Preserving User Privacy from Third-party Applications in Online Social Networks

Yuan Cheng
ycheng@cs.utsa.edu

Jaehong Park
jae.park@utsa.edu

Ravi Sandhu
ravi.sandhu@utsa.edu

Institute for Cyber Security
University of Texas at San Antonio
San Antonio, TX, USA

ABSTRACT

Online social networks (OSNs) facilitate many third-party applications (TPAs) that offer users additional functionality and services. However, they also pose serious user privacy risk as current OSNs provide little control over disclosure of user data to TPAs. Addressing the privacy and security issues related to TPAs (and the underlying social networking platforms) requires solutions beyond a simple all-or-nothing strategy. In this paper, we outline an access control framework that provides users flexible controls over how TPAs can access user data and activities in OSNs while still retaining the functionality of TPAs. The proposed framework specifically allows TPAs to utilize some private data without actually transmitting this data to TPAs. Our approach determines access from TPAs based on user-specified policies in terms of relationships between the user and the application.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Security

Keywords

Privacy, Online Social Networks, Social Applications

1. INTRODUCTION

To offer richer functionality, many online social networks (OSNs) have launched social networking platforms that enable third-party developers to contribute applications to the social network through the use of APIs (application programming interfaces). With the support of OSN platforms, third-party applications (TPAs) have become highly popular in a very short period of time.

The emergence of TPAs also poses severe privacy risks to users. TPAs need to consume users' and their friends' data to provide extra functionality to users. Under the current circumstance, TPAs usually receive privileges equal to the TPA users with respect to social graph traversal, and thereby gain access to an abundance of users' information regardless of the actual legitimate needs. Moreover, these applications are available via OSNs but are running on external servers outside the OSN's control. Once they acquire the data, they can use or dispose it in whatever way they want without user or OSN consent. There is no control regarding the usage of user data once it is released to the TPA. The developers of TPAs can aggregate such data and accrue benefit by using or selling the data. The most common approach adopted by existing OSNs is a simple all-or-nothing strategy: a user must agree to allow the application to access some subsets of her profile information before installing and using it. Thus, it is difficult for users to know and control how their information is accessed by those various external parties. The only choice for the user to not provide such data is to not use the application. In particular, TPA providers access the contact list of the TPA users and fetch information about the friends of the users, even though those friends did not install the application themselves or consent to this access.

Most previous research on access control in OSNs concentrates on access between regular users in the system, leaving the issues of TPAs out of scope. However, TPAs do not behave like a regular user. They have the ability to aggregate a huge volume of information from their users and can do anything with the collected data without any controls or consent from users or OSNs. We believe OSNs need effective means to prevent privacy leakage for users, in addition to the terms of services and limitation of their APIs. This is likely to be in both OSNs' and TPAs' interests as more users are likely to use privacy preserving OSNs and TPAs.

In this paper, we address the issue of inappropriate exposure of user's information to TPAs. We present an access control framework that provides flexible and fine-grained controls on how TPAs can access OSN user's data. For this purpose, we classify TPAs into three categories: running outside of OSN, running inside of OSN, and hybrid where some modules are running inside of OSN while others are running outside. The fundamental idea of our approach is to constrain and consume the private information within the OSN system but only allowing privacy-nonsensitive information to be sent outside the OSN for necessary functionality. Therefore, applications or functions running outside the OSN may only receive privacy-nonsensitive data, while those running inside may consume raw private data under the surveillance of the OSN system but are not allowed to transmit the data outside. We also define a relationship-based access control policy language to the framework that allows users to specify how TPAs can access their data in terms of the relationships among users and applications.

2. BACKGROUND

This section provides an overview of OSN platforms, describes the privacy issues in those platforms, and reviews the previous literature on these issues.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

Table 1: Comparison with previous solutions

	Felt [9]	Antho. [3]	Singh [14]	Viswa. [15]	Shehab [13]	Besmer [4]	Egele [8]	ours
Communication Intercepter	✓	✓					✓	
Information Flow Control			✓	✓				
Data Generalization					✓			
User Specified Privacy Preference		✓			✓	✓	✓	✓
User-To-Application Policy Model						✓		✓
Separating Components of Application			✓	✓				✓

2.1 Social Networking Platforms

Many of the popular OSNs have released web APIs to allow third-party developers and websites to implement their own services, which can utilize and aggregate user information and activities in OSNs. Figure 1 presents a typical architecture of current social networking platforms, where TPAs are designed in accordance with APIs and can access user data through APIs. Typically, these applications are integrated in an OSN site but rely on their own external server for running the application.

The Facebook Platform and Google's OpenSocial are two of the leading forces in the business of social networking platforms. The Facebook Platform [1] is a proprietary software environment for launching TPAs in Facebook. The core of Facebook Platform is the Graph API, which forms the primary way of retrieving and posting data in Facebook. It allows developers to define objects and actions in the social graph, and to create new instances of objects and actions. Google's OpenSocial [2] is an open source cross-platform competitor to the Facebook Platform for building social applications. It defines a common API for applications across multiple websites, through which OSNs can grant applications access to the social graph as well as messaging service and update feeds. Its greatest advantages over Facebook Platform is its interoperability within the context of multiple OSNs.

2.2 Privacy Issues with Current Platforms

On current platforms, for an OSN user to install an application, a dialog is displayed to her showing that the application requests an access to a list of her profile data. Different from access control policies for user-to-user interactions, the user cannot express her privacy preferences for data that are accessed by applications. In fact, many OSNs currently only provide an all-or-nothing policy when it comes to application-to-user interactions, without letting OSN users specify which information TPAs can access. Thus, even if the application only needs one piece of profile data, the user has to agree to grant the application full access to her profile data; otherwise, she cannot use the application at all. Recently, some OSNs offer users more customized privacy management regarding TPAs than before, where users are allowed to opt-in or opt-out some categories of profile data. However, the essential problem still remains. First, the coarse-grained privacy control does not support users to specify access control policies for each piece of data users own, nor does it distinguish different types of actions application can exercise on data. Second, users are not aware of what kind of data the application really needs. Applications can still ask for more data than they really need, which simply violates the principle of least privilege. We believe it is more appropriate to mediate access on a per request basis rather than approve access in advance during the time of installation. Third, some permissions are given by user's friend who installed the application, without user's knowledge. This poses a serious exposure to those users who are not willing or interested to use the applications.

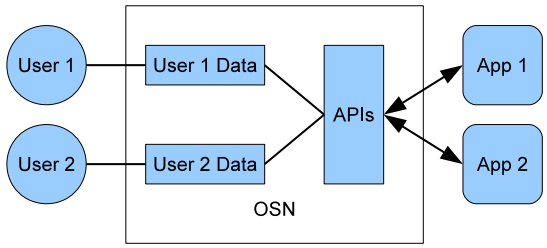

Figure 1: Typical OSN Architecture with TPA

TPAs run on their own servers or other hosting services that are beyond the control of OSN systems. Once applications get user information, they can aggregate the information for their own use, or sell it to other external parties. There exists no enforcement to police how applications and their developers use user data. The large number of privacy leakage incidents by applications in recent years raises OSN users' concerns on this matter, and suggests them not to completely trust these third-party applications.

It is difficult for user to know and control the various entities who can gain access to their information and to limit such flow without losing the various features brought by these third-party entities.

2.3 Related Works

There has been considerable work seeking to resolve the privacy issues regarding TPAs. Table 1 summarizes the characteristics of some solutions proposed recently.

All communication between applications and data in OSNs is encapsulated by APIs. However, current APIs are not designed with the purpose of access control in mind, letting TPAs access user data without obstruction. To prevent TPAs from directly interacting with user data through APIs, several proposed proxy designs intercept all requests from TPAs, exert users' privacy preference on data retrieved from the OSN, and then return the sanitized or dummy data to TPAs [3, 8, 9]. The enforcement of such designs varies from server-side proxy to client-side plug-in, according to the trusted entities the designs rely on.

Singh et al [14] presented xBook framwork to prohibit untrusted applications from leaking users' private information. TPAs are hosted on trusted xBook platform, which confines application execution and mediates information flow between different application components. Inspired by xBook, the system proposed by Viswanath et al [15] enforces sandbox structures on both server and client sides to restrict information flow from users to application developers. Our work shares a common design strategy with these two systems by separating applications into multiple components and restricting the information flow between components.

A number of researchers seek to support user-specified privacy preference over what information can be accessed by TPAs. The collaborative privacy management (CPM) framework [3] allows

Table 2: Strategy

Data Classification	Strategy
unnecessary & private	do not permit
unnecessary & non-sensitive	user's choice
essential & non-sensitive	transmittable outside of OSN
essential & private	processable within OSN

users to define privacy configurations and share them with others. In the approach presented by Shehab et al [13], users can choose to opt-in, opt-out, or generalize the application requested data to reflect their access preferences for the applications. PoX [8] uses a dedicated Facebook application to store access control lists for users, which indicate what application should be allowed to access what pieces of information. These proposals contrast with our work in that they do not offer a complete access control policy model for users to specify their privacy preferences. Besmer et al [4] introduced a new application-to-user policy that restricts application's access to user information while still allowing desirable functionality, which is most similar to ours. However, in addition to the policy model, we also provide a framework that separates application components for finer-tuned information flow controls.

3. THE FRAMEWORK

In this section, we present an access control framework that enforces better privacy preservation for users against third-party applications. Our goal is to prevent TPAs from learning user's private information while still maintaining the functionality of the applications.

3.1 Overview

As discussed earlier, existing access control mechanism gives applications too much freedom in accessing user's information. A survey conducted by Felt et al [9] indicated that 91% of the 150 top Facebook applications have unnecessary access to user's private data, violating the principle of least privilege. On the other hand, completely limiting the applications from user data or providing fake data to them may harm the functionality of applications or even their own business models.

To achieve a balance between application functionality and user privacy, we first present a taxonomy of user's information based on the value for applications and users, as shown in Table 2. User information can be classified into four categories. The first row indicates data that is inessential for application but of significance to user privacy. It is definitely unnecessary for TPAs to acquire such data, thus requests for the data will not be permitted in any circumstance. If the data is not necessary for running of application and is not private (Row 2), disclosing it does not breach privacy and the decision is up to user's own choice. For data essential for application, however, we need a special scheme to ensure the functionality without privacy leakage. Unlike other approaches that do not trust the OSN systems [12], we assume the OSN systems are always trustworthy with respect to user privacy, but TPA server may not be. User's private information should be kept away from these untrustworthy external servers. Among the data that is essential for application functionality, some are of less privacy concerns and thus accessible by untrustworthy external entities (Row 3), while others are sensitive private information that require proper protection against external entities (Row 4).

The premise of our idea is that running an application may not require users' private information, and even if it requires it not every function of the application needs the private information. Therefore, to preserve user privacy and application functionality, our fundamental strategy is to leave private data within OSN system and to allow external servers of TPAs to retrieve non-private data.

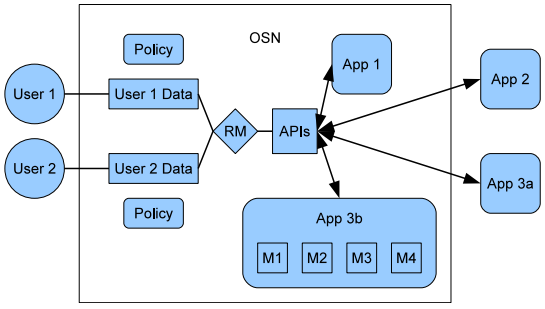

Figure 2: Proposed Architecture

3.2 The Design

We introduce an access control framework, which modifies the current architecture of Figure 1 to accommodate our strategy. As shown in Figure 2, the architecture of the proposed framework has a major distinction from the current architecture: in our framework, applications can be divided into two components, internal component and external component. An application requires a set of functional modules. These functional modules can run as either internal or external components. The modules running as an internal component can receive necessary information only through OSN API, hence are trusted not to transmit any sensitive information to the TPA server. The modules running as an external component are managed by the TPA developer, hence not trusted. TPA developers can decide whether a functional module should be in an internal or external component based on what kind of data the module consumes and, if users' private data is used by the module, whether the users' private data need to be stored at the TPA server or not. If a module utilizes privacy sensitive information but the corresponding application does not need to collect users' private information in its server, it should be the TPA developers' (and even OSN providers') interest to let the module reside within OSN as this is likely to attract more privacy-aware users to try this application.

As mentioned earlier, Figure 2 exhibits three different forms of applications based on the component types used: the one entirely running inside OSN (e.g., App 1), the one entirely running on external server (e.g., App 2), and a hybrid that includes both internal and external components (e.g., App 3). In the current architecture, OSN systems only provide APIs for TPAs to access user's information. Almost all of the existing TPAs are hosted on their own external servers, and therefore belong to the second category. However, if OSNs accommodate our design framework, the majority of TPAs are likely to operate in the hybrid form where internal components run the functions that utilize users' privacy sensitive information but do not need to store this information at TPA server, while the remaining functions are kept at external server.

Apart from the separation of application functions, the framework also includes three primary components: reference monitor (RM), API and policies. In Figure 2, Apps can request certain information only through APIs which limit the apps' communication capability. The RM evaluates the requests received by the APIs then approves or denies them according to the corresponding policies. In this framework, application policies are specified by the resource owner. We will describe our policy model in detail in the next section.

3.3 The Application Components

As mentioned earlier, in this framework, the majority of TPAs are likely to run in hybrid form, consisting of both internal and external components. The internal and external components of an application are hosted on servers with different trustworthiness, and can access various user information with different privacy sensitivity. Hence, the communications between these components needs to be precisely managed to prevent unnecessary information leakage. In the proposed social networking platform, the internal and external components can communicate with each other only through OSN-specified APIs which allows only certain types of pre-defined communications.

Between the internal and external components, we define three types of communications: communications with system calls, non-private data and private data. The communication using system calls is a mechanism used by both internal and external functional modules for sending a message (e.g., a function call) to other components. It is used to trigger and handle specific events, and does not contain information about users. In this framework, we allow two-way system call communications between the components. A non-private data communication refers to a message transmission between application components where the message contains non-sensitive information fetched from users, such as keystrokes, mouse clicks, or puzzle results. This type of communication needs to be used if the messages that need to be shared cannot be transmitted using a system call. A communication with private data, on the other hand, is a message transmission between internal and external components that includes users' private information.

Note that internal modules typically require to access users' private data as that is why they reside inside the OSN. As internal components reside in the trusted OSN servers, users are more likely to allow the internal components to access users' private information. Here, we assume all internal modules are evaluated by the hosting OSN and their functional behaviors (such as what kind of data can be released to external components) are verified. As shown in Figure 2, the internal components can be further divided into four types of modules, based on the criteria shown in Table 3. In Table 3, internal modules are classified using two main criteria. The columns show that internal modules can be created/provided by either OSN provider or TPA developer, while the rows show two types of communications based on what kind of messages can be transmitted.

Module Type 1 (M1): OSN systems may offer their own functional modules to fulfill some tasks for applications, such as gadgets and widgets. This type of module is trusted as it is created by the OSN system, and thus most likely to be allowed to access users' private information. It can communicate with other application components through system calls, but any communication related to user data is disallowed.

Module Type 2 (M2): Just like $M1$, this type of module can receive and send system calls with the components on external servers, but is not allowed to transmit any customized messages to the components outside. The difference between $M2$ and $M1$ is that $M2$ is provided by third-party developers rather than the OSN system. This can be the case when an application needs an internal module that is not available by OSN but can communicate with external component using only system calls provided by OSN API.

The above two modules can be used to perform tasks in which users' private information is needed and communication between internal and external components can be done using system calls only. One motivating example is the notification sending scenario, where an external component triggers a request, and then an internal component can take over and fulfill the job and request job status information back to the external component without letting the external component know the information about the actual notification recipients.

Table 3: Module Types in Internal Components

	OSN provided	3rd-party provided
Communication w/ system calls	M1	M2
Communication w/ non-private data	M3	M4

Module Type 3 and 4 (M3, M4): In addition to system call, these two types of module can send privacy-nonsensitive information to the external parties. For examples, such modules can record users' keystrokes and mouse clicks during the game, and send them back to external server. While a sanitized form of users' private data could be considered as non-private data and belong in these types, sanitization issues are outside of this paper's scope.

We have identified four possible types of modules for the internal component. While application developers can choose any combination of them to construct the internal component of their application, it is also possible to see these types in a hierarchy as $M1$ and $M2$ could be considered more privacy preserving than $M3$ and $M4$ depending on the available system calls. For simplicity, we do not consider any ordering of the modules. The proposed social networking platform can mediate information flow in any overt channel between the components, according to user's policies. While we recognize there remains a covert channel problem, we deliberately keep this issue outside the scope of this paper which is focussed on overt communication.

Theoretically, while there could be modules that do not need to communicate with external components, we believe this type of internal modules are less realistic hence not discussed in this paper. Likewise, one can also consider internal modules that can overtly transmit users' private data to TPA server. However, this approach does not preserve user privacy, hence is not considered in this paper.

4. THE POLICY MODEL

Most OSNs offer access control policies for regulating user-to-user interactions, which are usually based on relationships between users in the social graph. Many researchers including ourselves also proposed a number of relationship-based access control [11] solutions that can improve the current approach from different aspects [5–7, 10]. At the same time, the prevailing approach of addressing application-to-user interactions in existing social networking platforms is rudimentary and course-grained to say the least. For example, in Facebook, users now can select what kind of information they are willing to share or not share with third party applications in general. But for a specific application, the only thing users can do is to approve all access the application requests during the installation stage, or otherwise deny the request and terminate the installation.

The most unique characteristic that distinguishes OSNs from other systems is that users and data objects in OSNs are interconnected through different types of relationships in the social graph. Installing and using an application can be also viewed as an establishment of a special relationship between user and application. Hence, it is intuitive to apply relationship-based access control to govern application-to-user interactions. Inspired by our previous work about user-to-user interactions [6, 7], we propose a policy model for controlling application's access in OSNs in terms of relationships.

4.1 Relationship-based Policies

Social graph depicts the relationships between users in OSNs, and it can be extended to include resources as well. Information sharing and social interactions among users are typically based on such graph. Social networking platforms would also benefit from the social graph by taking advantage of the social relationships to offer a richer experience for users. The platform would allow an application to traverse the social graph to access information owned by the application user's friends[1] through the application user. Thus, applications are able to reach users that have not installed the application at all.

In most existing OSNs, access control for user-to-user interactions is based on the topology of the social graph, where granting access permission is subject to the existence of a particular relationship or a sequence of relationships between the access requester and the target or the owner of the target, and the access control policies are specified in terms of such relationships. We can apply this paradigm of access control to application-to-user interactions by connecting applications to the social graph and considering application-to-user relationship for relationship-based access control.

Our framework comes with a policy model that allows users to specify how applications can access information owned by themselves and their friends in terms of the relationships between the application and the users. The available relationships for users to choose from include "install" relationship between the application and the users, in addition to all other user-to-user relationships among users in OSNs. Basically, users can use a combination of these relationships to govern application's access to resource.

As mentioned earlier, applications can access the friends' information of an application user without consent from the friends, and sometimes, beyond the application user's expectation as well. Effectively, users have delegated the rights to decide which applications can be trusted to their friends. To resolve the problem, some OSN sites now offer users the ability to select categories of information that can be disclosed to applications that they have not installed in case their friends use those applications. However, this mechanism is still far from user's expectation for fine-grained access control. To this end, the proposed framework considers policies of both the application users and their friends regarding the application's access. Although most current APIs only allow applications to access direct friends of their users, our policy language actually supports the extension to multiple hops of relationships for more expressive policies.

4.2 Policy Specification

We model an application's access request to user information as a tuple <*requester, action, target*>, where *requester* indicates the application that launches the access request, *action* denotes the type of access the requester wants to exercise, whereas *target* represents the target object of the access. A *target* can be either the user executing the application (i.e., application user), the application user's friend, or a data object owned by one of them. The set of supported actions is determined by the API of the social networking platform. The granularity of target objects is chosen based on the system designer's decision, possibly varying from data object to data type.

Access control policies for application-to-user interactions are composed of four elements:

<*action, target, (start, path rule),* $2^{ModuleType}$>,

where *action* specifies the type of access, and *target* is an optional parameter only for access against resource, denoting the resource to be accessed. It remains blank when the requested access is against a user.

For the third element, *start* is the position where access evaluation begins, which can be either the target user indicated in the request, the owner of the target object (denoted as *owner*), or the application requesting access (denoted as *requester*). A *path rule* is composed of one or more relationship paths, with each representing the required pattern of relationship between the involved parties in order to grant access. For example, $install$ represents the relationship between the application and the application user, "$install \cdot friend$" denotes the relationship between the application and the friend of the application user, whereas "$\emptyset$" indicates no explicit relationship path but the starting node itself is allowed. Given a pair *(start, path rule)*, if *start* is the target user or resourcs owner, the reference monitor checks if the requester (i.e., application) can be reached via relationship path specified in *path rule*. If *start* is the requester, the reference monitor then evaluates from the requester to see the relationship path between the requester and the target user or resource owner.

The last parameter $2^{ModuleType}$ indicates the set of application module types that is allowed to access, where $ModuleType$ is a set composed of *M1, M2, M3, M4* and *external*. It enables the resource owner to specify different policies for different application modules regarding the same pair of action and target. For example, user Alice may allow some internal modules of an application (see App 3b in Figure 2) she installed to access her birthday, but disapproves the external component of the application to access (see App 3a in Figure 2) the same data.

4.3 Examples

Below we show two examples to elaborate how users can use the policy model to control application's access in OSNs.

Example 1: App Request Notification. Once a user has installed and used an application, the application may start sending app requests to the user's friends or suggesting new applications to the user. The notification of such requests is annoying to many people. In general, users usually tend to hide their identities from applications they do not know. In this case, the following policies may help user Alice and her friends keep their privacy with respect to app requests:

- For applications she installed #1: $< app\ request,$ _, $(target\ user, install)$, $\{M1, M2, M3, M4, external\} >$
- For applications she installed #2: $< app\ request,$ _, $(requester, install \cdot friend)$, $\{M1, M2\} >$
- For applications her friends installed: $< app\ request,$ _, $(target\ user, friend \cdot install)$, $\{M1, M2\} >$

The first policy applies to requests such as $< AppX, app\ request, Alice >$, saying that applications she installed are allowed to send her app requests about activity updates and suggestions for new applications. The pair $(target\ user, install)$ expresses the applications that she installed. The second policy states that only the M1 or M2 modules of the applications she installed are allowed to send app requests to her friends. Here, $(requester, install \cdot friend)$ identifies users who are friends of the application user (i.e., Alice). The last one regulates whether applications her friends installed can send her notification of app requests. In this case, $(target\ user, friend \cdot install)$ identifies the applications installed by Alice's friends. The last two policies both deny the external com-

[1]Here, by "friends", we mean people who have any direct user-to-user relationship with the user, instead of people who specifically maintain a "friend" relationship with the user.

ponent of the applications to identify users or access users' information who did not have them installed. Instead, the M1 or M2 modules can fulfill the task of sending out such requests without letting the external part learn the identities of those recipients.

Example 2: Access User's Profile Information. Among the data that users put or generate in OSNs, some contains private information about users and should be safeguarded within the trusted OSN environment, while others may be appropriate for less trustworthy parties to access. For example, Bob prefers not to release his date of birth, relationship status, and email address to game applications he or his friends installed. On the other hand, the game application requires his age for customer survey, and normal key strokes and mouse moves during playing for proper functionality. To balance privacy and functionality, he expresses his preference through the following policies:

- Policy about Bob's date of birth: $< access, dateofbirth, (owner, install), \{M1, M2\} >$
- Policy about Bob's key strokes: $< access, keystrokes, (owner, install), \{external\} >$
- Policy about Bob's email address for applications his friends installed: $< access, emailaddress, (owner, friend{\cdot}install), \{M1, M2, M3, M4\} >$

The policy about Bob's date of birth allows the M1 and M2 modules of the application to access the data but denies any request from other internal modules and the external component. Key strokes in the game contains information that may need to be processed (possibly with significant computation) that is not likely to be done within OSN systems but more likely to be done at TPA, therefore is allowed to be accessed by the external component of the application. Similarly, the third policy indicates that only the M1-M4 modules of the applications that Bob's friends installed are allowed to access Bob's email address. This policy is used together with his friend's policy when an application his friend installed requests to access his email address. Since there is no overt flow of private information between the internal modules and external components in the proposed framework, email address that the internal component accesses is not going to be transmitted to the external component.[2]

5. CONCLUSIONS

We presented an access control framework for social networking platforms, preventing users' private information from leaking to external parties. Our design splits third-party applications into internal and external components, allowing the internal components to access private information but keeping it away from the external ones. We provided a simple policy model for application-to-user policies to regulate application's access. Users can specify different policies for different components of the same application, enabling more flexible and finer-grained control. Though the proposed approach does not eliminate privacy issues completely, it offers users greater controllability for their privacy against TPAs while allowing necessary features for the applications.

[2]While these policies could be used to control an application's access to user data, in real world system, the similar policies as a whole can be used to evaluate an application's usages on a user's private data at the time of the user's initial access request to try out the application. If the data usages of the application modules satisfy the user's policies, the user's access to the application could be allowed. Otherwise, OSN may deny the user's request or warn the users while showing which policies are violated by the application.

Acknowledgement

This research is partially supported by grants from the National Science Foundation and the State of Texas Emerging Technology Fund.

6. REFERENCES

[1] Facebook platform. `http://developers.facebook.com/`.

[2] Opensocial. `http://opensocial.org/`.

[3] P. Anthonysamy, A. Rashid, J. Walkerdine, P. Greenwood, and G. Larkou. Collaborative privacy management for third-party applications in online social networks. In *Proceedings of the 1st Workshop on Privacy and Security in Online Social Media*, 2012.

[4] A. Besmer, H. R. Lipford, M. Shehab, and G. Cheek. Social applications: exploring a more secure framework. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, 2009.

[5] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Trans. Inf. Syst. Secur.*, 13(1), 2009.

[6] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *Proceddings of the 4th IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, 2012.

[7] Y. Cheng, J. Park, and R. Sandhu. A user-to-user relationship-based access control model for online social networks. In *Proceedings of the 26th IFIP Annual WG 11.3 Conference on Data and Application Security and Privacy (DBSec '12)*, 2012.

[8] M. Egele, A. Moser, C. Kruegel, and E. Kirda. Pox: Protecting users from malicious facebook applications. *Computer Communications*, 35(12), 2012.

[9] A. Felt and D. Evans. Privacy protection for social networking apis. In *Proc. of Workshop on Web 2.0 Security and Privacy (W2SP '08)*, 2008.

[10] P. W. Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*, 2011.

[11] C. E. Gates. Access control requirements for web 2.0 security and privacy. In *Proc. of Workshop on Web 2.0 Security and Privacy (W2SP '07)*, 2007.

[12] M. M. Lucas and N. Borisov. Flybynight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, WPES '08, 2008.

[13] M. Shehab, A. Squicciarini, and G.-J. Ahn. Beyond user-to-user access control for online social networks. In L. Chen, M. Ryan, and G. Wang, editors, *Information and Communications Security*, volume 5308 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[14] K. Singh, S. Bhola, and W. Lee. xbook: redesigning privacy control in social networking platforms. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, 2009.

[15] B. Viswanath, E. Kiciman, and S. Saroiu. Keeping information safe from social networking apps. In *Proceedings of the 2012 ACM Workshop on online social networks*, WOSN '12, 2012.