to her profile. Scripts like `<fb:friend-selector>`, a type-ahead user search box, provide developers with a rich interface through simple tags. In steps 6 and 7, the XHTML content is incorporated in the response page and served to the client. Profile box content is set for a user with an API call and will remain cached until it is reset by another API call.

When a client requests a profile page, Facebook examines the cached third party content to see if it contains viewer-dependent tags (such as the `<fb:visible-to-friends>` tag that determines whether the viewer may see the enclosed content). If so, it completes the output transformation. The content is then returned to the viewer along with the rest of the profile. Profile viewers are not identified to the third party unless they specifically log into the application through an AJAX window provided by Facebook.

2.2 OpenSocial

OpenSocial provides a set of APIs for its partner sites (which it refers to as "containers") to implement. An application that is built for one container should run with few modifications on other partner sites. The APIs give third parties access to social graph and personal user data, and containers can give gadgets access to canvas pages and profiles (like Facebook). Several sites have added limited support for the OpenSocial APIs.

Gadgets are XML files with JavaScript and HTML that can be hosted anywhere. The container's server fetches the XML on behalf of the user and renders it. (This is often done prior to load time for the benefits of caching.) To prevent cross-site scripting and request forging attacks, the gadgets are typically imported into iframes that are hosted on separate domains (e.g., MySpace profiles are served on `myspace.com` but gadgets are on `api.msappspace.com`).

In the future, containers will be able to add gadgets with JavaScript directly into profile pages [6]. The Google Caja project defines an object-capability subset of JavaScript [10]. It enforces limitations on regular JavaScript with static and dynamic JavaScript checks. The JavaScript in a gadget will be transformed accordingly so that iframes (and their associated additional latency) are not required. This functionality is in the

specifications but is not supported yet.

OpenSocial has the same privacy risks as Facebook, since gadgets can request and store user data on external servers. They also have architectural similarities: both proxy third-party gadget content, and OpenSocial will be adding a pre-processing transformation step for JavaScript. This means that privacy-by-proxy, as described in Section 5, could be implemented in OpenSocial as well as Facebook.

3. PRIVACY IN SOCIAL NETWORKS

Next, we survey user privacy expectations for social networking sites. The goal of our design is to provide a platform that satisfies these expectations by guaranteeing that a user's data is only accessible to another user through the programming platform if it can be viewed by that same user through the standard web interface.

3.1 User Expectations

Past work demonstrates that users have strong expectations for privacy on social networking sites. A social phishing attack experiment [9] harvested personal data from a social networking site by screen scraping and used it to construct phishing e-mails. Users expressed anger that their information could be collected and used in such a manner without consequences, with users protesting that their profile data should be protected [9]. Additionally, when Facebook launched the Facebook Beacon, which correlated activities on other websites with Facebook profiles, user outrage forced Facebook to retract the program [13]. These strong reactions illustrate that users believe social networking sites have a responsibility to shield their data.

Privacy expectations in social networks are based on relationships. Typical social networks support friends and networks with privileged access.

Friends. Friendships are a defining characteristic of social networking sites, and friends receive access to personal data. Friendships require acceptance by both parties. Some sites may extend privileges to the second or third degrees of connection.

Networks. Social networks also support networks, where members have some access to each other. Bebo and Facebook associate access controls with school attendance. Alternately, self-defined regions can be considered a network, and privacy controls may be associated with the chosen location (e.g., Friendster users can limit their profile visibility to certain continents).

Public visibility. Sites define some subset of a profile (such as the user's name and affiliation) visible by default for searching and identification. Most sites also allow users to relax or strengthen their definition of public information.

3.2 Platform Privacy

Application platforms give developers access to data that would not otherwise be available to them through the user interface. OpenSocial sites let third parties access the information of users who directly "install" an application. Users cannot add an application to their profile without also granting it data permissions. Facebook additionally gives third parties second-degree access: when Jane installs an application, the third party can also request information about Jane's friends and fellow network members. It is likely that OpenSocial containers will allow second-degree access in the future, since it is used by many Facebook applications (as shown in Section 4).

Application developers can therefore see user data that they would not otherwise have access to since they are not friends with users at the web interface level. Unlike regular friend relationships, the relationship is neither symmetric nor transparent; the user does not necessarily know who the application owner is. The current binary privacy situation (installed or not installed) forces users to give away access to data that may not even be needed. Second-degree permissions add another layer of complexity. Setting up new privacy settings for each application, however, could easily grow convoluted. Instead, regular web interface settings should be enforced with applications.

Host social networking sites have a responsibility to protect the user data that has been entrusted to them. The current approach is to display a *Terms of Service* (TOS) warning screen every time a user adds an application. Since this TOS agreement is present on *every* application, and the majority of applications do not appear to use user personal data, the warning becomes meaningless. Sites also have Terms of Service agreements for third parties, but the host site has no way of monitoring the path of information once it has been released from the database.

Social networking sites should not rely on untrusted third parties following their TOS agreements to protect user privacy. In fact, our survey of Facebook applications reveals several that clearly violate it through their externally-visible behavior, and there is no way to know how many others violate it with internal data collection. Instead, privacy policies should be enforced by the platform and applied to all data that has been entrusted to the social networking site.

4. APPLICATIONS

We studied the 150 most popular applications (collected from `http://www.facebook.com/apps/` on 22 Oct 2007) to determine their information requirements and behavior. To conduct the survey we installed each application on a user account with the minimum amount of information filled out. If an application re-

Information Used		Applications	
None		13	(8.7%)
Public	Yours	133	(88.7%)
	Friends	99	(66.0%)
	Strangers	51	(34.0%)
	Any	**133**	**(88.7%)**
Private	Yours	12	(8.0%)
	Friends	10	(6.7%)
	Strangers	7	(4.6%)
	Any	**14**	**(9.3%)**

Table 1: Application information requirements. **Data for 150 applications. Categories may overlap. "Yours" refers to the person using the application; "Friends" and "Strangers" follow from their relationship to that person.**

quested more data, broke, or required the interaction of multiple users, we installed it on a fully filled-out second account to observe the difference. We explored the features of each application to look for the appearance of data or use of the social graph. Although it is not possible to determine exactly how an application uses information without access to its source code, the simplicity and limited interactivity of Facebook applications gives us reasonably high confidence that this method captures application functionality well enough to understand their data use.

4.1 Data Usage

We found that applications generally do not need the extensive personal information that is available to them. Although two-thirds of applications depend on public friend data, far fewer require access to private data. Public data refers to information used publicly for identification or searching.

Table 1 summarizes the results of our analysis of the 150 most popular applications. Only 14 applications require any private data, meaning that over 90% of applications have unnecessary access to private data. Of the 14 applications that use private data, four clearly violate the Facebook Terms of Service: they pull user data and add it to an in-application profile, making it visible to other application users who would not otherwise have the ability to view it.

Nine applications (6.0%) performed processing on data beyond just displaying it. Two of these use public information, and 7 of them processed private data (e.g., two use birth dates to generate horoscopes). The most sophisticated use of data is an application that plots user locations on a map.

Regardless of their information needs, all of the applications require users to install or log into the application and thus provide access to private user information. This includes the 13 applications that use no personal information at all and the 141 applications that do nothing with data other than display it.

Behavior	Applications
Wall Posting	15 (10.0%)
Messages/Gifts	37 (24.7%)
Viral recruiting	20 (13.3%)
Top Scorers	30 (20.0%)

Table 2: Common application behaviors. **Applications may be counted in more than one category if they exhibit multiple behaviors.**

The application survey also included two applications that use Flash to query the API. Although we focus on FBML, Facebook also allows Flash to connect to the API. Flash applications are inherently insecure because any user can decompile them to obtain the application's secret key. Because of this, few developers use Flash to query the API; instead, they use Flash in combination with FBML/HTML.

4.2 Behavior

Table 2 summarizes behaviors commonly observed in the sample applications. Our platform must support these popular behaviors to be a feasible choice as a development platform. Common application features include wall posting, viral recruiting, global top scores for games, and message passing behaviors as described next.

Wall posting. A wall-posting application lets users leave publicly visible messages on each other's profile pages. If Alice has the application installed, her profile displays past messages and a form for visitors to leave messages.

Gifting and messaging. A gift-sending or message-passing application lets users send each other small images and text. To send Bob a martini, Alice would visit her application homepage, select his name, and choose the martini. The gift would appear in Alice's sent list and Bob's received list.

Viral recruiting. Several popular applications' primary function is to spread through the social network. Users get points by inviting friends to install the application. The game may continue as a pyramid scheme.

Top scorers. Many applications are games users can play on the application homepage against other users. Scores appear on their profiles. Many games provide a leader board that displays the names of the top scoring players to all visitors.

In addition to these application types, general social graph connectivity information is needed by many applications (67 of the 150). Aspects of the real social graph must be available to applications in some way. Other tasks we encountered include uploading files, setting static content to a profile wall, or taking

quizzes. None of these other behaviors require access to data or social graph information beyond what we have already addressed.

5. PRIVACY-BY-PROXY

We now present a simple solution for providing privacy a site has the ability to transform the third-party output (as do Facebook and OpenSocial sites). Our goal is to simultaneously shield users' identities and provide applications with the capabilities identified by our survey, including the ability to display friends' information and traverse the social graph. As described in Section 5.1, user data may be displayed to users with appropriate permissions using tags that are replaced with real values before being shown to the user. This approach can protect personal data, but third party applications need direct access to the social graph information embodied in the user's friend list. This is accomplished privately with user and graph anonymization described in Section 5.2. Access to public data presents risks for exposing anonymized user identities, so we limit access to normally public information through the platform as described in Section 5.3.

5.1 Data Hiding

With privacy-by-proxy, the networking site does not provide any personal data to third party developers. Instead, applications display information to users with special markup tags like FBML. We extend the markup language to include tags that abstract user data and handle user input without providing private data to the application.

The simplest data tags take a user ID (encrypted as described in the next subsection) and private profile field, and are replaced by the server with the corresponding data. For example, to display a friend's birth date, an application would use `<uval id="[$id]" field="birthday" />`. For application homepages, a single permission check is done by the proxy server (e.g., Facebook) to decide if data should be displayed. If the user viewing the page has appropriate privileges to see the identified data, the data is retrieved and displayed. For content in profile gadgets, however, a second permission check is needed to ensure that the content *viewer* also has sufficient access to see the requested data. Personal data displayed in a profile gadget should always be in the intersection of the privileges of the profile owner and profile viewer.

Conditional tags (e.g., `<if-male>`) let applications display output that depends on private data unavailable to the application. The content within conditional blocks must be restricted to prevent applications from learning information about a user since anything that reveals to the third party application whether or not the if-block is followed would leak private data. The server must remove all forms and form elements from if-blocks. Images are permissible only if the social network server caches the images so no leaking requests are sent to external servers. (Facebook does this already to keep the third party from learning the IP addresses of viewers.)

Our design assumes the application cannot observe its output after it is transformed by the network server. This would be violated if an application could include scripting code in the output that could send information from the displayed output to an external server. FBML and FBJS are already specifically designed to prevent this; OpenSocial's planned JavaScript rewriting tool could be used similarly.

5.2 User Identification

Application users are identified to the third party with an application-specific ID number. These IDs must be consistent across all user sessions, without revealing the actual user identities to the application. Hence, we encrypt user IDs using a symmetric encryption function keyed with the application ID and a secret kept by the server. Since each application obtains user IDs encrypted with a different application-specific key, applications cannot collude by correlating encrypted user IDs or use the encrypted user IDs to identify global public user information.

Applications can request the friend and contact lists of a logged-in user through the API. All user IDs in query responses are transformed. Hence, an application cannot access actual user IDs, but can construct an anonymized view of the social graph starting from application users.

In addition to the simple data lookup tags, form elements need to be added to the markup language to allow user input to be part of the anonymous social graph scheme. A user needs to be able to select friends from a list. This can be accomplished with the transformation of a special form input tag (in the third party code) into a functioning HTML form input tag (in the final user-viewed content). When the form is submitted, the friend's encrypted ID is sent to the third party. Facebook currently has a user selection tag that accomplishes this, so the only change is for that tag to return application-encrypted IDs instead.

Profile viewers need some way to submit actions on other users' profiles such as making a wall post. This is done using an `<identify>` tag. When transformed, it is replaced with a form tag `<input type="text" id="vid" name="vid" value="[$id]" />`. The `$id` variable is the encrypted ID of the submitter.

5.3 Public Data

Some information is meant to be completely public for the purpose of user identification (e.g., user names

and networks), and this public data is displayed with the same tags as in Section 5.1. We place an artificial restriction on personally identifying public information to prevent de-anonymization. This restriction is necessary to prevent an attack in which an application developer installs the application on her own account and attempts to use it to learn the mapping between application-encrypted IDs and real user IDs. If arbitrary public information could be displayed, the attacker could create a page that enumerates through a list of application-encrypted user IDs, displaying each one using the `<uval id="[$id]" field="name" />` tag.

We combat public data attacks by limiting an application's ability to display public information to only those IDs in the current user's *contact list*, a new structure maintained by the social networking site. Contact lists are distinct from existing social networking site user lists and are always conservative. They prevent an attacker from seeing the names of arbitrary users by only allowing users to see the public names of users with whom they already have an explicit or implicit relationship. An explicit relationship is a friendship, so the contact list includes all of the user's friends. An implicit relationship is created when a user receives a message, gift, or other communication from a person. These implicit events can be recognized using the `<identify>` tag described in Section 5.2; submission of a form with this tag in it indicates that the social networking site should add the sender to the recipient's contact list. For example, an implicit relationship is established with a wall post. When Alice submits a form to write on Bob's wall, Alice is added to Bob's contact list so that Bob can see the identity of the wall poster.

Unlike friendships, contacts are one-way relationships. The one-way nature of the contact list is an important feature to prevent abuse and circumvention. If the list were reciprocal, spam bots would give attackers privileges. In the wall-posting example, Alice is added to Bob's contact list, but Bob is not added to Alice's. The display decision for public data is based on the contact list of the profile owner only. Thus, Bob's friend Charlie will be able to see Alice's name on Bob's wall even if Charlie has never interacted with Alice. Only the profile owner's privileges need to be restricted, because that prevents the attack.

6. ANALYSIS

The privacy-by-proxy design must resist attacks (Section 6.1), allow applications to function (Section 6.2), and be implementable with minimal performance overhead (Section 6.3).

6.1 Attack Analysis

We described how privacy-by-proxy resists basic attacks using public information in Section 5. Here, we consider the possibility of more sophisticated attacks that attempt to de-anonymize the encrypted user IDs. Pure social graph anonymity through pseudonyms is imperfect since it reveals aspects of the graph structure that may be used to expose user identities.

In the simplest graph attack, the attacker locates herself by being the first user to install the application. She could then compare her friends' friend lists (obtained through the web interface) to her own friends to look for patterns. This attack is fairly limited, however, since the chain of de-anonymization would end when she runs out of external information to compare the graph structure to. Identification can only take place when a node's surrounding structure is already known.

Attacks that rely on known data from the web interface (either from screen scraping or a conspiracy of users) face the same limitation. Correlating real structure and anonymous graph structure requires knowledge of the real structure, which means that the API is not revealing new data. Privacy-by-proxy does not leak any personal information that cannot be obtained through the web interface, thus satisfying the reasonable privacy expectations of users.

Our design still betrays information about the nature of the social graph as a whole. A popular application may be used by enough Facebook users to allow the application owner to construct a fairly complete social network graph. Structural facts like average connectivity can be computed, and nodes of high influence can be located in the graph (although their identity is not revealed). This leaked information, however, is of little commercial value compared to the original situation where third parties can connect graphical features with actual users and their personal data.

6.2 Application Functionality

Our privacy design enforces a conservative privacy policy, preventing applications from accessing personal data. Most existing Facebook applications could be supported using the additional markup tags introduced in Section 5.

Our application survey identified only 9 applications that need access to real data from Facebook for external computation. Seven of them used private data, and 2 used public data. The small minority of applications that need to process real data would need to be redesigned to work with our platform. The preferred solution is for applications that need private data to directly prompt users to provide that information, instead of obtaining it from the social network API. This shifts the liability of protecting the data from the social networking site to the third party, and users will be more informed about their data use if they are entering it at its point of use. Three of the 7 personal

data applications already prompt users with a form to add, correct, or delete data, with the user's personal profile data only used to seed the default values of the form. An alternate solution would be to provide a richer (privacy-compromising) interface to applications that have been installed through a more extensive agreement process (possibly involving extra monitoring by the social network operator).

The contact list constraint on public information affects a larger number of applications, but the severity of the restriction is mild. Of the sample applications, 34% displayed the public information of strangers. This constraint can be mitigated with contact lists or the use of non-unique data in public badges. The amount of information that identifies a user in context (e.g., message boards) is less than what is necessary to uniquely identify a specific user in a public data attack. Alternately, users could enter their own names or create an in-application profile. In-application profiles containing application-specific data are already common.

6.3 Performance

Privacy-by-proxy does not impose a substantial burden on the application or social networking site servers. New transformations are simply added to the existing post-processing or caching step, and network communications are decreased.

To evaluate the feasibility of our approach, we constructed a simple mock social network application and measured its performance. Without access to a large-scale social network, it is difficult to accurately predict the impact of our design on server load. To get a rough estimate of the cost of supporting privacy-by-proxy, we conducted some experiments with our mock application server, which includes a simple transformer.

We built a profile for "Anne", who was given 500 friends and 750 contact list members. Additionally, there were 249 strangers. The database maintained tables for user data, friend lists, and contact lists, all of which were filled with pieces of fake data. Anne's profile contains a third party gadget (modeled by an application running on the same server) that requests Anne's friend and contact lists, and it has access to the remaining stranger IDs as if they were other application users. User IDs are encrypted using AES through the PHP Mcrypt extension. On our test machine, a 1.86 GHz Intel Core 2 Duo CPU with 2GB RAM, it takes 0.7 milliseconds on average to encrypt a user ID, which supports over 1400 ID encryptions per second.

The third party application iterates through friend, contact, and stranger lists, requesting their names, networks, and hometowns using fake markup tags. For each tag that is matched, the ID is compared to contact and friend lists for a permissions check. If permissions are satisfied, the data is retrieved from the database. The average time for a permissions failure is 2.3 milliseconds, and the average total time for a successful data lookup is 3.6 milliseconds.

On a commercial server running highly optimized permission and data lookup algorithms, the cost will be even smaller. Although it is not possible to precisely determine the cost of our approach without a large-scale experiment, both the details of our design, and the results from these experiments, support the conclusion that privacy-by-proxy would not substantially increase server load or latency experienced by users.

7. RELATED WORK

Substantial work has been done on the problem of preserving statistical characteristics of data sets without revealing the unique identity of database members [4, 14, 2]. Although this work also has the goal of preserving privacy under queries, social networking site applications need more than statistical information to provide the essential functionality of linking users.

Secure function evaluation computes $f(x_1,x_2)$ without revealing x_1 or x_2 [5]. Selective private function evaluation is a specific instance of secure function evaluation in which a client wishes to compute some function over a server's data set, but neither party should learn more than the response [4]. This might be useful for large-scale studies of social network properties of a social network, but does not provide the information needed for applications.

Past work in the field of untrusted databases assumes an untrusted server and a trusted client. Security in this model can be done with information dissociation or encryption [11, 16, 12]. Our scenario is the reverse.

Backstrom, Dwork, and Kleinberg describe sophisticated active and passive attacks on anonymous social graphs. Their goal is to determine whether two given known nodes are connected, and they assume the attacker obtains a static copy of the entire social graph [3]. In an active attack, the attacker is aware of the pending release of the social graph and is able to insert nodes. The active attacker creates a unique subgraph to locate herself in the graph and friends the target nodes. Once she has found herself in the graph, the attacker knows which nodes belong to the targets. A passive attack occurs after the social graph is procured. The attacker colludes with a coalition of the targets' friends that defines a unique subgraph; once the coalition is found, the targets are found.

Both their goal and assumptions are inappropriate for our social networking site privacy model. The goal of an attacker in our model is to start with the social graph and connect it to a large number of real users, as opposed to starting with a small number of known nodes and trying to find them in the graph. Also, our attacker does not have a complete social graph. Friend-

ships could not be realistically used to mount an attack on our model, because an attacker who can become friends with the targeted users would render the attack unnecessary by directly gaining access to their data through the web interface. Because contact lists are unidirectional, they cannot be exploited either; if the attacker is interacting with users' profile gadgets, then the attacker already has access to their profiles.

8. CONCLUSIONS

The Facebook Platform has proven wildly popular, and other large social networking sites are releasing similar public APIs. Current platforms cannot enforce their privacy policy with third-party applications, however, thereby increasing the risk of malicious data harvesting. Our privacy standard holds that access controls established for the web interface should be enforced for third-party applications.

Using the privacy-by-proxy approach, a social networking site has control over the application output and can support applications without compromising user data. Our design is motivated by how current Facebook applications (which were developed with an open, privacy-compromising interface) actually use information. Our findings indicate that a simple privacy-preserving interface, combined with our privacy-by-proxy approach is able to support nearly all existing applications while providing privacy protection.

Admittedly, open social network platforms are new and it is possible that a privacy-preserving interface would inhibit potentially valuable future applications more than is apparent from out study of current Facebook applications. There is an inherent tradeoff between protecting privacy, which requires limiting access to information, and providing a rich interface; our work seeks a compromise.

Acknowledgments

Our thanks to Andrew Spisak for helping with the Facebook application survey and Chris Soghoian for bringing the policy issue to public attention. This work was partially funded by the NSF CyberTrust program, award CNS 0627523.

9. REFERENCES

[1] C. Abram. Thirty Million on Facebook. *The Facebook Blog*, 10 July 2007.

[2] N. R. Adam and J. C. Wortmann. Security-Control Methods for Statistical Databases: A Comparative Study. In *ACM Computing Surveys*, volume 21, pages 515–556, 1989.

[3] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Stenography. In *16th International World Wide Web Conference*, 2007.

[4] I. Dinur and K. Nissim. Revealing Information While Preserving Privacy. In *Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 202–210, 2003.

[5] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *19th Annual ACM Conference on Theory of Computing*, pages 218–229, 1987.

[6] Google. People Data API Developer's Guide: Protocol. *Open Social Developer's Guide*, 9 December 2007.

[7] R. Gross and A. Acquisiti. Information Relevation and Privacy in Online Social Networks. In *Workshop on Privacy in the Electronic Society*, 2005.

[8] M. Helft and B. Stone. MySpace Joins Google Alliance to Counter Facebook. *The New York Times*, 2 November 2007.

[9] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 50, 2006.

[10] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja: Safe active content in sanitized JavaScript. Draft tech report., November 24 2007.

[11] A. Motro and F. Parisi-Presicce. Blind Custodians: A Database Service Architecture that Supports Privacy Without Encryption. In *19th Annual IFIP WG 11.3 Working Conference on Database and Application Security*, 2005.

[12] G. Ozsoyoglu, D. A. Singer, and S. S. Chung. Anti-Tamper Databases: Querying Encrypted Databases. In *17th Annual IFIP WG 11.3 Working Conference on Database and Application Security*, 2003.

[13] B. Schiffman. Facebook CEO Apologizes, Lets Users Turn Off Beacon. *Wired*, 2007.

[14] A. Shoshani. Statistical Databases: Characteristics, Problems and Some Solutions. In *8th International Conference on Very Large Data Bases*, pages 208–222, 1982.

[15] L. Sweeney. Uniqueness of Simple Demographics in the U.S. Population. Technical report, Carnegie Mellon University, Laboratory for International Data Privacy, Pittsburgh, PA, 2000.

[16] Z. Yang, S. Zhong, and R. Wright. Privacy-Preserving Queries on Encrypted Data. In *11th European Symposium on Research in Security*, 2006.