Privacy Preserving Information Sharing in Modern and Emerging Platforms

Submitted in partial fulfillment of the requirements for
the degree of

Doctor of Philosophy
in
Department of Electrical and Computer Engineering

Yuan Tian

BASc Communication Engineering, Zhengzhou University
MS Computer Science, Beijing University of Posts and Telecommunications

Carnegie Mellon University
Pittsburgh, PA

May 2018

Acknowledgements

I would like to thank my advisor Professor Patrick Tague for the excellent mentorship, the research freedom, and the great encouragement given throughout my PhD career. I also give my thanks to all my committee members. Professor Lujo Bauer for giving me valuable advice for my research and career choice, Dr. Shuo Chen for all insightful discussions and the collaborations, Dr. Stuart Schechter for the suggestions and mentorship at Microsoft Research. Without you all, this thesis would definitely have not been possible!

My family, Huixian Tian and Hui Pang, thank you for all the love and support!

During my Ph.D. study, I got great support from my friends in Carnegie Mellon University and outside. Thank you very much for your friendship! I would like to thank my labmates Arjun Athreya, Eric Chen, Bruce DeBruhl, Jun Han, Madhu Harishankar, Yuseung Kim, Le T. Nguyen, Emmanuel Owusu, Brian Ricks, and Xiao Wang. I would like to thank my collaborators Amar Bhosale, Sungho Cho, Prof. Lorrie Faith Cranor, Weisi Dai, Xianheng Guo, Dr. Lin-Shung Huang, Prof. Collin Jackson, Prof. Limin Jia, Robert Kotcher, Wookjong Kwak, Dr. Yue-Hsun Lin, Dr. Bin Liu, Ying-Chuan Liu, Sanjay Parab, Yutong Pei, Joao Sa Sousa, Prof. Blase Ur, Dr. Helen Wang, Prof. XiaoFeng Wang, Prof. Carlee Joe-Wong, Nan Zhang, and Prof. Joy Zhang.

I would like to thank my friends during the graduate school: Aniruddha Basak, Youzhi Bao, Irina Brinkste, Akshay Chandrashekaran, Guan-Lin Chao, Chen Chen, Nicolas Christin, David Cohen, Dr. Willam Chan, Prof. Anupam Datta, Amit Datta, Samantha Goldstein, Chih Hu, Abhinav Jauhri, Wonkyum Lee, Bing Liu, Suyoun Kim, Prof. Michelle Mazurek, Dr. Piotr Mardziel, Prof. Ole Mengshoel, Prof. Bryan Parno, Shijia Pan, Prof. Vyas Sekar, Sari Smith, Stephanie Scott, Prof. John Shen, Nathan Snizaski, Michael Stroucken, Zheng Sun, Jennifer Wolfeld, Miao Yu, Ming Zeng, Hengzhi Zheng, Dr. Jun Zhao, Prof. Pei Zhang, Prof. Jia Zhang, Yingrui Zhang, and Dr. Zongwei Zhou.

To all my Facebook, Microsoft Research, and Samsung Research coworkers, mentors, and friends, thank you all! I would like to thank Dr. Cormac Herley, Dr. Peng Ning, Dr. Ioannis Papagiannis, Marjori Pomarole, Henry Corrigan-Gibbs, Chenguang Shen, Anhong Guo, Dr. Yan Michalevsky, Xiang Ren, Peng Gao, Prof. Huan Sun, Robert Xiao, and Yu Zhang.

The thesis research was supported in part by NSF CNS-1223477, 1223495, 1527141 and 1618493, and ARO W911NF1610127.

Abstract

Users share a large amount of information with modern platforms such as web platforms and social platforms for various services. However, they face the risk of information leakage because modern platforms still lack proper security policies. Existing security policies, such as permission systems and isolation, can help regulate information sharing. However, these policies have problems, such as coarse granularity, bad usability, and incompleteness, especially when new features are introduced. I investigate the security impacts of new features in web and mobile platforms and find design problems that lead to user information leakage. Based on these analyses, I propose design principles for permission systems that mediate how information should be shared in modern and emerging platforms, such as web and social platforms, to provide functionality with privacy preserved. I aim to design permission systems that only allow least-privilege information access. Specifically, I utilize program analysis and natural language processing to understand how applications use sensitive data and correlate these data with their functionality. With this understanding, I design schemes that ask for user consent about unexpected information access and automatically reduce overprivileged access. I provide guidelines for platform designers to build their permission systems according to respective adversary models and resources. In particular, I implement the new permission system for social platforms and Internet of Things (IoT) platforms that enable least-privilege information sharing. For the social platforms, I incorporate the primitives of Opaque handle, Opaque display, and User-driven access control (OOU) to design a least-privilege, user-friendly, developer-friendly, and feature-rich permission system.

According to my study on Facebook, OOU can be applied to remove or replace 81.2% of sensitive permission instances without affecting functionality. For IoT platforms, I present a new authorization framework, SmartAuth, that supports user-centric, semantic-based authorization. SmartAuth automatically collects security-relevant information from an IoT application's description, code, and annotations, and generates an authorization user interface to bridge the gap between the functionalities explained to the user and the operations the application actually performs.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern platforms such as web platforms and social platforms host a large amount of user data (users' locations, photos, posts, etc.). Third-party applications use the information to provide customized services and social experiences through the applications. Sharing the information to third-party applications without sacrificing privacy is always a challenging problem. For example, Facebook removed 30 permissions to access friends' data, such as their emails and photos, in 2015 over privacy concerns [1]. This change led to the death of many third-party applications that utilized this social information, such as Jobs with Friends and CareerSonar [2].

In this thesis, we first study the problems with the current design and implementation of sharing information with applications on modern platforms such as web platforms and social networking platforms. We identify problems with existing permission systems; for example, social networking platforms claim that they have to sacrifice functionality of third-party applications for privacy concerns [2], and a smarthome application that claims only to control the air conditioner in the application's description also has access to the smart locks. With knowledge of these problems, we provide design principles for least-privilege information sharing with applications. We consider least-privilege information sharing as sharing only what is needed for the functionality of applications. Our hypothesis is that the platform can work as a mediator to share sensitive information with one third-party application without sacrificing privacy. We approach the problem in the following steps: 1) analyze how user information is used and correlated with the functionality of the application; 2) stop sharing of the information that are not needed for the functionality automatically when possible, such as by using opaque display (only allow applications to display sensitive content without allowing access the content), according to the observed category of information usage; 3) communicate efficiently with the user about the unexpected data usage for the rest of the cases; and 4) examine trade-offs such as functionality and performance to optimize the design of the information-sharing framework. Specifically, we implement our solutions on social network platforms and smarthome platforms, including Facebook and Samsung SmartThings.

For the social platform, we analyze how social networking applications use information and we design schemes to only expose the information applications need for functionality. We run an analysis of the applications and categorize different information usage and choose solutions accordingly. For different categories, we show how we can use various techniques to remove the need for certain permissions. Based on our insights from the application analysis, we propose the following three primitives that can be used to remove permissions and share less information with applications. First, the platform can give an opaque handle referring a datum to an application, allowing the application to share or track the datum without learning the datum itself. For example, when a user tags a friend in a photo on a social networking platform, the platform generates a token (opaque handle) for the friend and shares the token with the application instead of the friend's real user ID. Second, the platform can provide an opaque display capability to allow an application to show a datum to the user without learning the datum itself. For example, a third-party application cannot get the profile photo and name of users' friends to show them to the users to enrich their social experiences inside the application because Facebook has removed permissions pertaining to friends' information; however, the application can display the friends' information in an isolated view without accessing sensitive content. Third, the two platforms can enable user-driven access control to allow a user's natural interaction with an application to implicitly grant the necessary permissions to support the user's activities. We evaluate our permission framework using the following metrics: 1) privacy gain—how much information users do not need to share with the applications in the new framework compared to previous permission systems; 2) developer effort—how difficult it is to implement the framework; 3) platform overhead—how much computation power the platform needs to support the scheme; and 4) coverage—what fraction of permission instances can be replaced by the framework.

For the smarthome platform, the overprivilege problem stems from the design of the frameworks. Different from platforms such as mobile and social networks, the privacy implication of the permission in smarthome is related to the context. For example, motion sensors on the door can detect people knocking on the door, and motion sensors on a medicine drawer can detect whether people pull out the drawer. What really matters here is the gap between the operations that the user thought the smarthome application would perform, which mostly comes from the applications' descriptions, and those that actually take place. We propose to bridge the gap by automatically collecting security-relevant information from a smarthome application's description, code, and annotations and generating a usable authorization user interface. To address the unique challenges in the smarthome application authorization, where states from a collection of devices are used to determine the operations that can happen on other devices, we come up with a set of new technologies that link the context of a device (e.g., a humidity sensor in a bathroom) to the semantics of an activity (e.g., taking a bath) reported by the applications' descriptions through natural language processing and program analysis. We would like to demonstrate that such policies can be enforced efficiently without interfering with the normal operations of existing smarthome applications.

Overall, this thesis will provide insights into designing a new information sharing/permission system for new platforms that can share user data with untrusted applications. We make two major contributions: 1) For the platforms we study, we design least-privilege information sharing systems that fit the needs of the applications and platforms. 2) For new platforms, we provide design frameworks and best practices for developers of new platforms. Developers can learn from our experiences to study what information applications need and design their platforms with the tradeoffs we discussed.

Chapter 2

Identifying and Understanding the Conflicts between Functionality and Privacy

When introducing new features in a platform or designing new platforms, developers run into the conflicts among functionality, security and privacy goals. In this chapter, we use the browser platform and the social platform as examples to study the conflicts between functionality and privacy in practice. On the one hand, we show that new features can introduce security violations. For example, the HTML5 screen-sharing API provides compelling functionality for benign applications, but malicious applications can utilize the feature to steal user data, transfer money from a user's bank account, or steal users' browsing history. On the other hand, platforms such as Facebook sometimes sacrifice functionality for user privacy [1]. We will talk about the details of these problems in the following sections.

2.1 Screen-Sharing Attacks Using a New HTML5 API

HTML5 has introduced many new concepts in the browser world. In particular, the new HTML5 screen-sharing API impacts the security implications of browsers tremendously. One of the core assumptions on which browser security is built is that websites cannot get the content displayed on a user's screen. However, screen sharing enables websites to see a user's screen. Consequently, websites will potentially be able to see all visible content from the user's screen, irrespective of its origin. This cross-origin content access, when combined with human vision limitations, can introduce new vulnerabilities because the foundation of browser security — same-origin policy can be bypassed easily. An attacker can capture sensitive information from a victim's screen using the new API without the consent of the victim. We investigate the security implications of the screen-sharing API and discuss how existing defenses against traditional web attacks fail during screen sharing. We show that several types of attacks are possible with the help of the screen-sharing API, namely cross-site request forgery, history sniffing, and information stealing. We discuss how popular websites such as Amazon and Wells Fargo can be attacked using this API and demonstrate the consequences of these attacks including economic loss, compromised accounts, and information disclosure.

2.1.1 Threat Model

In-Scope Threats. We are concerned with the security impact of the new screen-sharing API. We discuss attacks that can be launched when the user is using a screen sharing service hosted by a malicious website. The threat model for our study consists of a web attacker whose goal is to steal the user's sensitive information from other target sites and the browser and to affect the integrity of the user's account. To provide an outline of the threat model, we identify four roles involved and define their abilities below. Figure 2.1 displays the relationships and interactions among these four roles.

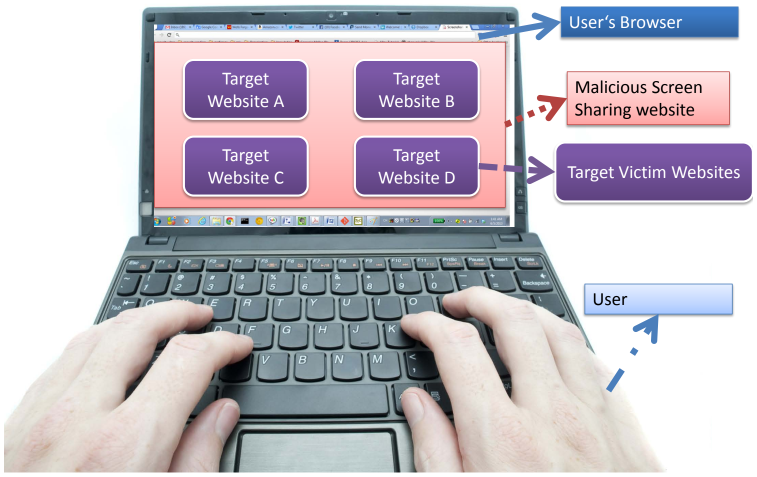


Figure 2.1: The threat model of attacks with screen-sharing API.

- **Web Attacker:** A web attacker is a malicious website owner who hosts a website with a valid SSL certificate and uses the screen-sharing API to see the user's screen. The attacker lures the user to visit the site and convinces the user to share the screen. Once the user starts using screen sharing on the malicious website, the web attacker displays sensitive information of the user on the screen and collects this information. Furthermore, the web attacker can utilize security credentials that they extracts from the pixels of the user's screen to launch more sophisticated attacks, such as the Cross-Site Request Forgery (CSRF) attack with the CSRF token.

- **User:** A user is a victim that visits a malicious website and uses the screen sharing services from the website. The website does not request the user to enter any sensitive data when sharing the screen. Also, the user does not open any browser tab or a window that reveal their personal data to the attacker. We assume that the user does not log out of target sites such as Gmail, Wells Fargo and Amazon. We argue the assumption is reasonable because most popular sites do not invalidate the user's session until the user logs out. Users might believe that the attacker's site cannot access their data on other sites since the browser's same-origin policy prevents cross-origin data access. Therefore, the user may not feel that it is necessary to log out of services before starting screen-sharing.

- **User's Browser:** We assume that the user's browser is patched for vulnerabilities that allow history sniffing and autocomplete history stealing reported in the past [3, 4, 5].

- **Target Sites:** Target sites contain users' sensitive information that an attacker attempts to steal. Our threat model assumes that target sites defend against cross-site scripting (XSS) and CSRF attacks. These sites use secret validation tokens to protect against CSRF attacks and these tokens cannot be stolen by traditional web attackers without screen sharing because they cannot access cross-origin page sources.

Out-of-Scope Threats. We do not consider the following threats.

- **Network Attackers.** We assume that network attackers cannot steal any sensitive information by acting as a Man-in-the-Middle (MitM) because all of the target sites operate over HTTPS.

- **Accidental Screen-sharing.** It is possible that users can be tricked to share their screens without knowing what is happening. However, in this paper, we only consider the case where people clearly know they are sharing their screens. We assume that people know what they input while their screens are being shared and that their screens will be shown to other parties. We do not consider information disclosure via actions such as users typing sensitive data on the screen while screen-sharing is on.

2.1.2 New Cross Origin Request Forgery Attack with the API

In the following section, we use a new CSRF attack as an example to explain the vulnerabilities of the screen-sharing API. The screen-sharing API breaks the same-origin policy, therefore, many defenses based on the same-origin policy no longer work. One dangerous example is the CSRF defense. In the CSRF attack, the attacker disrupts the integrity of the

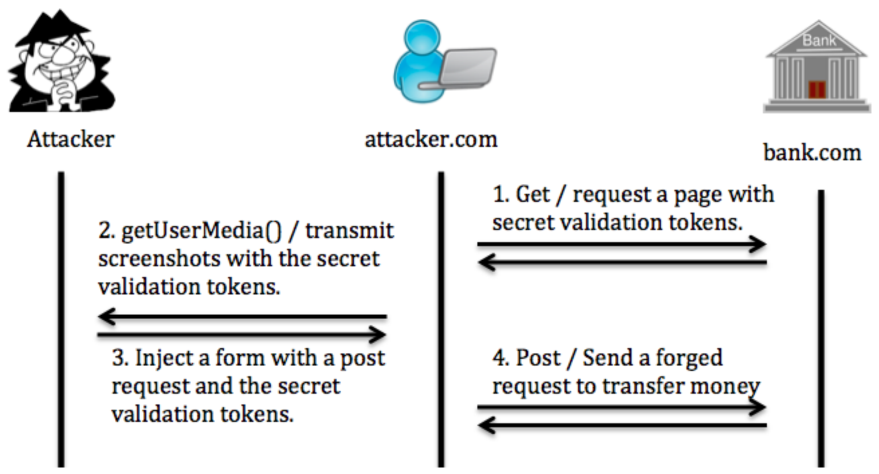

Attacker
attacker.com
bank.com
Bank
1. Get / request a page with secret validation tokens.
2. getUserMedia() / transmit screenshots with the secret validation tokens.
3. Inject a form with a post request and the secret validation tokens.
4. Post / Send a forged request to transfer money

Figure 2.2: Steps of a CSRF attack. 1. attacker.com requests a page with secret validation tokens from bank.com. 2. The screenshots with the source code and secret validation tokens are transmitted to the attacker's server via the screen-sharing API. 3. The attacker sends a form with a post request and the secret validation tokens to attacker.com to transfer money. 4. The post request is sent from the user's browser to bank.com and accepted by bank.com.

user's account state by forging a request with the user's credential. A commonly adopted defense against CSRF attacks relies on the trusted site to set secret validation tokens that are only known by the user's browser and are sent back with the request to authenticate the sender. However, this defense is vulnerable in a situation where the content of the target site is likely to be leaked to third parties [6]. Particularly, in the case of screen sharing, the user's secret validation tokens are accessible to the attacker hosting attacker.com with malicious screen sharing services (see Figure 2.2). When the user is fooled to start screen sharing, the attacker could catch the pixels of the user's screen and embed an iframe pointing to the target website. Besides displaying content from the target website, the attacker can even display the target website's code by the *view-source* links in the iframe. If the target site bank.com doesn't enable X-Frame-Options [7], the *view-source* link will expose the entire page source including the secret validation tokens to the attacker. Once the attacker obtains the secret token, the attacker can send forged requests from the user's browser. The trusted site accepts this request since it contains the expected authentication information: the HTTP cookie and the secret validation tokens.

Retrieving secret validation tokens from target sites is a crucial step in a CSRF attack. First, the attacker uses an iframe with the view-source attribute to expose the source code on the screen. The view-source attribute syntax is different between Google Chrome and Firefox. For Google Chrome, the code can be implemented as follows:

```
<iframe viewsource="viewsource"
    src="https://bank.com" ... />
```

For Firefox, the code has slight changes on the attribute:

```
<iframe src="view-source:https://bank.com" ... />
```

Next, the attacker can exploit relative positioning of CSS to locate the lines that contain the secret validation tokens inside the iframe. By using a negative value for the top attribute along with the position property, the attacker can scroll the iframe to a specific position. For example,

```
<div style="position:absolute;top:-2000px">
    <iframe style="width:800px;height:10000px"
        viewsource="viewsource"
        src="http://bank.com" />
</div>
```

With this technique, the attacker can collect the CSRF token or other security credentials from a size-limited window.

We use the code mentioned above to test multiple popular websites that adopt secrete validation tokens to defend against CSRF. Below, we describe one case study of Wells Fargo where we are able to break its defense mechanism through screen-sharing.

The Wells Fargo Case. Wells Fargo uses session arguments intensively on their websites.

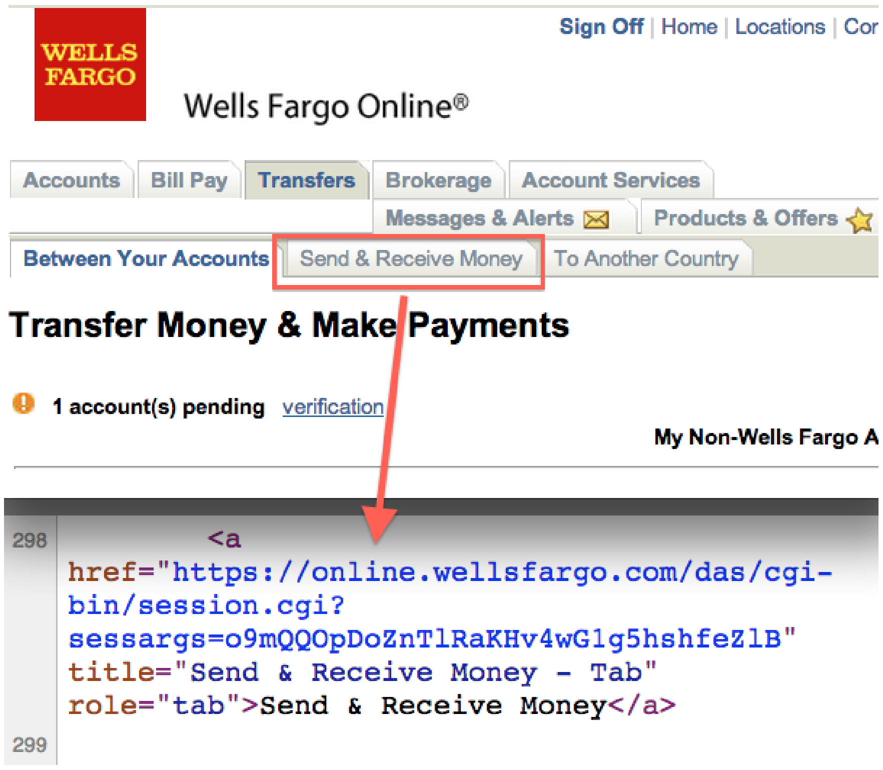
One usage is passing a session argument as a URL parameter in the page to send money. The mechanism prevent CSRF attacks in most cases; however, it will not work while the user is sharing the screen because the attack can retrieve the URL parameter from the source code. For example, in Figure 2.3 the source code of the “Transfer" page reveals the URL of “Send & Receive Money" which is supposed to be secured by the session argument. Since the Transfer page doesn’t enable X-FRAME-OPTIONS, the attacker is able to use iframe and view-source mentioned above to extract the URL.

When the attacker successfully lands on the “Send & Receive Money" page, the attacker is able to perform a variety of severe and persistent CSRF attacks because this page contains multiple critical session arguments and URLs that allow the attack to send forged requests. For example, in Figure 2.4, the source code exposes the session arguments and URLs for requests to update recipients, add recipients, and manage contacts. The attack can expose critical information to manipulate the user’s recipient and contact lists. Worse, attackers can transfer money to their own accounts because the page also provides a link to transfer money (see Figure 2.5).

2.2 Privacy and Functionality Conflicts of Data Sharing in Social Platform

Social platforms such as Facebook, Google+, Instagram, and Twitter, share user data with third-party applications to enable customized services. However, current best practices cannot provide a good balance between functionality and privacy. In the following sections, we study limitations of the social network permission systems, especially for conflicts between functionality and privacy.

Social platforms benefit highly from knowledge of user activities as well as the specific context around those activities, where context includes (but is not necessarily limited to) aspects

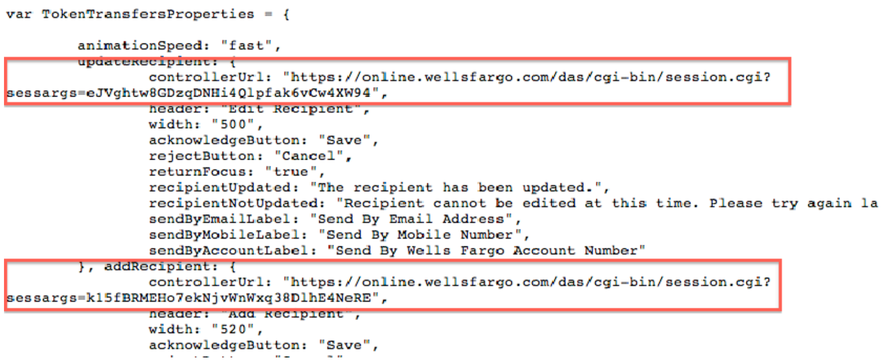


```
        <a
href="https://online.wellsfargo.com/das/cgi-
bin/session.cgi?
sessargs=o9mQQOpDoZnTlRaKHv4wG1g5hshfeZlB"
title="Send & Receive Money - Tab"
role="tab">Send & Receive Money</a>
```

Figure 2.3: Secret session arguments can be retrieved from the source code.

of *when* and *where* the activity takes place, *who* else is involved in the activity, and details of *how* the activity occurs. To demonstrate the importance of activity context, we provide several illustrative examples, together with context-free scenarios.

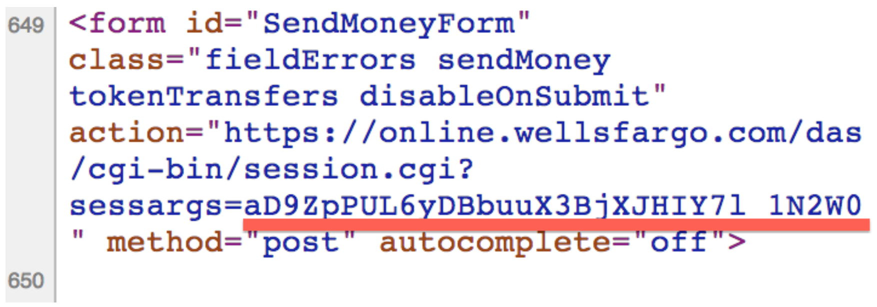
- "Alice watched a movie" versus "Alice watched the show of *Star Trek Beyond* with Bob at the AMC theater at 7:05 pm last Thursday".



```
var TokenTransfersProperties = {

        animationSpeed: "fast",
        updateRecipient: {
                controllerUrl: "https://online.wellsfargo.com/das/cgi-bin/session.cgi?
sessargs=eJVghtw8GDzqDNHi4Q1pfak6vCw4XW94",
                header: "Edit Recipient",
                width: "500",
                acknowledgeButton: "Save",
                rejectButton: "Cancel",
                returnFocus: "true",
                recipientUpdated: "The recipient has been updated.",
                recipientNotUpdated: "Recipient cannot be edited at this time. Please try again la
                sendByEmailLabel: "Send By Email Address",
                sendByMobileLabel: "Send By Mobile Number",
                sendByAccountLabel: "Send By Wells Fargo Account Number"
        }, addRecipient: {
                controllerUrl: "https://online.wellsfargo.com/das/cgi-bin/session.cgi?
sessargs=k15fBRMEHo7ekNjvWnWxq38DlhE4NeRE",
                header: "Add Recipient",
                width: "520",
                acknowledgeButton: "Save",
```

Figure 2.4: Multiple session arguments are exposed in the source code.



```
<form id="SendMoneyForm"
class="fieldErrors sendMoney
tokenTransfers disableOnSubmit"
action="https://online.wellsfargo.com/das
/cgi-bin/session.cgi?
sessargs=aD9ZpPUL6yDBbuuX3BjXJHIY7l_1N2W0
" method="post" autocomplete="off">
```

Figure 2.5: The form of a post request to send money.

- “Carol shopped online” versus “Carol *liked* the Backcountry.com retailer that was suggested by a Facebook advertisement, after she clicked on the link and spent four minutes browsing the company’s online store”.

As each of the examples illustrates, these scenarios include not just private information about what the user did, but also user behavior and contextual details across multiple platforms, apps, and websites. The context may have immense value to the various parties involved, especially business intelligence value for the companies and improved social relevance for the users. In fact, this is a large part of what has recently propelled social networking platforms like Facebook, LinkedIn, and others to success as advertising platforms [8].

Along with the value of sharing user activities and contextual information, there is a significant privacy concern about sharing the fine-grained personal behavioral information across apps and users. Hence, there is an important issue of privacy versus utility in such scenarios.

Currently, social platforms such as Facebook, Instagram, Twitter, and LinkedIn use permissions for third-party applications built on their platform APIs for accessing personal profile, activity, friendship status, and other data. The existing permission systems are very similar to those of modern client OSes such as Android and iOS. However, designing permission systems for social platforms poses additional challenges compared to these modern client OSes. The role of permission systems for modern client OSes is to let the user of a device grant permissions to applications that access sensitive resources on (and local to) the device, such as cameras, location information, and address book. In comparison, user context in social platforms is accumulated from different applications' contributions as well as from other users' contributions (such as a status update from the user's friend concerning the user). Therefore, permission system designers need to be concerned about (1) whether an application's contribution can be accessed by another application and (2) whether a user's contribution should be accessible to another user's application.

In order to understand current permission use in social platforms, we perform a set of user surveys and a study of how both the permission models and applications' permission usage. From our study, we identify several shortcomings of current permission systems in social platforms.

- Many permissions do not satisfy the least-privilege principle and they do not match users' expectations.
- The application review process to approve permission usage incurs significant delay for developers [9], affecting app availability and revenue generation.
- Platforms have even sacrificed functionality due to privacy concerns, specifically including examples of allowing users to share their friends' information, ultimately leading to an end-of-life for several applications (e.g., Jobs with Friends, Adzuna Connect, and CareerSonar) [1]. In our 18-month study of Facebook apps, we found that permissions themselves have evolved, been deprecated, and have become subject to new review criteria.

2.2.1 Study Methodology

To understand whether current permission systems can enable context sharing without privacy concerns, we investigate the permission systems in several popular social platforms, especially for Facebook. Our approach to studying existing social platform permission systems and get a deeper understanding of how they work includes the following tasks.

First, we perform an in-house study about the permission documentation and version history. Our study includes documentation of the social platforms Facebook, Google+, Instagram, Twitter, and LinkedIn. Moreover, since several of these platforms have changed their permissions and APIs over time, we also compare the historic versions to understand the trends. We read the permission and API documents to see what context information is available and how to retrieve the information from these platforms. We pay special attention to change logs and announcements to gain insight into platform designers' motivations.

Second, we perform a survey to better understand how users feel about permission systems in existing social platforms and to gauge their mental model of how such systems work. Our survey was performed using Amazon Mechanical Turk[1] (MTurk, for short), and we recruited a total of 300 Facebook users, identified by an average age of 29.7 years old (age range is 19-67) with a gender breakdown of 41% female, 58% male, 1% unknown. 79% of the participants use Facebook for more than 3 years, 16% of the participants use it for 1-3 years, and only 4% of the participants use it for less than 12 months. In our survey, we aske the user's perspective about how the context information is shared in social platforms and how do they manage the context sharing. Since Facebook is the most popular platform, we design the questions based on Facebook, and tests whether current permissions match users' mental models, and whether they will utilize permission management methods provided by Facebook (opt out permissions when loging in or revoking permissions). More interestingly, we also used a variety of third-party apps to gain first-hand experiences with the different systems from the user's perspective. The survey details are in Appendix A.1.2.

Third, we take the role of a developer by creating and submitting apps and participating in the app review process to better understand the app approval criteria. We submit two music recommendation apps four times with different sets of permissions. To test how effective the review is, we also use the permission data differently in each app.

[1]`https://www.mturk.com/mturk/welcome`

2.2.2 Permission Systems in Social Platforms

Before diving into the identified limitations, we first introduce the current permission systems briefly. An application asks a user's approval of permissions the first time a user logs in to their account on a social platform. Instagram and Twitter list all the permissions requested on one screen and request users to accept all permissions in order to sign in. For example, a third-party application for Instagram asks for many permissions such as access to comments, relationships, and likes; while the user has to approve all permissions to install the application. Google+ and Facebook try to give users more control over permissions. For example, Google+ introduces a feature called incremental permissions [10]. A Google+ application can request no permissions at sign-in, and later ask for permissions when a user accesses corresponding functionality. For example, a document management application only asks for the basic profile permission when a user signs in, and then asks the user about Google Drive permission right before the user starts to manage the documents. Once a user approves an incremental permission, the application gets the privilege forever. Similarly, Facebook also enables users to only choose some permissions to grant when signing in [11]. Moreover, Facebook displays more sensitive "extended" permissions on a separate screen to draw more attention from users. For example, in a music application, users will see the `publish_actions` permission displayed on a separate screen, and they can choose to skip the permission. Later, the music application can ask the user once again, and the user can choose whether to approve it. Note that the application can access resources with the permission in the future once the permission is granted.

Permission systems for social platforms contain permissions for reading and writing contextual information, and the permission numbers are usually not small. For example, Twitter provides data access permissions to access users' profile, tweets, friends, followers, and direct messages and action permissions to modify users' profile and post tweets on the users' behalf [12]. Facebook has even more permissions. Facebook has 47 permissions in its version

2.2 for applications to access users' information [11]. An application requests these permissions when a user connects his Facebook account to the application. Facebook divides these permissions into the following five categories. (1) Basic permissions: these include three permissions that enable access to users' public profiles, users' emails and users' friends who already installed the same app. (2) Extended profile properties: these permissions are all sensitive properties of a person's public profile. (3) Extended permissions: these include the most sensitive pieces of profile information. All extended permissions appear on a separate screen in the login interface so a user can decide whether to approve these or not. (4) Open graph permissions: these permissions are for gaining access to open graph data stored in users' profiles. (5) Page permission: this permission enables applications to administer any Facebook pages that a user manages.

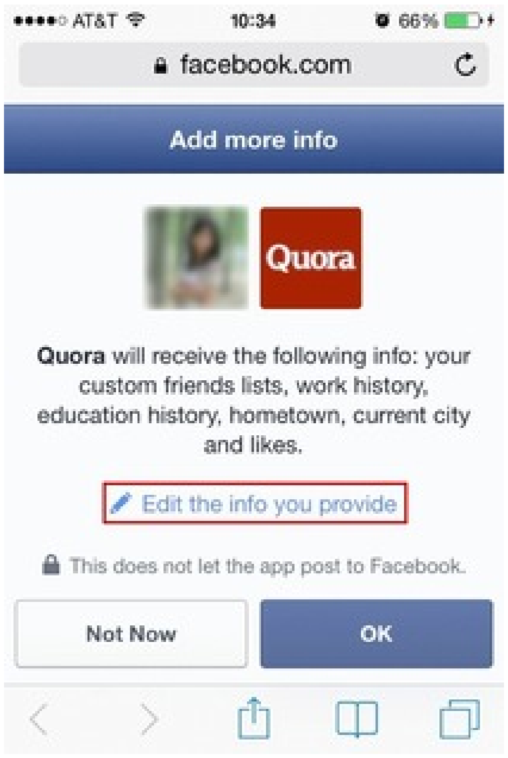
To audit permission usage in applications, Facebook has employed an application review process since permission version 2.0. Among all permissions, three basic permissions could be used without the review process. For the other 36 permissions, the applications need to go through the review process to submit documents about permission usage and provide test accounts. The review process differs depending on the number of permissions asked and the sensitivity of requested permissions [13]. For Instagram, developers can only use one specific permission without review and need to go through the review process for the other five permissions [14].

2.2.3 Limitations of Permission Models for Social Platforms

We discuss the limitations of current social platform models in this section. As we observe from the overview in Section 2.2.2, current social platforms primarily rely on manifest-based permissions for sharing context information, meaning that the user is forced to accept all of the requested permissions or abandon the app. This model is often referred to as the "all or nothing" permission model. A few social platforms, including Google+, use on-demand user prompts for a small subset of permissions. However, both of these approaches have fundamental limitations that make it extremely difficult to effectively share context while preserving privacy, forcing the platform to make a trade-off between privacy and utility.

Lack of Least-Privilege Context Sharing One primary limitation of both manifest-based and prompt-based permission systems is that the permissions are coarse-grained and share more information than what is needed by the application. Facebook is a representative example of a social platform that does not follow the least-privilege principle. Of the 47 permissions used by the Facebook application platform, 29 allow persistent access to user content once granted, as detailed in Table A.1. For example, once a user approves the `user_posts` permission, the app can access all future posts of the user, regardless of the context. In addition, once an app (and its developer) has access to a user's activity and context data, they can use it on- or off-line for any purpose at their leisure, regardless of what was disclosed in their application documentation and approval form. To understand how Facebook's review works, we submitted our own apps to Facebook for review, and we submit apps with similar descriptions and interfaces but have different usages of the permissions in the app's code. We do not observe any differences in Facebook's approval process although the apps' behaviors are different. For example, developers can claim that they only use sensitive information such as username, email, and company affiliations to create app profiles, when in reality they can freely share this information with ad libraries or for malicious purposes.

Our intuition is that persistent access with a single approval likely does not match many users' expectations. To test this idea, we run a survey on Amazon Mechanical Turk (MTurk) to collect real users' sentiments about content sharing in social networks. We display Facebook permission pages for third-party applications such as the one shown in Figure 2.6 and ask their understanding and attitude towards the permissions. We also ask about their experiences of using third-party applications on Facebook to get deeper insights. The survey details are in Section A.1.2 of the appendix. Out of the 300 participants who completed our


●●●●○ AT&T 10:34 66%

facebook.com

Add more info

Quora will receive the following info: your custom friends lists, work history, education history, hometown, current city and likes.

Quora
Edit the info you provide
This does not let the app post to Facebook.
Not Now
OK

Figure 2.6: One of the Facebook permission pages we show in our survey.

MTurk survey:

- 40% (120 out of 300) do not expect an application to access their future content using permissions granted in the past,
- 43% (129 out of 300) do not expect an application to share future content using permissions granted in the past,
- 61% (183 out of 300) are uncomfortable with the application accessing future content, and
- 73% (219 out of 300) are uncomfortable with the application posting future content.

While these results do not necessarily show an overwhelming displeasure with the current permission model, they demonstrate that there is significant room for improvement toward giving users what they expect.

Burdens on Developers and Users As a part of the Facebook application review process described previously [13], developers are required to put extra time and effort into documenting various aspects of data access and sharing, and additional time is spent on the review process itself. Of the 47 permissions currently allowed by the Facebook platform [2], 44 need to be manually reviewed and approved [13], costing a developer between three and 14 business days before their application is published. Additionally, users need to review several permissions when they sign into a third-party application. Since many of the permission decisions do not apply directly when the user signs in, the user is forced to make good privacy decisions out of context, which is unnecessary mental burden. The latest versions of the Facebook and Google+ platforms provide a mechanism for users to configure the permissions to be granted to an application [15, 16]. While this is good for user privacy because users can opt in or out of information sharing, it requires extra effort and deeper understanding of the permissions and their usage. Out of the 300 participants who completed our MTurk survey, only 53% (159) said they would spend the time to configure the permissions. Again, this demonstrates a gap between expectation and reality.

Sacrificing Functionality for Privacy Another important limitation for the Facebook permission system is that the current scheme does not have a good balance between functionality and privacy. We observe that many social platforms decided to make design decisions that sacrifice functionality of context sharing due to privacy concerns [1]. Currently, in order to enable context-aware services, social platforms treat contextual information like other data under the current privacy system. As the contextual information can be very sensitive, these platforms either support very limited sensitive context sharing or don't support the functionality at all. As a representative example, version 1.0 of Facebook's permission system allowed users to accept a permission that would share their friends' context data (using a total of 30 permissions), but these permissions were later removed in versions 2.0+ in favor of privacy [1]. These restrictions manifest in two ways: sharing across applications

[2]The research was done in 2016, and the latest version of Facebook permission is v2.6.

and sharing across users.

Blocking Information Sharing across Applications: Facebook enforces very strict requirements for sharing context across applications, making it almost impossible to utilize context of other applications. Despite context sensitivity, sharing context across applications can be very useful and even enable new services. On the side of privacy, blocking cross-application sharing prevents profiling agents or malicious parties from tracking user activities. However, user activities in one app can provide precious utility to another app. To exemplify the privacy-utility conflict, consider a wedding registry app that keeps a record of which gifts have already been purchased, so guests will not bring duplicate gifts. From the user's perspective, every shopper would want to know what has already been purchased, regardless of which store it was purchased from. However, competing stores may not want each other to know about their customers' respective activities. Facebook claims that the `read_stream` permission could allow an app to read the posts in a user's news feed, so we assumed that apps could access content from other apps in this way. However, the permission is only granted to apps building a Facebook-branded client [11]. In our study, we didn't find any apps granted this permission since Facebook started reviewing apps in permission model version 2.0+. They planed to deprecate this permission when version 2.3 expires in August 2017 [17]. Similarly, Instagram, Google+, and LinkedIn do not support sharing context information across applications. Though these social platforms enable applications to post on behalf of the user to their platform, they don't allow the applications to read other unrelated posts. Therefore, these applications cannot benefit much from the contextual information of the social platforms.

Blocking Information Sharing across Users: Facebook also has deprecated permissions for sharing friends' context to applications for privacy reasons. Sharing friends' context can improve the social experience in the applications, but this sharing creates issues with consent. Facebook deprecated 30 permissions about friends' context in versions above 2.0, including `friends_checkin` (access places where users' friends checkin on Facebook), `friends_likes` (acess items friends likes on Facebook), and `friends_photos` (access friends' photos on Facebook) because of privacy concerns [17]. Removing these permissions can block many useful scenarios. For example, a shopping app can display a list of friends with upcoming birthdays and a wishlist according to the pages that the friend liked and other information available on the friend's profile. With Facebook version 1.0, the shopping app could ask for `friends_birthday` (access friends' birthdays) and `friends_likes` to achieve this useful functionality to help find gifts for their friends. However, the user's friends might feel uncomfortable sharing their birthday and likes with the shopping application. Therefore, Facebook has removed these `friends_*` permissions to protect user privacy, also preventing functional uses of friends' context such as the above example. Many apps such as Job Fusion[3] had to remove important features or even shut down when they could no longer get such information about users. For example, an app cannot access a user's friend's employer, so it cannot match and refer job opportunities within the same company. Unsurprisingly, developers have filed complaints about losing such functionality and expressed their opinions that they had legitimate reasons to get access to such information [18, 19].

2.3 Related Work

In this section, we compare our work with previous work on CSRF attacks and permission systems for social platforms.

2.3.1 CSRF Attacks and Defenses

Previous studies show various techniques of stealing the CSRF tokens. Vela [20] demonstrates a heavy-load CSS-only attribute reader by using attribute selectors. However, it is not practical to read CSRF tokens with high entropy within a short period of time. Heiderich

[3]`https://jobfusion.co/blog`

et al. propose another CSS attack by using features such as web-fonts based on SVG and WOFF, CSS-based animations, and the CSS content property to extract CSRF tokens [21]. However, their attack focuses on CSRF token protected links, so the attack will not work in a scenario in which the CSRF token is not attached to the links. In contrast, the attack using the screen-sharing API can extract any CSRF token, irrespective of whether the X-Frame-Options are set or not by the target page.

There have been multiple proposals for CSRF defenses. SOMA [22] and App Isolation [23] provide CSRF defenses by defining valid entry points for the website. This can protect against the CSRF attacks using the screen-sharing API, but it is not feasible to whitelist all the entry points. Moreover, the web relies heavily on interlinks, so these solutions are not widely adopted. Gazelle [24] and Tahoma [25] provide cookie isolation between different apps, which also protect them from CSRF attacks. However, the strict isolation has some usability issues. Researchers [6] investigate current CSRF defense methods such as CSRF token, Referer header validation, and custom header and propose an approach of checking the origin of the request. According to their study, the CSRF token, which is the most popular defense, is reliable if well implemented. However, we find that the CSRF token defense does not work during screen sharing because the attacker can read the CSRF token directly. Mao et al. propose a defense by inferring if a request reflects a user's intention [26]. To judge the intention, they suggest checking the request's referer and all webpages hosted in ancestor frames of the referer. However, referer information can be manipulated by the attacker and sending referer information also raises privacy concerns.

2.3.2 Limitations of Permission Systems on Client OSes

Permission systems for modern client OSes such as Android and iOS have been studied extensively. Researchers have examined the privacy of using manifest and prompts for sharing locally stored resources. On the one hand, research shows that many users do not read or understand manifest permissions [27, 28, 29]. Further study also find that Android applications have historically requested more permissions than they need [30, 31]. In addition, researchers point out that the current permission system on Android lacks information such as frequency of accessing sensitive information [32]. On the other hand, studies find that users get bored or annoyed by prompts and tend to ignore them [33, 34, 35]. In contrast to these previous results on client OSes, we care not only about local resources on users' devices, but also about sensitive context accumulated on the social platforms from different applications and different users.

2.3.3 Limitations of Permission Systems in Social Platforms

Users have high expectation for privacy on social platforms [36, 37], but current permission models fail to match their expectations [38, 39]. However, these previous papers usually focus on information revelation in users' profiles or the usability of the privacy settings. For example, Gross et al. investigate the information users provide in their profiles in Facebook [40].

Instead, we study a few examples to understand the conflict between the functionality and privacy in current permission systems on social platforms. Previously, only papers about ad hoc case studies of the conflict exist. For example, Wisniewski et al. study the conflict stirred by using the tagging feature and privacy [41].

Chapter 3

Designing and Building Secure and Privacy-Preserving Systems

In Chapter 2, we analyze the conflicts of privacy and functionality in current platforms. With the understanding of the problems and challenges, we propose design principles for sharing information so that both privacy and functionality requirements are satisfied. We implement special designs for social platforms and smarthome platforms.

3.1 Privacy Preserving Context Sharing for Social Networks

Context sharing in social platforms enables context-aware services for third-party applications, providing great utility and value. However, context should be considered carefully due to its privacy-sensitive nature. In fact, protecting user privacy in social platforms is more challenging than that in client operating systems such as Android and iOS because the context is accumulated from different applications and users, not just from the local device resources. In Section 2.2.3, we investigate current permission models for Facebook to identify limitations of context sharing. Learning from these findings, we incorporate the primitives of opaque handle, opaque display, and user-driven access control (the OOU primitives, in short) to design a least-privilege, user-friendly, developer-friendly, and feature-rich permission system. We present our study of context sharing in Facebook applications and our findings show that OOU can be applied to remove or replace 82.64% (200 out of 242) of permission instances without affecting functionality. We further demonstrate the application of OOU by designing a permission system for the next-generation social platform, Somex, that can enable compelling context-aware scenarios with very limited information sharing.

3.1.1 Introduction

Learning from our surveys and several cases observed during our study in Section 2.2.3, we identify five desirable goals for a permission system in a modern social network that involves detailed activity and context sharing. First, users should be minimally burdened by the permission process, ideally by minimizing the number of permissions that must be approved by users. Second, users should be minimally interrupted by prompts to approve permissions on the fly to further reduce user burden. Third, applications should be limited by a permission system that satisfies the least-privilege principle. Fourth, the social network platform should ensure that data access and sharing match users' expectations. Fifth, applications should be able to provide full functionality without being inhibited by permission restrictions, enabling cross-user and cross-application sharing without compromising on other goals.

When studying social platforms, we observe that applications often do not need to have full access to the sensitive context. For example, applications just display the information or link to the information. For example, a contact app may simply need to display the friends' contact information of a user. Therefore, platforms could operate in a more privacy-preserving way that does not affect application utility by making the activity and context information opaque to the app. With a small amount of effort, the platform can mediate the information sharing between the user and the app, instead of blindly giving the information to the app. For example, the platform can provide a service so that the contact app embeds a display interface such as an iframe to display friends' contact information, without allowing the app to access the contact information.

Motivated by the insights, we advocate for the inclusion of the OOU primitives to design permission systems for activity and context sharing in social platforms. The platform can provide the application with an **opaque handle**, a pointer bound to the data without giving the data itself to the application. The platform can similarly provide an **opaque display** capability to allow an application to show a piece of data to the user without handing over the data itself. The platform can enable **user-driven access control** to allow the user's natural interaction with an application to implicitly grant necessary permissions to support the user's activities [42]. We believe that the appropriate combination of these OOU primitives can lead to improvement in social network usability and privacy, without sacrificing functionality (and at times even while providing additional functionality).

To evaluate our hypothesis, we study popular applications on the Facebook social platform. We conducted a study of Facebook applications over an 18-month period to observe trends in permission usage and to further evaluate the potential values of OOU primitives. From surveying 60 Android applications that use Facebook login out of the top 300 applications, we found that we can eliminate 200 of 242 permission instances (82.64%).

In fact, the OOU primitives can even enable more compelling context-based features. Building on the outcomes of our Facebook study, we design and implement a permission system for the emerging Somex activity-based social platform [43], based on the OOU primitives. In Somex, an activity is an abstraction that is used to group or tag items from different applications. An activity is an ongoing effort that a user keeps returning to, such as a project, trip planning, or event planning. An activity can involve multiple partners, allowing collaborative activity tagging of shared items. As a result, third-party applications for Somex provide more superior user experience when they are activity-aware, as the activity context enables more personalized services. We demonstrate that Somex requires only one manifest permission and no user prompts to enable more pervasive and frequent sharing of activity and context information.

In summary, the contributions of this project are as follows.

- We identify a key principle for designing permission systems of social platform that is least-privilege, user-friendly, developer-friendly, and feature-rich. The principle is to use three primitives: opaque handle, opaque display, and user-driven access control.
- We evaluate the efficacy of the OOU primitives by analyzing Facebook permissions in existing popular applications and find that the OOU primitives can help remove or replace over 80% of sensitive permission instances.
- We design and implement a permission system from scratch for the emerging social platform Somex using the OOU primitives. Our full-fledged implementation of the permission system enables many more feature-rich scenarios comparing to current social platforms.

3.1.2 System Model

As illustrated in Figure 3.1, a social network platform maintains users' context across a wide array of users, their friends, and the applications that they use. The platform then supports a variety of ways that applications can leverage this context information and feed additional application information back into the platform. Moreover, the platform may allow users to discover each other or provide services to each other based on common context. The platform is solely responsible for managing, protecting, and sharing activity and context information among users, applications, and services. Meanwhile, applications are isolated from each other, but they are allowed to embed user interfaces and content from other applications, as long as the platform supports such interaction and protects embedded content from leaking across the application boundaries.

Current social platforms share a wide variety of user-specific information including users' activities, interests, and application usage. By analyzing trends in social network platform capabilities, we expect the depth and breadth of information collected and shared to increase, especially in relation to contextual information around user and application activities. Specifically, new activity-based social platforms such as Somex [43] rely heavily on the current and historical context of users' activities to provide richer functionality and user experiences, as illustrated in Figure 3.2. Storage and sharing of users' sensitive context information in such platforms is thus a question of serious concern.

3.1.3 Threat Model

We consider attackers that are untrusted third-party applications on a social platform. Our work approaches are designed from the social platform's perspective, so we assume that the platform is trusted. Building trustworthy or decentralized social platforms is an orthogonal problem. We do not address misbehavior at the operating system level in this work, so we further assume that the OS is benign and not buggy. We consider the following potential threats: an application tries to access user context when the user does not grant access, and an application tries to get more context than it needs to function. In this work, we aim to reduce the context information that third-party applications can access while preserving their functionality or even providing more value to applications. In the permission model that we design, a social platform does not share context information with a third-party application when the data is not needed or can be made opaque for the functionality of the application. When the application needs to compute using the context information, the social platform shares only the information that is granted with user's authentic and usable authorization. We do not address the problem that applications leak user data via network or other schemes after they get the data. Mechanisms such as information flow control are helpful but are complementary. We consider the problem of identifying malicious applications to be out of our scope.

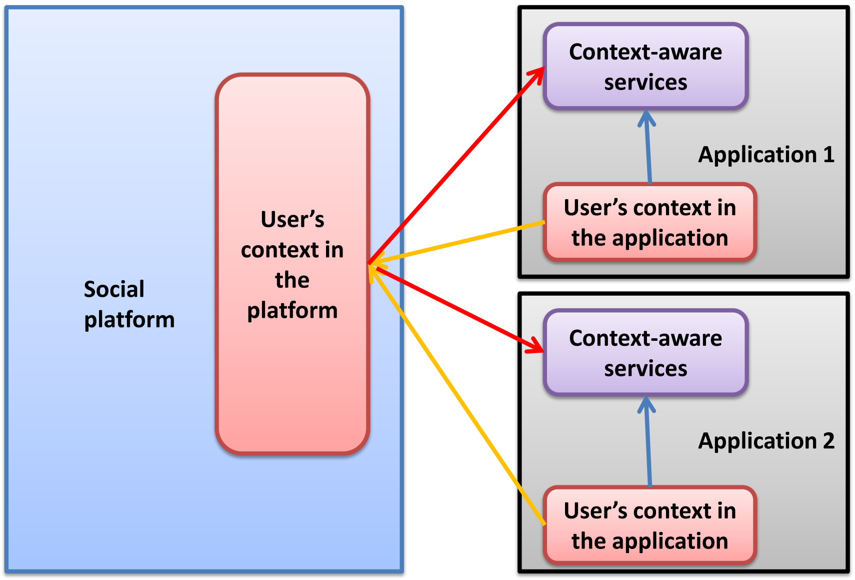

Social platform
User's context in the platform
Context-aware services
Application 1
User's context in the application
Context-aware services
Application 2
User's context in the application

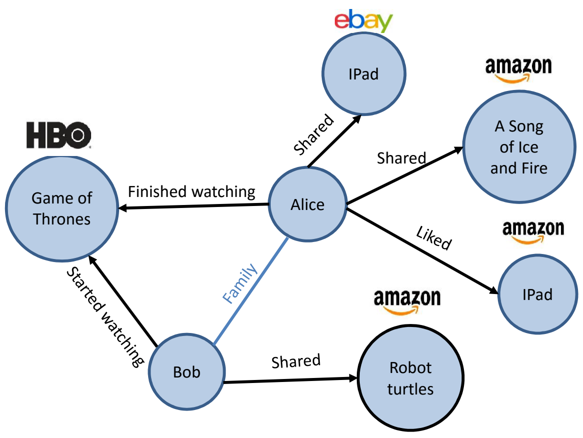
Figure 3.1: System model of the social platform and applications.

3.1.4 Design Goals

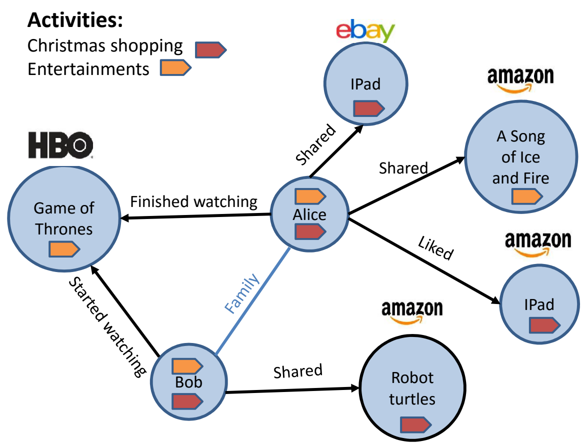
Based on our system model described above, our goal is to design usable permission systems that can enable rich functionality using context sharing while protecting user privacy. We note that the three properties of usability, rich functionality, and privacy preservation are often in conflict and require designers to choose one property over the other. However, our goal is to avoid trade-offs and provide all three properties to the greatest extent possible. As such, we aim to minimize users' burden in interacting with platform permissions and constrain applications' access to sensitive contextual data while allowing the platform to provide services around the sensitive data on behalf of the applications.

We approach the above-mentioned aim of providing usability, rich functionality, and privacy preservation in context-rich social network platforms through specific goals. In terms of usability, we have the following goals.

- **Minimize the number of permissions.** Fewer permissions can reduce the user's burden of inspecting the permission lists and the developer's burden to request and pass the platform's permission review process.



(a) Facebook's social graph is a representation of social context, and consists entities of (users and objects) and their relationships.



(b) Activity-based social platforms such as Somex augmented the social graph with activities to which existing entities belong.

Figure 3.2: Activity-based social platform share more sensitive context about user activities, which enables compelling features for third-party applications. For example, knowing the exact activity (whether the user is doing Christmas shopping or entertaining), Amazon can provide more accurate recommendations. Amazon will know whether to link the user to a previous session about "iPad" or a session about "A song of Ice and Fire".

- **Minimize the number of prompts.** The applications should present fewer prompts to users to further reduce mental effort of the user that participates in privacy protocols and permission actions.

In terms of privacy, we have the following goals.

- **Operate according to the least-privilege principle.** Applications should only have access to the information that they need for functionality. Moreover, applications should not obtain or share any information without the user's explicit or implicit consent. Finally, applications should not be allowed to gain persistent access to current and future information sources unless it is clear to users that consent is permanent. Cross-application and cross-user sharing should match the user's expectations whenever possible.

- **Eliminate privacy side effects.** Applications should not be able to infer social or contextual relationships from available data unless that is the expressed purpose of the data access. Moreover, applications should not be able to collude with each other to infer user activities across applications.

Finally, in regard to providing rich functionality without compromising on privacy or usability goals, we approach the following goals.

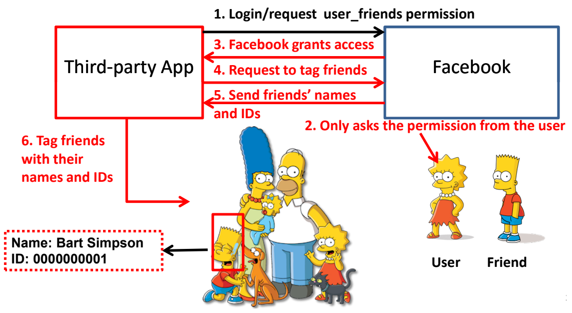
- **Support effective context retrieval.** Social platforms should support sharing context information to provide better functionality for services such as context-aware search or targeted ads. Knowing the context can enable applications to optimize results to be more accurate for the current activities. For example, user actions may have a different meaning or different importance depending on the activity context. Moreover, recommendations that utilize current activities' contextual information are more fine-grained, accurate, and valuable.

- **Support collaboration.** Users should be able to share experiences and knowledge to collaborate on similar tasks, and they should be able to view each other's activities on a shared project. Such context sharing across users can improve application value to users.

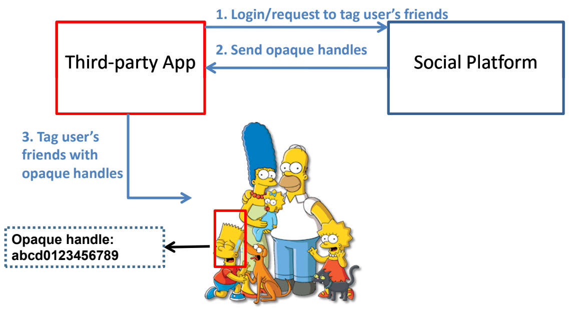
Ultimately, these goals combine to provide great value to users, application developers, and the platform. Specifically, users appreciate privacy protections and usable interactions, developers provide the utility necessary to gain reputation and loyalty from customers, and the platform gains attention and grows due to adoption by developers and users.

3.1.5 OOU Primitives

To achieve these goals for designing permission systems, we present three primitives that can be used to provide functionality, usability, and privacy for context sharing on social platforms. In what follows, we introduce the three data sharing primitives of opaque handle, opaque display, and user-driven access control (referred to collectively as the OOU primitives) and describe their application in social platforms.


1. Login/request user_friends permission
Third-party App
3. Facebook grants access
4. Request to tag friends
5. Send friends' names and IDs
Facebook
2. Only asks the permission from the user
6. Tag friends with their names and IDs
Name: Bart Simpson
ID: 0000000001
User
Friend

(a) In Facebook permission Version 1.0, app developers can request `user_friends` permission to tag friends of users. The third party app can get friends' names and unique Facebook IDs from this permission. After Facebook permission Version 2.0, third party app can get friends' names when tagging the friends.



(b) With opaque handle, the application can tag a friend of a user without knowing who that friend is.

Figure 3.3: We provide an example to illustrate the simple usage of the opaque handle primitive. In fact, the opaque handle can also point to more complicated data structure to provide even more utility. For more details, see Section 3.1.7.

Opaque Handle

An opaque handle is a reference to sensitive information that enables applications to provide functionality without getting the sensitive data directly. It is a random string that identifies a single piece of information, a single user for that piece of information, and the application request for the handle. To prevent colluding applications from tracking users, unique handles can be provided to different applications to reference the same piece of information.

In current social platforms, applications ask for permissions to access sensitive content, though they only need a reference linking to a specific user's content. In such a case, granting the applications access to a handle pointing to the user's content is sufficient.

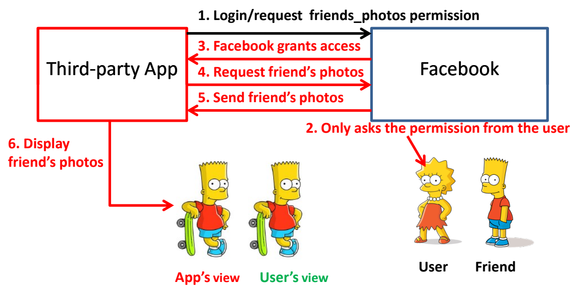
A potential case where opaque handles could reduce the number of permissions is to retrieve the IDs of a user's friends. For example, when a user tags one of her friends in a photo in Version 2.0+, Facebook generates a token (opaque handle) for the friend and shares the token with applications that have `user_friends` permissions. This approach is better for privacy than sending user id and name directly in Version 1.0. However, Facebook still doesn't achieve the privacy benefits of an opaque handle. Along with the token, Facebook also sends the friend's name to the applications, while in fact, applications get enough utility from the token and don't actually need to know the friend's name as well. By using the opaque handle, developers do not need permissions to get friends tagged in photos. If an application wants to also display the friend's name as part of the application, it needs to access the name to display it on the screen, but the technique described in the next section will address this.

The concept of an opaque handle can be very powerful. It is possible to attach an opaque handle to complex data structures or abstractions, far beyond simple data like a username. An opaque handle can enable very compelling context-based features. We will revisit the use of opaque handles for complex data in Section 3.1.7 for the Somex platform.

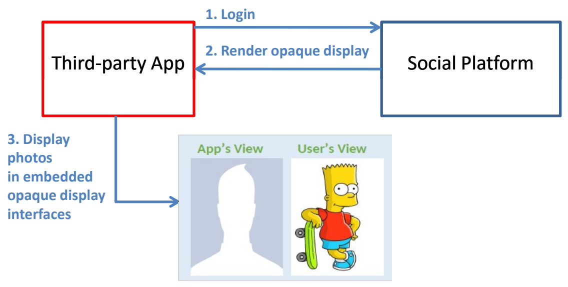
Opaque Display

An opaque display is a widget provided by the social platform that an application can use to display content to the user without exposing the content to the application. In many cases, applications ask for sensitive information from the user, but they only need to display this information in the interface. Thus, applications that are given access to this information are actually over-privileged. The opaque display can eliminate this unnecessary information sharing; instead of giving the information to the application, the social platform can provide an isolated interface that displays the information. The opaque display can be implemented with techniques similar to cross-domain iframes in a web browser [44] or similar to LayerCake [45] in Android to guarantee that the application can only display the content but cannot know the content being displayed. As illustrated in Figure 3.4, the application wants to display a user's profile photo in the application account of the user, and the opaque display widget allows it to do so without needing access to the profile photo.

With the opaque display, the social platform can remove undesired privilege from the appli-


1. Login/request friends_photos permission
3. Facebook grants access
4. Request friend's photos
5. Send friend's photos
Third-party App
Facebook
2. Only asks the permission from the user
6. Display friend's photos
App's view
User's view
User
Friend

(a) In Facebook permission Version 1.0, app developers can request `friends_photos` to access photos of friends without friends being aware. Facebook removed these permissions about friends' data in Version 2.0, therefore applications can no longer get photos from friends.



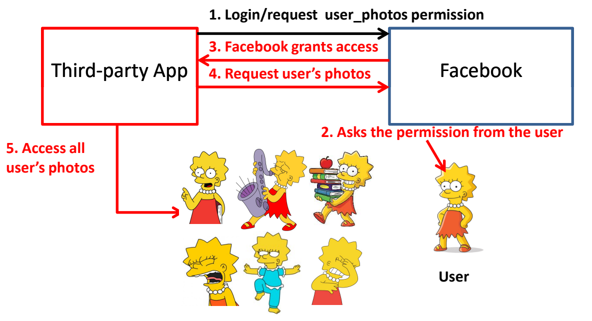
(b) With opaque display, the application can display the photos of friends without accessing these photos.

Figure 3.4: We provide an example to illustrate the use of the opaque display primitive.

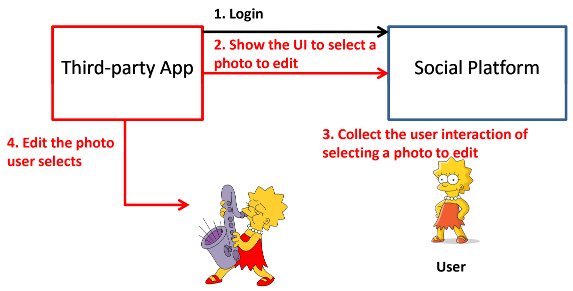
cations. As users don't have to approve the display permission, they also have less burden, thereby improving usability. Again, this can be achieved without impacting application functionality.

User-Driven Access Control

Roesner et al. proposed user-driven access control to get user approval implicitly [42]. Without additional permission requests, the application can embed the approval into existing user interactions. Consider a photo editing app that needs to access and modify a user's photos. Instead of using explicit permission requests which place more burden on users, the application can embed a user-driven access control widget, showing a button "Choose Photo", asking the user to share photos with the application. By clicking on the button, the user is implicitly approving the application's abilities to access photos, instead of directly asking the user for photo access permissions. User-driven access control removes unnecessary user interactions and is easier for users to understand and opt out (by not clicking the button). The social platform can get the user's authentic interaction from the embedded widget and infer user's intention to share the photo.


1. Login/request user_photos permission
Third-party App
3. Facebook grants access
4. Request user's photos
Facebook
2. Asks the permission from the user
5. Access all user's photos
User

(a) In Facebook, app developers can request `user_photos` to access all the photos of users. Users will not be notified again when the app is accessing these photos.


1. Login
Third-party App
2. Show the UI to select a photo to edit
Social Platform
3. Collect the user interaction of selecting a photo to edit
4. Edit the photo user selects
User

(b) With user-driven access control, the permission is granted by user's natural interaction with an application. For example, the user can explicitly click on the select button to share a photo about playing saxophone to share with a photo-editing app. The permission is built into the design of the functionality of selection in the applications. Instead of sharing all the photos and even all the photos in the future, the user can share the photo that she wants the app to access.

Figure 3.5: We provide an example to illustrate the use of the user-driven access control primitive.

Platform Design with the OOU Primitives

Social platforms can apply the OOU primitives of opaque handle, opaque display, and user-driven access control to reduce the number of permissions and user prompts, thereby reducing user burden (increasing usability) and developer burden (potentially increasing the availability and revenue).

We provide the following design process to outline how to design platforms that support the OOU primitives. First, the platform designer will identify useful scenarios of sharing context or other sensitive information. For each useful scenario, the platform designer can determine whether it is possible to share the data opaquely. If so, an API can be provided to developers to support this scenario using an opaque handle or opaque display. For example, the platform can provide an API to allow app developers to display the user's profile photo in the application, a feature that is useful (or even essential) for a game app that wants to display the user's profile photo in the high score board. If alternatively, the platform wants to allow computation over the data, for example, photo editing, the platform can provide an API for interactions that rely on user-driven access control, reducing the need for an explicit permission or prompt. In the example of photo editing, if a user takes action indicating that they want to edit a photo, it should be clear that they are implicitly granting permission for the application to access the photos to edit. If a scenario cannot support user-driven access control, the platform can fall back to a default of manifest permission or user prompt.

Observations

In our survey of existing social network applications, we did observe some experimental usage of the OOU primitives. However, the primitives are not widely deployed and are not used systematically. For example, as we discussed in Section 3.1.5, Facebook has started to use tokens to tag user's friends but doesn't get the privacy benefits of an opaque handle. Facebook also adopts opaque display in some scenarios such as the comment box [46], but doesn't use this primitive systematically. Facebook and Twitter also introduce user-driven access control for posting to user's timeline (applications can embed a Facebook "like" button and a Twitter tweet button to share information), but they still support applications to post to a user's timeline with the manifest permission. In addition, Facebook provides embedded "like" buttons for websites, which can get the user's information only if he or she clicks the "like" button. Similarly, Facebook provides user information to a page only if the user clicks to "like" the page. These implementations are ad-hoc and only cover a very small subset of permissions. Furthermore, when social platforms support these implementations, they usually also keep the traditional manifest permissions or prompts. To date, there has been no systematic study available about the extent to which these design primitives can help to improve the privacy and the functionality of social platforms.

3.1.6 Evaluation of OOU on Facebook

In this section, we present our evaluation of the OOU primitives on the popular Facebook social platform. We demonstrate that the OOU primitives can provide tangible improvements to the functionality, usability, and privacy for real social platforms. Since Facebook has adjusted their permission policies over time, we did a longitudinal study (October 2014 to March 2016) about real applications to see how the OOU primitives can help. Our investigation shows that the OOU primitives can account for most current permission usage: 200 out of 242 permission instances in our study. The OOU primitives can even restore functionality of permissions that have been deprecated due to privacy concerns. By applying OOU, platform designers can reduce the developer's efforts of going through the reviews, simplify user interactions, and protect user privacy.

We also modified an open source app [47] to show the effectiveness of the OOU primitives. To evaluate how helpful these OOU primitives are to app developers, we further surveyed developers to gauge their perspective about sensitive permissions and applying the three primitives.

Analysis Methodology

Since Facebook is one of the most popular social platforms, we analyze Facebook apps to evaluate how well OOU work. The first step is to download Facebook apps. Facebook marketplace is only a games portal (`https://www.facebook.com/appcenter` now redirects to `https://www.facebook.com/games/`), which does not provide a representative survey of applications using Facebook APIs because the applications in the game portal are all games. For diversity, we look at popular Google Play applications that use Facebook API. From the top 300 applications in Google Play, we chose 60 applications randomly from the applications that use the Facebook OAuth service and are not heavily obfuscated. We study these applications over an 18 month period from two perspectives: (1) analysis of decompiled source code to determine how permissions are used and (2) use of the app and inspection of resulting network traffic to further understand information usage. We repeat our analysis for current and previous versions of the applications. We use the `dex2jar`[1] and `JD`[2] tools to review the decompiled code manually, and use the `Charles` proxy[3] to analyze the traffic. Analysis of the permission usage is a non-trivial task, involving extensive manual analysis of the logic flow inside the applications to identify possible data use. We first decompile the code to inspect the level of obfuscations. Usually, if the app uses the obfuscation feature provided by ProGuard, the Java class names and function names are obfuscated to some extent (depends on the parameters the developers set). However, if apps are written with a different framework and just use Java as a wrapper, the core logic of the app is not obfuscated. Then we will work on these unobfuscated or lightly obfuscated apps in our study. Current Android apps usually have many files in the app, and it is very time-consuming to read line by line. Therefore, we first go through different files in the apps' codes to have a brief idea of the functionality implemented in the app. Then we focus on the files which are most relevant to the permissions to see how the data is requested and follow the logic to see where the data is used or whether the data is sent away to a remote server or a third-party service (for example, to an advertisement library). Sometimes the first estimation of the functionality might not be correct, and we didn't find reasonable things. Then we will study more files in details. We try our best to identify all the permissions in the app. On the other hand, we also use the app to understand its functionality and observe the traffic when we perform certain actions in the app. We use the traffic information to verify whether our analysis about sending data is correct.

[1] `https://sourceforge.net/projects/dex2jar/`
[2] `http://jd.benow.ca/`
[3] `https://www.charlesproxy.com/`

Eliminating Permissions Using OOU

We found that the three OOU primitives can help to remove permissions for Facebook applications. When the sensitive information is only referred to or displayed, we can replace the permission with the opaque handle, and opaque display. For example, Timehop[4] presents activities from a user's social networks from the same date in previous years. It requests many sensitive permissions such as `user_status` (read or edit a user's status on Facebook), `user_posts` (read a user's Facebook posts or post to a user's Facebook wall), `user_events` (read or edit a user's events on Facebook), and several more [11]. However, the app only displays the sensitive information, so the opaque handle and opaque display can step in to provide value. An interesting observation is that the opaque handle and opaque display can be extremely helpful in utilizing friends' data because most apps simply need to refer to or display the friends' information.

If the app needs to compute over the data, we cannot make the data opaque. When it is obvious that the application needs the permission for a certain feature, we can replace the permission by user-driven access control. We have seen many posting activities from the application and location-based recommendations that can leverage user-driven access control.

When apps need to compute over the sensitive data, and the usage of the data cannot link directly to a user interaction, we cannot apply the OOU primitives. These cases are relatively rare. For example, Hinge[5] uses `user_location`, `user_likes`, and `user_hometown` to recommend potential partners to date. Because of the high-level functionality, it is not clear to the user that this information will be exposed when they take action to find matches. Therefore, we cannot leverage user-driven access control.

In our investigation, we encountered instances of 230 out of the 242 permissions. Of these

[4]`https://play.google.com/store/apps/details?id=com.timehop`

[5]`http://hinge.co/`

Table 3.1: We tabulate and index the representative set of apps included in our study.

IndexApp		IndexApp	
1	com.gogobot	31	com.headspace
2	tunein.player	32	com.seatgeek
3	com.tappple.followersplus	33	com.birthdaycards
4	com.contextlogic.wish	34	com.picpal
5	com.yellowpages.android.ypmobile	35	com.podio
6	com.lucktastic.scratch	36	com.zedge
7	com.expedia.bookings	37	com.niketraining
8	com.skyscanner	38	com.sharethemeal
9	com.olx	39	com.feedly
10	com.tviplayer	40	com.dashradio
11	com.holyfood	41	com.younow
12	com.imdb.mobile	42	com.spotafriend
13	com.bleacherreport.android.teamstream	43	com.myfitnesspal.android
14	com.morecast	44	net.flixster.android
15	com.fox.now	45	com.myyearbook.m
16	com.espn	46	com.groupme.android
17	com.yidio.androidapp	47	com.webascender.callerid
18	com.timehop	48	com.clearchannel.iheartradio.controller
19	com.melodis.midomiMusicIdentifier	49	com.playrix.township
20	com.kickstarter	50	com.machinezone.gow
21	com.airbnb	51	com.glidetalk.glideapp
22	com.paxvoice	52	com.king.candycrushsaga
23	com.hinge	53	com.midasplayer.apps.diamonddiggersaga
24	com.papayamobile.kiwi	54	com.midasplayer.apps.bubblewitchsaga2
25	com.thescore	55	com.king.farmheroessaga
26	com.handmark.expressweather	56	com.spacegame.dessert
27	com.link	57	com.hotsuite
28	com.habitica	58	com.quora
29	com.shareity	59	com.groupon
30	com.anydo	60	com.droidhen.game.poker

230 permission instances, 169 permissions can be made opaque and 31 permissions can utilize user-driven access control, while only 12 permissions cannot be eliminated using OOU. We are unable to reach a conclusion about the remaining 30 permissions. There are 30 permissions that we did not see usages. It could be because the developer submit reviews to apply for the permission but does not use it actually, or it could be our investigations did not figure out the usage. Please find the table for the permission usage details in Table 3.2. We also provide the index information of the apps in Table 3.1.

Table 3.2: We analyze the Facebook permission usages in the 60 apps (242 permission instances), and find that 169 permission instances can be removed by opaque display or opaque handle, and 31 permission instances can be removed by user-driven access control.

App	Permissions	ODOH	UDAC	Cannot remove the permission with OOU	Cannot find the permission usage
1	public_profile, email, user_friends, user_birthday, user_location, user_hometown, publish_actions	email, public_profile, user_birthday, user_hometown	publish_action user_location		user_friends
2	public_profile, email	public_profile, email			
3	public_profile, user_friends	public_profile, user_friends			
4	public_profile, email, publish_actions	public_profile, email	publish_actions		
5	public_profile, email	public_profile, email			
6	public_profile, email	public_profile, email			
7	public_profile, email	public_profile, email			
8	public_profile, email	public_profile, email			
9	public_profile, email	public_profile, email			
10	public_profile, email	public_profile, email			

11	public_profile, email	public_profile, email			
12	public_profile, email	public_profile, email			
13	public_profile, email, publish_actions	public_profile, email	publish_actions		
14	public_profile, email, publish_actions	public_profile	publish_actions		email
15	public_profile, email, user_about_me, user_activities, user_birthday, user_education _history, user_hometown, user_interests, user_likes, user_location, user_relationships, user_relationship _details, user_religion_politics, user_website, user_work_history, user_photos, publish_actions	public_profile, email, user_location	publish_actions		user_about_me, user_activities, user_birthday, user_education _history, user_hometown, user_interests, user_likes, user_relation ships, user_relation ship_details, user_religion _politics, user_website, user_work _history, user_photos
16	public_profile, email, user_birthday	public_profile, email			user_birthday

17	public_profile, email, user_birthday, user_interests, user_likes, user_location, publish_actions	public_profile, email, user_birthday	publish_actions	user_likes, user_locaton, user_interests	
18	public_profile, email, user_birthday, user_location, user_events, user_photos, user_videos, user_status, user_posts, user_friends, publish_actions	public_profile, email, user_birthday, user_location, user_events, user_photos, user_videos, user_status, user_posts, user_friends	publish_actions		
19	public_profile, email, user_birthday, user_location	public_profile, email, user_birthday, user_location			
20	public_profile, email, user_friends	public_profile, email, user_friends			
21	public_profile, email, user_friends, user_activities, user_birthday, user_education _history, user_hometown, user_interests, user_likes, user_location	public_profile, email, user_friends, user_activities, user_birthday, user_education _history, user_hometown, user_interests, user_likes, user_location			

22	public_profile, email, user_friends, user_birthday, user_location	public_profile	user_friends		email, user_birthday, user_location
23	public_profile, email, user_friends, user_birthday, user_work_history, user_education _history, user_hometown, user_location, user_photos, user_likes	public_profile, email, user_photos, user_birthday, user_work_history, user_education _history, user_hometown, user_likes	user_location, user_friends		
24	public_profile, email, user_friends	public_profile, user_friends	email		
25	public_profile, email, user_likes	public_profile, email		user_likes	
26	public_profile, publiish_action	public_profile	publish_action		
27	public_profile, user_birthday, pub-lish_actions	public_profile, user_birthday	publish_actions		
28	public_profile, user_friends	public_profile		user_firends	
29	public_profile, email	public_profile, email			
30	public_profile, email	public_profile, email			

31	public_profile, email	public_profile, email			
32	public_profile, email, user_location, user_likes	public_profile, email		user_location, user_likes	
33	public_profile, email, user_friends, user_brithday, publish_actions	public_profile, email, user_friends, user_birthday	publish_actions		
34	public_profile, user_friends	public_profile, email			
35	public_profile, email, user_work_history, user_location, user_website, user_about_me	public_profile, user_work_history, user_location, user_website, user_about_me			
36	public_profile, email	public_profile, email			
37	public_profile, email, user_friends, user_birthday, user_location	public_profile, email, user_friends, user_location			user_birthday
38	public_profile, email, user_friends, publish_actions	public_profile, user_friends	publish_actions	email	
39	public_profile, publish_actions	public_profile	publish_actions		
40	public_profile, email	public_profile, email			

41	public_profile, user_friends, publish_actions	public_profile	publish_actions		
42	public_profile, email	public_profile, email			
43	public_profile, email, user_birthday, user_friends	public_profile, email, user_friends		user_birthday	
44	public_profile, email, publish_actions	public_profile, email	publish_actions		
45	public_profile, email, user_birthday	public_profile, user_birthday			email
46	public_profile, email	public_profile, email			
47	public_profile, user_friends, user_work_history, user_location, user_hometown, email	public_profile, email, user_friends, user_work_history		user_home town, user_location	
48	public_profile, email, user_birthday, pub-lish_actions	public_profile, email	publish_actions	public_profile	
49	public_profile, user_friends, publish_actions	public_profile, user_friends	publish_actions		
50	public_profile, email	public_profile, email			

51	public_profile, user_about_me, email, user_hometown, user_likes, user_birthday, user_friends	public_profile, email, user_about_me	user_friends		user_hometown, user_likes, user_birthday
52	public_profile, email, user_friends, publish_actions	public_profile, email	user_friends publish_actions		
53	public_profile, email, user_friends, publish_actions	public_profile, email	user_friends, publish_actions		
54	public_profile, email, user_friends, publish_actions	public_profile, email, user_friends	publish_actions		
55	public_profile, email, user_friends, publish_actions	public_profile, email, user_friends	publish_actions		
56	public_profile, email, user_friends, publish_actions	public_profile, email, user_friends	publish_actions		

57	public_profile, user_friends, user_status, email, user_about_me, user_events, user_likes, read_custom_ friendlists, user_photos, user_posts, user_videos, user_managed_ groups, read_page_mailboxes, read_insights, publish_action	public_profile, user_friends, user_status, email, user_about_me, user_events, user_likes, read_custom_ friendlists, user_photos, user_posts, user_videos, user_managed_ groups, read_page_mailboxes, read_insights	publish_actions		
58	public_profile, email, user_likes, user_location, user_work_history, user_education_ history, user_hometown	public_profile, email			user_likes, user_location, user_work_ history, user_education_ history, user_hometown
59	user_friends, email, public_profile	public_profile, email			user_friends
60	public_profile, user_friends	public_profile	user_friends		

Proof-of-Concept Implementation

To demonstrate the practicality of using the OOU primitives, we modify the code of a popular app UberSync from Google Play with the OOU primitives, and we demonstrate that we can provide the same functionality and even bring back functionality that was lost due to permission deprecation, without exposing users' and their friends' data. UberSync is a popular application for syncing Facebook contacts to a phone's contact list. The application is open source, and it has over one million downloads and positive ratings. After version 2.0, Facebook removed the permission `friends_photos` and only provides the names of friends that also install the application for the `friends_list` permission. The application can no longer get users' friends' information, such as profile photos and emails. The developers had to stop the project and made a disclaimer that they can only sync the contacts that also install the application or the user has to invite their friends that are not using the application. The permission changes caused a drastic drop in the ratings of the UberSync app due to loss of functionality, including many recent one-star reviews [47].

Since UberSync only displays friends' names and photos, we can employ the opaque display to achieve the same functionality without needing the permission. We update UberSync to utilize the opaque handle and opaque display to achieve the same functionality in version 1.0 of the Facebook permission system without allowing the application or developer (us in this case) to access the information. Specifically, we modify UberSync to use a token to represent each friend to fetch content about the friend and embed an isolated view that displays friends' names, photos, and contact information, as is shown in Figure 3.6[6]. The modification is lightweight, requiring modification of around 30 lines of code for UberSync to embed the opaque display and an additional 50 lines of code for the social platform to host the service. With the opaque handle and opaque display, UberSync recovers the functionality

[6]We cannot modify Facebook itself, so we built a clone app that hosts users' friends' information just like Facebook, and UberSync interacts with the clone app. If Facebook adpots the OOU primitves, it can get similar results.

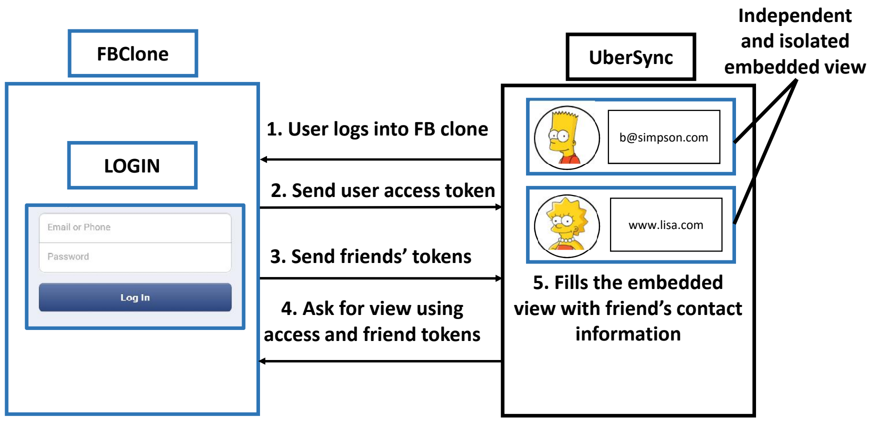

FBClone
LOGIN
Email or Phone
Password
Log In
UberSync
Independent and isolated embedded view
b@simpson.com
www.lisa.com
1. User logs into FB clone
2. Send user access token
3. Send friends' tokens
4. Ask for view using access and friend tokens
5. Fills the embedded view with friend's contact information

Figure 3.6: Friends' data access is not allowed after Facebook permission model 2.0, so UberSync lost the ability to sync friends' data. The opaque handle and opaque display allow UberSync to display friends' profile photos and contacts without sacrificing privacy. All UberSync learns is the number of friends, which is not considered as sensitive.

provided under version 1.0 of the Facebook permission system. Instead of getting all the sensitive data about friends, UberSync only gets the number of user friends, which is not considered as very sensitive and is granted to apps by Facebook by default.

3.1.7 Designing a Permission Model for the Somex Activity Platform

The OOU primitives not only enhance users' privacy but also enable new context sharing capabilities for next-generation activity-based social platforms. In this section, we describe our efforts toward designing a permission model for the Somex activity platform recently designed by Microsoft Research [43]. Somex hosts users' and collaborators' sensitive activities and shares the information with third-party applications for customized services. If Somex adopts the same permission system as Facebook, the platform will run into privacy issues of sharing collaborators' data. Therefore, we design the Somex permission system around the OOU primitives detailed in Section 3.1.5 to expose meaningful user activity and context information to third-party applications. The resulting Somex core API requires only one manifest permission. In what follows, we provide background on Somex, its requirements, and our permission model design.

Overview of Somex

Somex is an activity-based, context-aware social platform. All of the digital interactions between a Somex user and a device are organized based on the user's high-level *activities*. Examples of such activities include Christmas shopping, travel planning, or paper writing. Somex employs a mixture of techniques (including manual labeling and machine learning) to determine and infer, at any given time, the user's activity context. In addition, Somex is also able to provide good user experiences by associating each user's activity context with collaborators (e.g., shopping friends, travel partners, paper co-authors) [43]. The high-level visions of Somex overlap with those of several prominent digital assistant engines including Google Now[7] and Cortana[8]. We believe the interests and recent advancements in building activity-based, context-aware social platforms call for an in-depth study of the API design and the permission system required for these platforms.

Privacy Requirements of Somex

The permission system of an activity-based, context-aware social platform such as Somex has a fundamentally different set of privacy requirements than the permission systems of traditional social platforms like Facebook. We highlight the differences below.

Activity-Sensitive Trust Model In traditional social platforms, third-party access to certain user information is defined by a single Boolean variable (i.e., a user either grants or denies an app access to his/her news feed). However, for an activity-based, context-aware social platform, the trust between a user and a third-party application must be established on a per-activity basis. For example, a user may trust Amazon with information regarding her "Christmas Shopping" activity, but not her "Personal Finance" activity. In a normal

[7]`https://www.google.com/landing/now/`
[8]`http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana`

browsing session, the user may seamlessly switch between her two activities (e.g., shop for Christmas presents while checking her credit score). A well-designed permission model should minimize user burden, preserve the user's privacy goals, and provide sufficient functionality to the third-party application.

Time-Sensitive Trust Model Unlike traditional social platforms where user data is obtained through explicit sharing, activity-based social platforms such as Somex or Google Now possess a large amount of sensitive user information that is passively collected or inferred. Therefore, it is crucial that passively collected user information is not exposed to third-party applications without the user's consent. In traditional permission systems (e.g., for Facebook or Android), access to certain user information is granted for the lifetime of an application, unless explicitly revoked by the user. This type of permission scheme is undesirable for activity-based platforms because it does not give the user full control of the information being shared at any given time, as discussed earlier.

Overlapping Ownership of Digital Artifacts Suppose Alice and Bob are collaborating on a wedding registry. Alice grants Amazon access to her current activity on Somex and logs into Amazon to purchase an item. At this point, Somex detects that Bob already has the same item in his shopping cart at Macy's. Should this information be relayed to Amazon? Clearly, warning Alice about potential duplicate purchase has value, but Bob may not want to let Amazon know of his activity at Macy's. The collaborative aspect of activity-based social platforms raises an interesting research question: How can one enable secure information sharing for digital artifacts with multiple owners? In Section 3.1.7, we address this problem using opaque display.

The Somex API

We design the permission system and APIs for Somex with the OOU primitives, which provide rich context-based functionality and still meet the privacy requirements. With our

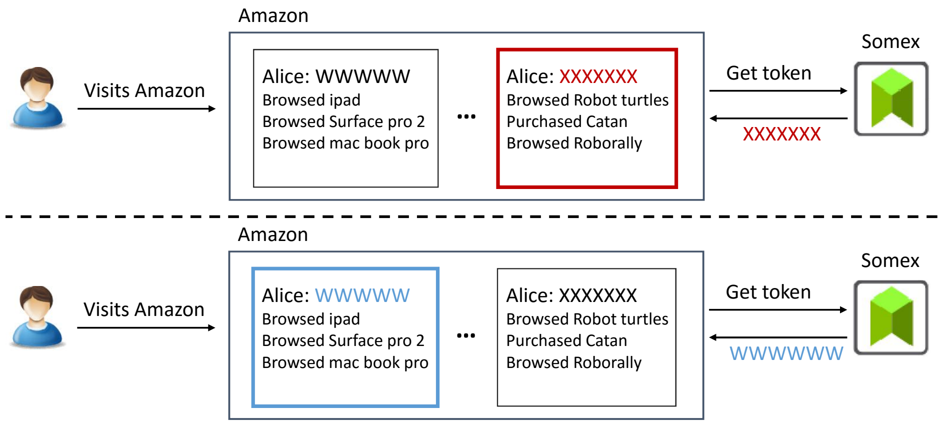


Figure 3.7: An activity token is an opaque handle that points to the current activity of a user. The activity token enables Somex to point to current activity without sacrificing privacy. Amazon can query the activity token to link the user to a previous browsing session.

design, Somex needs only need one permission, no prompts, and enables third-party applications to provide superior user experiences because applications can leverage rich cross-session context and cross-user context.

Somex organizes a user's digital artifacts and actions performed on these artifacts into activities. We first define the three concepts that will be used throughout the rest of this section: artifact, action, and activity.

Somex defines an *artifact* as a digital object or a piece of information that a user can interact with. An artifact can be a file, a web page, merchandise on an e-commerce site, an identifier, or a string. Similar to the concept defined in the Facebook Graph API [48], Somex designers view an *action* as a human-readable string (defined by a developer) that represents a user's logical interaction with a certain artifact (e.g., "creates" a file, "purchases" merchandise). Finally, Somex defines an *activity* as a label assigned to artifacts and actions that represents the underlying user context. Each artifact can be associated with multiple activities, but each action can only be associated with one activity. For example, merchandise from Amazon can belong to both the "Christmas Shopping" activity and the "Wedding Registry" activity while a user can "purchase" the merchandise for only one activity.

There are three meaningful services that Somex can offer to third-party applications:

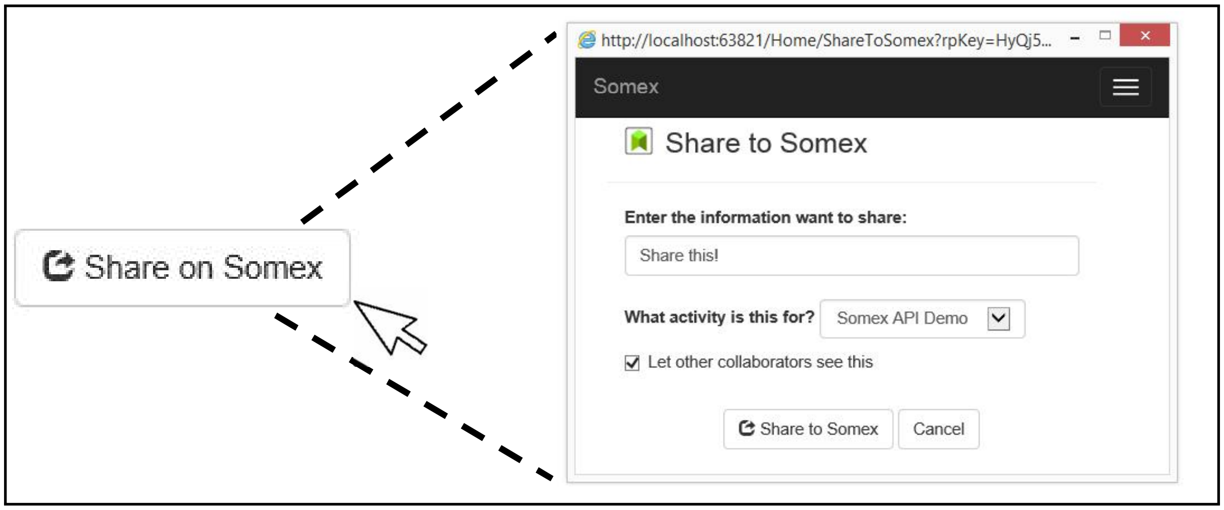
- provide contextual information about the user's current activity,
- record and share application-specific artifacts and actions for the current user,
- fetch relevant user artifacts and actions for the current activity or application.

We describe the API designed with OOU primitives to achieve each of the three services and the techniques used to minimize the number of permissions required.

GetCurrentActivityToken

The `GetCurrentActivityToken` function provides context information about the user's current activity to a third-party relying party. It achieves this without explicitly revealing any private data associated with the activity. This is done by using an *activity token*, an opaque handle that identifies the user's current activity context. One can imagine the activity token as a browser cookie that is associated with an activity instead of a browsing session. Each activity token is assigned on a per-application, per-user basis. That is, different relying parties would receive distinct activity tokens for the same activity, ensuring that two applications cannot collude to track a single user. Different collaborators of the same activity would also receive distinct activity tokens for the same activity, ensuring that the relying party cannot infer collaboration relationships between different users. Detailed attacks beyond this type of inference are beyond the scope of this work.

By itself, the activity token does not expose any user data. However, it can be used similarly as a browser cookie to link a user's current "activity session" with a previous session that the relying party has on record. Figure 3.7 illustrates an example of using the activity token to obtain the user's current activity context. In this example, the user has established previous browsing sessions with Amazon. For the first browsing session, Amazon obtained the activity token `WWWWW` from Somex and observed the user browsing laptops and tablets. For the second session, Amazon obtained the activity token `XXXXXXX` from Somex and observed


http://localhost:63821/Home/ShareToSomex?rpKey=HyQj5...

Share to Somex

Enter the information want to share: Share this!

What activity is this for? Somex API Demo

☑ Let other collaborators see this

Figure 3.8: Using user-driven access control to share context from an application to Somex. An app can call the Share_Action API to ask users to share context implicitly.

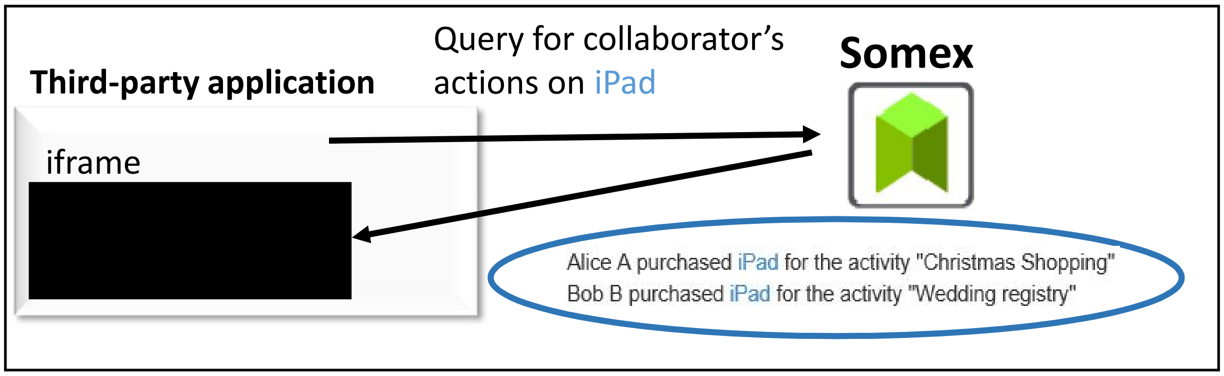
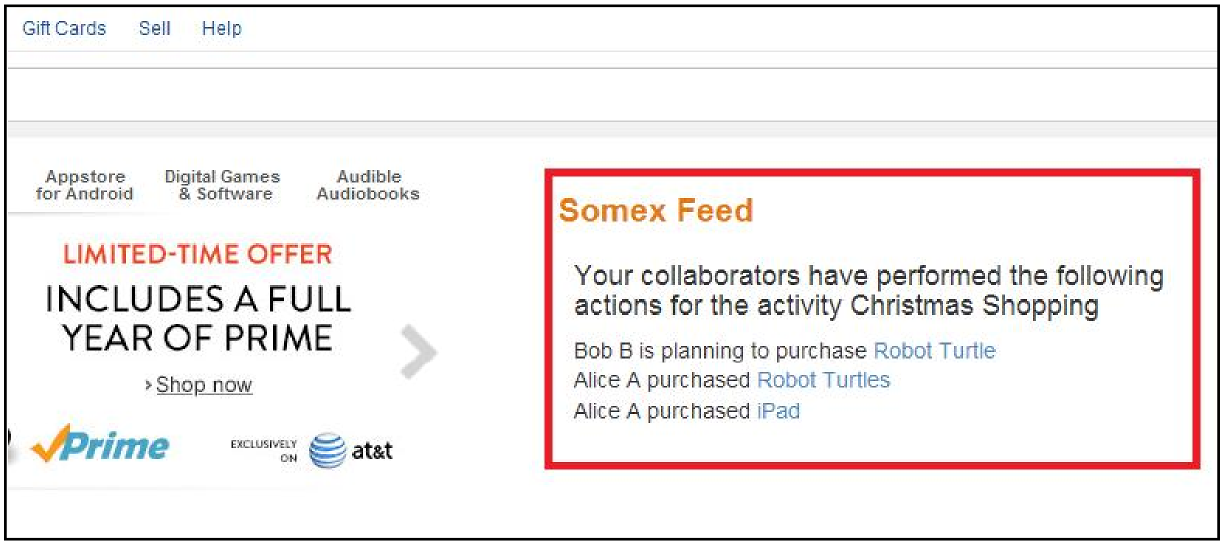

Third-party application
Query for collaborator's actions on iPad
Somex
iframe
Alice A purchased iPad for the activity "Christmas Shopping"
Bob B purchased iPad for the activity "Wedding registry"

Figure 3.9: Somex uses opaque display to display context in an application without exposing context to the application.

the user browsing children's toys. When the user visits Amazon once again with the activity token `XXXXXXX`, Amazon could look at its history to infer that the user's current browsing session is related to shopping for children's toys.

Share_Action The `share_action` function is designed for users to share an artifact to applications and users. It is implemented using the user-driven access control primitive. We illustrate in Figure 3.8 that users can share current context explicitly without a permission prompt.



Somex Feed

Your collaborators have performed the following actions for the activity Christmas Shopping

Bob B is planning to purchase Robot Turtle
Alice A purchased Robot Turtles
Alice A purchased iPad

Figure 3.10: Case study: collaborators shop together for Christmas. An app such as Amazon can call the `get_actions_of_activity` API to display the collaborators' actions such as Christmas shopping.

APIs to Fetch Artifacts and Actions We observe that applications in Somex do not perform computation on Somex-supplied data and the Somex feed is usually directly presented to the user, hence we use the opaque display to show relevant artifacts and actions. We use cross-domain iframes [44] in web and LayerCake [45] in Android to embed an opaque display interface, thus we can display activity feeds without violating privacy. For example, in Figure 3.9, Mike is visiting a shopping website and he wants to know his collaborators' behaviors on an iPad. The shopping website displays his collaborators' behaviors in a cross-domain iframe such as "Alice purchased an iPad while Christmas shopping". Mike can see the context while the shopping website cannot.

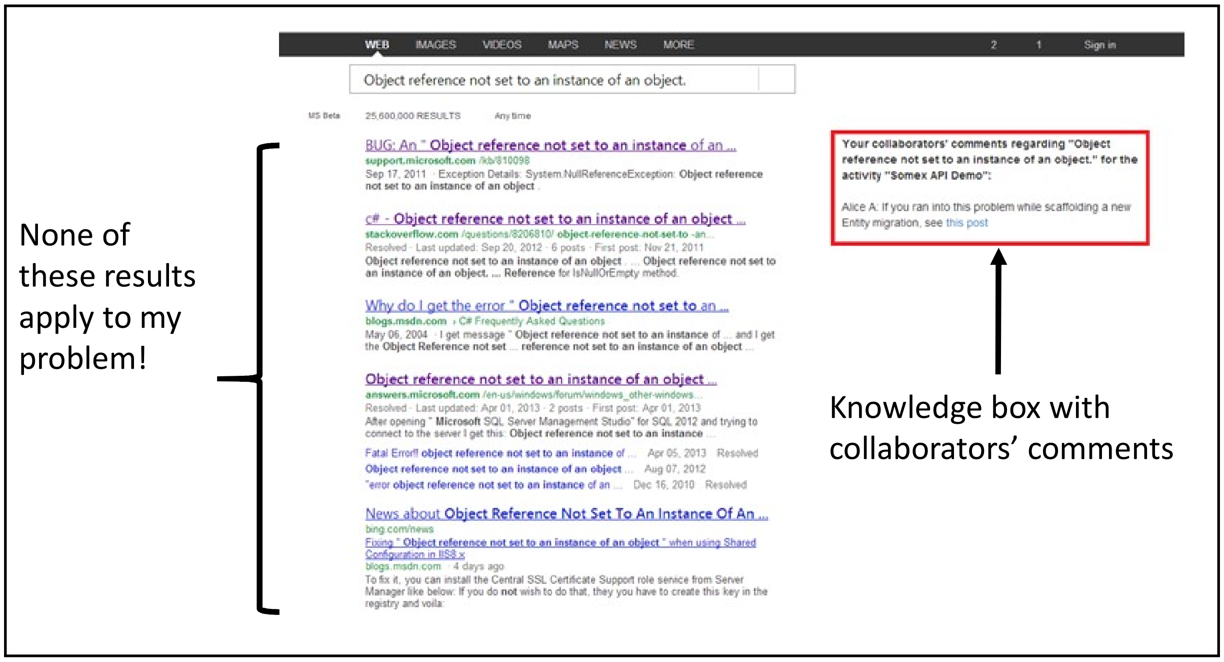
We design several APIs to fetch actions opaquely without requiring permissions. We use the `get_actions_of_activity` and `get_actions_on_objects_of_activity` APIs as examples and explain the apps.

The `get_actions_of_activity` API is designed to get all collaborators' actions performed on a given activity. In Figure 3.10, a third party application such as Amazon can query the actions with an actions token of one activity such as "Christmas shopping". Somex will send related actions such as “Adam bought a robot turtle while Christmas shopping”.

The `get_actions_on_objects_of_activity` API provides more specific query results than `get_actions_of_activity`. It is designed to fetch all collaborators' actions performed on this activity for a particular artifact. In Figure 3.11, Eric is searching for a solution to an error in his coding project. Instead of providing general search results, Google can combine collaborators' context to help Eric. Google queries Somex with an Object key (e.g., hash of the search query) and activity token, and Somex passes collaborators' actions to the embedded opaque display. Therefore, Eric can see his collaborators' comments about how to solve his problem.

Using Somex APIs As is shown in Figure 3.12, Somex shares the utility of the context information to third-party applications without disclosing the context. We demonstrate the scenario when two users are collaborating to purchase birthday gifts. The third-party applications can call the Somex APIs to benefit from the utility of rich context. For example, Amazon can leverage the cross-session context (users' previous actions with the same activity such as gift shopping can be used to improve user experience), cross-user context (collaborators' actions can be used to improve user experience), and cross-application context (users' actions from different applications can also be used to improve user experience). The cross-session context sharing can be achieved using an activity token. The cross-user context and cross-application context can be achieved using secure embedded user interface and activity token. The only permission that is being used is the manifest permission to get the activity token.

Overall, our permission model design for Somex relies on only one manifest permission, while allowing Somex to provide feature-rich services utilizing shared context across applications and users without sacrificing privacy.


WEB
IMAGES
VIDEOS
MAPS
NEWS
MORE
2
1
Sign in
None of these results apply to my problem!
Knowledge box with collaborators' comments

Object reference not set to an instance of an object.

MS Beta 25,600,000 RESULTS Any time

BUG: An " Object reference not set to an instance of an ...
support.microsoft.com /kb/810098
Sep 17, 2011 · Exception Details: System.NullReferenceException: Object reference not set to an instance of an object .

c# - Object reference not set to an instance of an object ...
stackoverflow.com /questions/8206810/ object-reference-not-set-to -an...
Resolved · Last updated: Sep 20, 2012 · 6 posts · First post: Nov 21, 2011
Object reference not set to an instance of an object . ... Object reference not set to an instance of an object. ... Reference for IsNullOrEmpty method.

Why do I get the error " Object reference not set to an ...
blogs.msdn.com › C# Frequently Asked Questions
May 06, 2004 · I get message " Object reference not set to an instance of ... and I get the Object Reference not set ... reference not set to an instance of an object ...

Object reference not set to an instance of an object ...
answers.microsoft.com /en-us/windows/forum/windows_other-windows...
Resolved · Last updated: Apr 01, 2013 · 2 posts · First post: Apr 01, 2013
After opening " Microsoft SQL Server Management Studio" for SQL 2012 and trying to connect to the server I get this: Object reference not set to an instance ...
Fatal Error! object reference not set to an instance of ... Apr 05, 2013 Resolved
Object reference not set to an instance of an object ... Aug 07, 2012
"error object reference not set to an instance of an ... Dec 16, 2010 Resolved

News about Object Reference Not Set To An Instance Of An ...
bing.com/news
Fixing " Object reference not set to an instance of an object " when using Shared Configuration in IIS8.x
blogs.msdn.com · 4 days ago
To fix it, you can install the Central SSL Certificate Support role service from Server Manager like below: If you do not wish to do that, they you have to create this key in the registry and voila:

Your collaborators' comments regarding "Object reference not set to an instance of an object." for the activity "Somex API Demo":

Alice A: If you ran into this problem while scaffolding a new Entity migration, see this post

Figure 3.11: Case study: collaborators share comments and experiences about a question. An application such as Google can call the `get_actions_on_objects_of_activity` API to display collaborators' actions for this activity performed on an artifact.

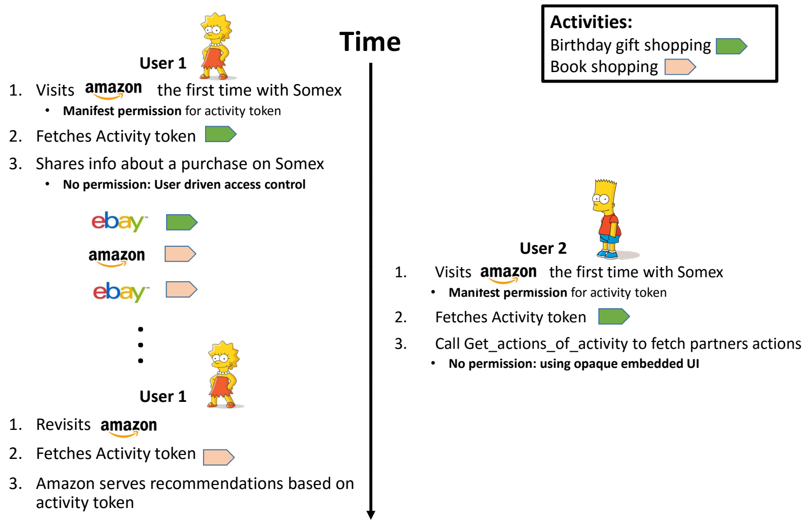

Time
Activities:
Birthday gift shopping
Book shopping
User 1
1. Visits amazon the first time with Somex
• Manifest permission for activity token
2. Fetches Activity token
3. Shares info about a purchase on Somex
• No permission: User driven access control
ebay
amazon
ebay
User 1
1. Revisits amazon
2. Fetches Activity token
3. Amazon serves recommendations based on activity token
User 2
1. Visits amazon the first time with Somex
• Manifest permission for activity token
2. Fetches Activity token
3. Call Get_actions_of_activity to fetch partners actions
• No permission: using opaque embedded UI

Figure 3.12: User 1 and user 2 collaborate on shopping for birthday gifts. With the APIs we designed, Somex enables applications to utilize the cross-session, cross-user, and cross-application context without sacrificing privacy. For example, users can see the context-aware recommendations and collaborators actions on the same gift shopping activity.

3.1.8 Discussion

We have shown that the three primitives can be very helpful in removing information sharing to third-party applications. In most cases, platform designers can make data opaque by using the first two primitives, where the platform don't share privacy information but just provide the utility for the third-party applications. When a piece of context needs to be seen by the functionality of the application, then opaque handle and opaque display will not be sufficient and a permission is needed. Even when a permission is needed, user-driven access control could kick in to make granting seamless and share less information. Although adopting user-driven access control means that the platform still needs to share user data with third-party applications, user studies show that granting permission implicitly matches users' expectations to share data [42]. Also, the user-driven access control is designed to ensure the user interface integrity, which enables reading authentic user interactions to infer user intention. However, it is possible that if the permission cannot be integrated with the application function (e.g., Somex Share button), user-driven access control also cannot be used. From our evaluation, such occurrence is very rare. Homomorphic encryption [49] and other privacy techniques [50] might be helpful to solve the problem when the applications need to compute over data and the permission cannot be made user-driven. However, even if the problem is solved, our work is still useful because we demonstrate the light-weight OOU primitives have practical value for permission system design.

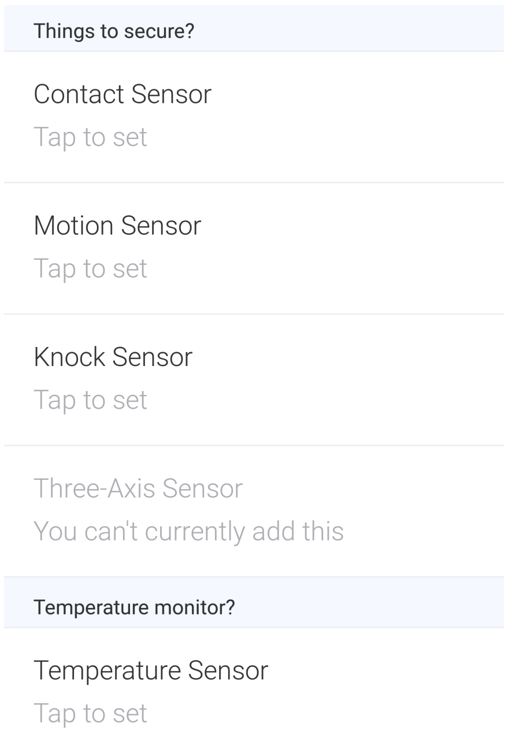
3.2 User-Centered Authorization for Smarthome Apps

After studying information sharing and permissions in social platforms, we also use similar techniques in other collaborative platforms such as Internet of Things. Internet of Things (IoT) platforms often require that users grant third-party apps permissions, such as the ability to control a lock. Unfortunately, because few users act based upon, or even comprehend, permission screens, malicious or careless apps can become overprivileged by requesting unneeded permissions. To meet the IoT's unique security demands, such as cross-device, context-based, and automatic operations, we present a new design that supports user-centric, semantic-based "smart" authorization. Our technique, called *SmartAuth*, automatically collects security-relevant information from a SmartApp's description, code, and annotations, and generates an authorization user interface to bridge the gap between the functionalities explained to the user and the operations the app actually performs. Through the interface, security policies can be generated and further enforced through an enhancement of the existing SmartThings platform [51]. To address the unique challenges in SmartApp authorization, where states of multiple devices are used to determine the operations that can happen on other devices, we devise new technologies that link a device's context (e.g., a humidity sensor in a bathroom) to an activity's semantics (e.g., taking a bath) using natural language processing and program analysis. We evaluate SmartAuth through user studies, finding participants who use SmartAuth are significantly more likely to avoid overprivileged apps.

3.2.1 Introduction

The rapid progress of Internet of Things (IoT) technologies has led to a new era of home automation, with numerous smart-home systems appearing on the market. Prominent examples include Samsung's SmartThings, Google's Weave and Brillo [52, 53] and Apple's HomeKit [54]. These systems use cloud frameworks to integrate numerous home IoT devices, ranging from sensors to large digital appliances, and enable complicated operations across devices (e.g., "turn on the air conditioner when the window is closed") to be performed by a set of applications. Such an application, called a *SmartApp* in Samsung SmartThings or generally an *IoT app*, is instantiated in the cloud. A user interface (UI) component on the user's smartphone enables monitoring and management. Like mobile apps, IoT apps are disseminated through app stores (e.g., the SmartThings Marketplace [55]), which accept third-party developers' apps to foster a home-automation ecosystem. Unlike mobile apps, IoT applications control potentially security-critical physical devices in the home, like door locks. Without proper protection, these devices can inflict serious harm.

A recent study on Samsung SmartThings brought to light security risks of such IoT apps, largely caused by inadequate protection under the framework [56]. Most concerning is the


Things to secure?

Contact Sensor
Tap to set

Motion Sensor
Tap to set

Knock Sensor
Tap to set

Three-Axis Sensor
You can't currently add this

Temperature monitor?

Temperature Sensor
Tap to set

Figure 3.13: The installation interface of SmartApp Safety Watch lists configuration options without connecting to higher-level app functionality. There is also no guarantee the app's actual behavior is consistent with its description.

overprivilege problem in SmartApp authorization. Each SmartApp asks for a set of *capabilities* (the device functionality the app needs), and the user must choose the IoT devices to perform respective functions for the app (for example, see Figure 3.13). In mapping capabilities to devices, the user allows the IoT app to perform the set of operations defined by those capabilities (e.g., turn on a light, unlock the door) based on event triggers (e.g., the room becomes dark, a valid token is detected near the door). However, this *implicit authorization* suffers from issues related to coarse granularity and context ignorance, namely that an app given *any* capability (e.g., monitoring battery status) of a device (e.g., a smart lock) is automatically granted *unlimited* access to the whole device (e.g., lock, unlock) and allowed to subscribe to *all* its events (e.g., when locked or unlocked).

In addition to the overprivilege that results from conflating all capabilities of a single device, malicious IoT apps can overprivilege themselves by requesting unneeded, and sometimes dangerous, permissions. While asking users to authorize third-party apps' access to IoT devices would, in concept, seem to prevent this sort of overprivilege, prior work on permissions systems for mobile apps has repeatedly documented that users often fail to act based on, or even understand these permission screens [57, 58, 28].

Even worse, unlike the Android permission model, which asks the user for permission to access specific resources on a *single* device (e.g., location, audio, camera, etc.), access control in a smarthome system is much more complicated. The policy is applied across devices, defining the operations that should take place on certain devices in certain scenarios, as described by the events observed by other devices (e.g., "ring the bell when someone knocks on the door"). Explaining such complicated policies to users is challenging, and effective authorization assistance should certainly go beyond what is provided by SmartThings (illustrated in Figure 3.13). In particular, it may be difficult for a user to understand what is being requested in the capability authorization UI, due to the gap between the app's high-level mission and the technical details of the capabilities it seeks across devices. As an example, a typical user may have no idea how the ability to read from an accelerometer relates to the purpose of detecting when someone knocks on a door. Furthermore, in the absence of robust monitoring and enforcement by the platform, the authorization system provides little guarantee that the capabilities requested by an app actually align with the app description.

As a result, despite the existing authorization system for IoT platforms, there can exist a crucial *gap between what a user believes an IoT App will do, and what the app actually does.* The idea that privacy is context-sensitive has been widely studied [59]. For example, providing an individual's sensitive health information to a doctor for the purpose of treating the individual would often not violate the notion of contextual integrity, whereas providing the same information to the individual's financial institution would likely violate his or her privacy. A similar principle holds in the IoT ecosystem. If an IoT app describes its own purpose as unlocking the door when a visitor arrives, it is likely no surprise to a user that the app can unlock the door. If, however, the same app had advertised itself as a temperature-monitoring app, a user would likely find the app's ability to unlock the door to be a security risk.

In this project, we propose new user-centered authorization and system-level enforcement mechanisms for current and future IoT platforms. We designed our approach, *SmartAuth*, to minimize the gap between a user's expectations of what an IoT app will do and the app's true functionality. To this end, SmartAuth learns about each IoT app's actual functionality by automatically harvesting and analyzing information from sources such as the app's source code, code annotations, and capability requests. Because the natural-language description developers write to advertise an app in the app store is the key source of users' expectations for what the app will do, we use natural language processing (NLP) to automatically extract capabilities from this description.

SmartAuth then compares the app's actual functionality (determined through program analysis) to the functionality developers represent (determined through NLP). This automated process is far from trivial because an in-depth understanding of the app focuses not only on the *semantics* of the app activities, but also their *context* among the app's broader goals. Our approach for achieving this level of contextual understanding relies on program analysis of the SmartApp's source code and applying NLP techniques to code annotations (e.g., the constant string for explaining the position of a sensor). We use further NLP to analyze the app description provided by the developer to extract higher-level information about the stated functionality, including entities (e.g., "a coffee machine"), actions (e.g., "taking a shower"), and their relationships (e.g., "turn on the coffee machine after taking a shower"). SmartAuth then compares such descriptions against insights from program and annotation analysis to verify that the requested capabilities and called APIs match the stated functionality, leveraging semantic relations among different concepts and auxiliary information that correlates them. For example, an annotation indicating a "bathroom" and the activity "take a shower" are used to identify the location of the humidity sensor of interest.

To minimize the burden on the user, SmartAuth automatically allows functionality that is consistent between the app's natural-language description and code, yet highlights for users discrepancies between the description and code since these are potentially unexpected behaviors. SmartAuth employs natural-language-generation techniques to explain, and seek approval for, these unexpected behaviors. The outcome of this verification analysis is presented to the user through an automatically created interface that is built around a typical user's mental model (for example, as in Figure 3.16). SmartAuth then works within the platform to enforce the user's authorization policy for the IoT app.

We incorporated SmartAuth into the Samsung SmartThings platform as a proof of concept. We evaluated our implementation over the 180 apps currently available in the SmartThings marketplace. SmartAuth successfully recovered authorization related information (with a false positive rate of 3.8% (7 out of 180) and no false negatives) within 10 seconds. We found that 16.7% (30 out of 180) of apps exhibit the new type of overprivilege in which some functionality is not described to the user even though these apps have passed the official code review from Samsung [60]. In some cases, the problem comes from the brevity of the descriptions, such as an app stating it can "control some devices" in your home. In other cases, however, hidden functionality is more security-sensitive, e.g., accessing and actuating an alarm without consent.

We also performed user studies to evaluate SmartAuth's impact on users' decision-making process for installing IoT apps[9]. In a 100-participant laboratory study, we found that SmartAuth helped users better understand the implicit policies within apps, effectively identify security risks, and further make well-informed decisions to mitigate overprivilege hazards. For instance, when using the current Samsung SmartThings interface to choose between two similar apps, one of which was overprivileged, roughly half of participants chose to install

[9]Our user studies were conducted with IRB approval.

the overprivileged app in each of five tasks. Using SmartAuth, however, the majority of participants chose the apps whose privileges better matched their descriptions, successfully avoiding the overprivileged apps.

We also patch the 180 SmartApps automatically to validate the compatibility of our policy enforcement mechanism. By observing the app behaviors when we trigger events in the applications, we find no apparent conflicts with SmartAuth. Given our observations of the effectiveness of the technique, the low performance cost, and the high compatibility with existing apps and platforms, we believe that SmartAuth can be utilized by IoT app marketplaces to vet submitted apps and enhance authorization mechanisms, thereby providing better user protection.

Our key contributions are as follows:

- We propose the SmartAuth authorization mechanism for protecting users under current and future smarthome platforms, using the insights from code analysis and NLP of app descriptions. This approach offers a new solution to the overprivilege problem and contributes to the process of human-centered secure computing.
- We design a new policy enforcement mechanism, compatible with current home automation frameworks, which enforces complicated, context-sensitive security policies with low overhead.
- We evaluate SmartAuth over real-world applications and human subjects, demonstrating the efficacy and usability of our approach to mitigate the security risks of overprivileged IoT apps.

3.2.2 Background

We provide relevant background information about home automation systems, challenges in the IoT app landscape, and useful tools for natural language processing (NLP).

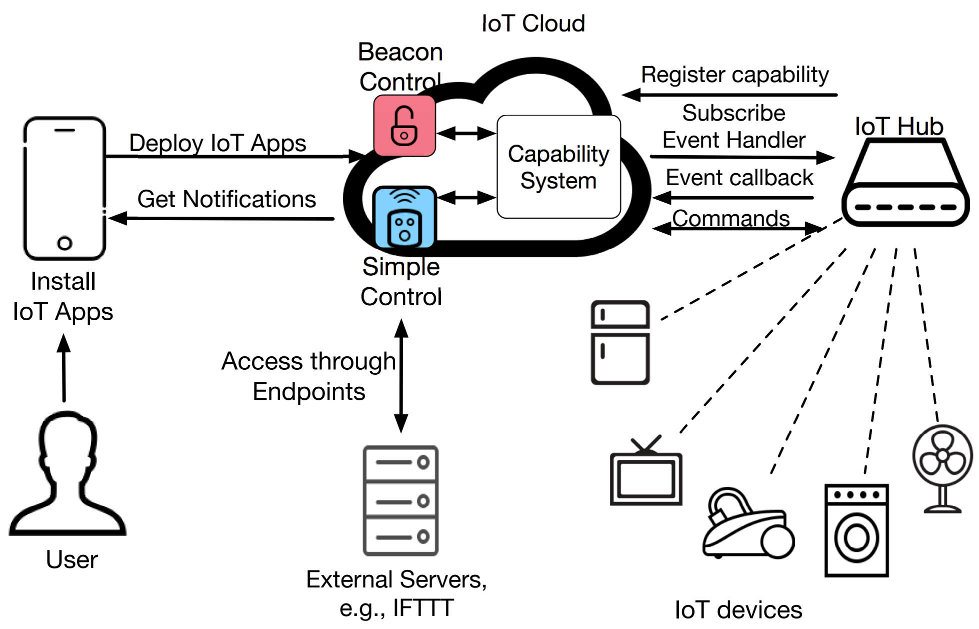


Figure 3.14: Users install commercial IoT apps through their mobile devices, allowing the vendor's IoT cloud to interact with the user's locally deployed IoT devices through direct Internet connectivity or an IoT hub. IoT apps pair event handlers to IoT devices, issue direct commands to IoT devices, and provide web interfaces to interact with external servers.

Home Automation Systems

Home automation is growing with consumers, with many homeowners deploying cloud-connected devices such as thermostats, surveillance systems, and smart door locks. Recent studies predict home automation products will have $100 billion in revenue by 2020 [61], drawing even more vendors into the area. As representative examples, Samsung SmartThings and Vera MiOS [62] connect smart devices with a smart hub which is then linked to a remote cloud server. Such vendors typically host third-party IoT apps in the cloud, allowing remote monitoring and control of a user's home environment.

Figure 3.14 illustrates a typical home automation system architecture. We use Samsung SmartThings to exemplify key concepts and components of such a system.

IoT apps written by third-party developers can get access to the status of sensors and control devices within a user's home environment. Such access provides the basic building blocks of functionality to help users manage their home, for example turning on a heater only when the temperature falls below the set point. Figure 3.14 depicts cloud-based IoT apps Beacon Control and Simple Control installed by a user from their mobile device and with access to the user's relevant IoT devices.

Current IoT platforms use *capabilities* [63] to describe app functionality and request access control and authorization decisions from app users. Unlike permissions, capability schemes are not designed for security, but rather for device functionality. A smart lighting application, for example, would have capabilities to read or control the light switch, light level, and battery level. Because of the complexity of home automation systems, capabilities in such platforms are often coarse-grained. One capability might allow an app to check several device attributes (status variables) or issue a variety of commands. This functionality-oriented design creates potential privacy risks, as granting an app a capability for a device allows it to access all aspects of the device's status and fully control the device.

An IoT app can also act as a *web service* to interact with the outside world, referred to as an *endpoint* in Samsung SmartThings. Such an app handles remote commands from servers and reacts accordingly. Many home automation platforms support standard authentication and authorization mechanisms such as OAuth to grant permission to third parties for commanding or accessing devices.

Threat Model

We consider malicious smarthome applications or benign but vulnerable smarthome applications as attackers. In particular, we focus on the smarthome applications that request permissions more than they need for their functionality. We assume that the smarthome platform is secure and not buggy, and the users are benign. We assume that the users understand the functionality of the smarthome applications through reading the applications' names and descriptions.

IoT App Security Challenges

Beyond basic overprivilege where an app requests a capability that is not needed, previous research on IoT apps has studied two additional types of overprivilege [56]: coarse capability and device-app binding. The former occurs when a capability needed to support app functionality also allows unneeded activities. The latter involves implicitly granting a device additional capabilities that are not needed or intended.

We have identified an additional type of overprivilege that relates not only to the functionality of the IoT app, but also to the user's perception of the app functionality, as seen through the app description. We observe that several IoT apps exhibit capability-enabled functional behaviors that are not disclosed to the user, causing a discrepancy between the user's mental model and the actual privilege of the app. We refer to this problem as *undisclosed overprivilege*. This kind of overprivilege has been discussed in mobile apps [64], but was never studied in the IoT space. An example of this type is an IoT app that describes the ability to control lights while requesting capabilities to read and control a door lock. Previous approaches may not flag this app as overprivileged, as long as the capabilities are used. In fact, even after a majority of Samsung SmartThings apps were removed from the market due to the previously reported overprivilege issues [56], we found that 16.7% (30 out of 180) of the remaining apps still exhibit overprivilege risks. Details about identifying the overprivilege risks are in Section 3.2.4.

Remote access is also an important security risk, as it enables apps to send sensitive data to and receive commands from third-party servers. In our study, we found 27 cases of such behavior, including cases where data was shared without user consent, a clear privacy concern. A SmartApp's ability to act as a web service expands the attack surface and potentially allows a malicious server to send dangerous commands to an app running on a user's smart devices, even though users may not expect such remote control. We observed 17 apps with this behavior. Similar to the undisclosed overprivilege, remote access does not match the user's mental model, which illustrates a gap in the current configuration and approval process.

Based on these observations, a general threat in the IoT app landscape is the ability for a malicious or compromised IoT app to steal information from sensors or home appliances or to gain unauthorized access to IoT device functionality. Even if the IoT platform itself is secure and trustworthy and previous issues of authentication and unprotected communication are patched [56, 65, 66], such issues with malicious apps may remain.

NLP Technologies

Since our approach analyzes app descriptions and gaps in users' expectations, we rely on several existing tools and techniques for natural language processing (NLP). The following tools are employed in our work.

Word2Vec [67] is a state-of-the-art tool used to produce word embedding that maps words to vectors of real numbers. Specifically, Word2Vec models are neural networks trained to represent linguistic contexts of words. We use Word2Vec to determine the relationship between two words by calculating the distance between the words in the embedding space. Word2Vec has many advantages over previous approaches, including catching syntactic and semantic information better than WordNet [68] and achieving lower false positive rates than ESA [69].

Part-of-speech (POS) tagging is used to identify a word's part of speech (e.g., noun or verb) based on definition and context. A word's relations with adjacent and related words in a phrase, sentence, or paragraph impact the POS tag assigned to a word of interest. In our work, we rely on the highly accurate Stanford POS Tagger [70].

We also rely on the typed dependencies analysis [71] to understand the grammatical structures of sentences, grouping words together to recognize phrases and pair subjects or objects with verbs. The Stanford parser applies language models constructed from hand-parsed sentences to accurately analyze sentences of interest.

3.2.3 SmartAuth Design Overview

In this section, we present the high-level design of SmartAuth, including our design goals, system architecture, and security policy model.

Given the unique security challenges of smarthome systems, we believe that an authorization system for IoT apps should be designed to achieve the following goals.

- *Least-privilege*: The system should grant only the minimum privileges to an IoT app, just enough to support the desired functionality.
- *IoT-specific*: Compared with authorization models for mobile devices, which are designed to manage a single device, the authorization system for a smarthome framework should meet the needs for multi-device, context-based, automatic operations. Permission models based on manifest permissions or run-time prompts, such as those employed in Android or iOS, either do not allow users to make context-based decisions or cannot satisfy real-time demands (e.g., approval to actuate an alarm when fire is detected).
- *Usable*: The authorization system should be human-centric, minimizing the burden on users while supporting effective authorization decisions.
- *Lightweight*: The authorization approach should not inhibit performance with significant overhead.
- *Compatible*: The authorization approach should be compatible with existing smarthome platforms and applications without breaking app functionality.

The key observation is that authorization decisions are made by humans, so a critical goal is providing a human-centric design that helps users recover adequate semantic information from IoT apps and presents it in a way that supports well-informed decision making. Our design thus aims for an intelligent authorization system that extensively utilizes semantic

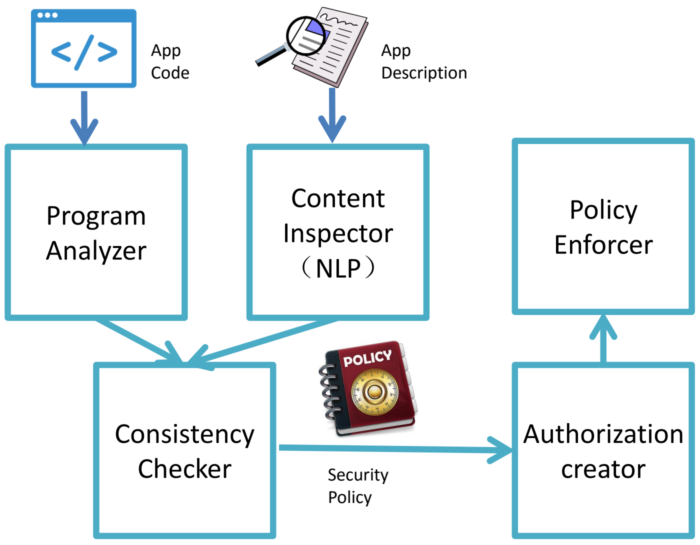


Figure 3.15: We provide a high-level block diagram to illustrate the design overview of our SmartAuth system.

analysis techniques to automatically understand an IoT app description, code, parameters, and annotations; analyzes their semantic meaning to discovery inconsistency; and automatically generates natural-language explanations of findings for the user.

Based on these design principles, our SmartAuth system includes five components: a program analyzer, a content inspector, a consistency checker, an authorization creator and a policy enforcer, as illustrated in Figure 3.15. The code analyzer extracts the semantics of an IoT app through program analysis and NLP of app code and annotations, creating a set of *privileges* that support the app functionality. In parallel, the content inspector performs NLP on the app description to identify the required privileges explained to the user. The consistency checker compares the results of code analysis and content inspection to generate security policies and identify discrepancies between what is claimed in the description and what the app actually does. These policies and information needed to support user decisions are then presented through an authorization interface produced automatically by the authorization creator, and the resulting policies are then implemented by the policy enforcer.

Our security policy model for the smarthome architecture is described in the form of a triple

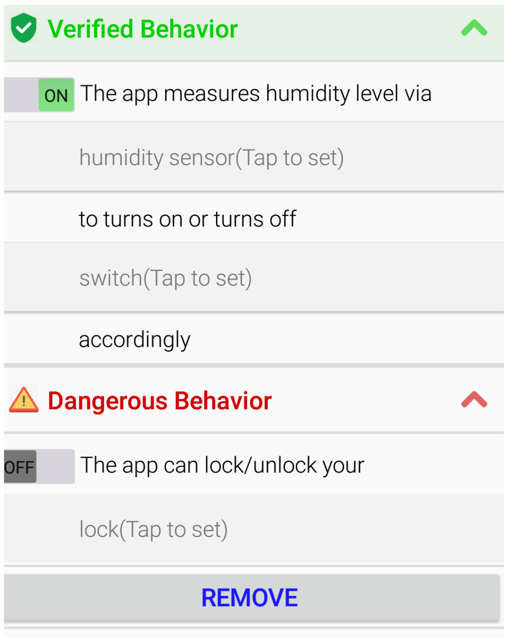


Figure 3.16: We illustrate the security policy generated for the Humidity Alert app, which is communicated to the user to request authorization.

(E, A, T). Item E represents the events, inputs, or measurements involving IoT devices and describes the *context* of the policy. Item A represents the actions triggered by elements of E, including commands such as "turn on". Item T represents the group of *targets* of the actions in A, such as a light receiving a command, noting that an empty target implies broadcast of a message or command. This model captures typical IoT app functionality, as apps are designed to issue commands to respond to observed state changes.

This model describes not only the policy produced by the authorization process, but also the privileges both claimed in an app's description and recovered from its code. Analysis of the policy actions thus allows identification of overprivilege and presentation of conflicts or situations that require the user to make a policy decision. Figure 3.16 illustrates an example of such policies.

3.2.4 Design and Implementation

In this section, we detail our design and implementation of SmartAuth.

Automatically Discovering App Behaviors

To extract an app's security-critical behaviors, we perform static analysis on the app's source code and use NLP techniques on code annotations and API documents.

We collected the source code for 180 Samsung SmartThings apps from a source-level market in May 2016 [72]. This represents 100% of open-source SmartApps and 80.2% (180 out of 224) of all SmartApps at that time.

For each app collected, we parse its code and create an Abstract Syntax Tree (AST) from the code, resolving classes, static imports, and variable scope. We choose to do AST transformation for the app analysis for two reasons: (1) SmartThings apps are written in Groovy, which transfers methods calls into reflective ones and creates challenges for existing binary analysis tools to deal with reflections, (2) we have access to the source code which is suitable for AST transformation. We extract the following key components of the AST for further analysis: (1) method names, (2) variable names and scope, (3) a variety of expressions, and (4) conditional statements.

Since capabilities are directly associate with the security behaviors, we first extract the capabilities. Since apps request capabilities in their preference block, we search for the text "capability." in the preference block. We compare the search result with the list of capabilities we collect from IoT app documents and add those found onto the list of requested capabilities. Note that we maintain a global mapping of capabilities to commands and attributes, and that one capability can involve multiple commands and attributes.

To identify what features of the capabilities that the app need, we analyze the commands and attributes associated with the capabilities. Using our global mapping of capabilities to commands and attributes, we search on the AST for the relevant commands called and attributes subscribed. During this process, we identify and create a list of the methods and commands used, together with another list of methods triggered in device subscription.

Recall that a SmartApp gets status updates by subscribing to events. Apps normally register event subscription in the following format: `subscribe(device, attribute, handler)`, where `device` is the IoT device to which a SmartApp subscribes, `attribute` is the device's status whose change is being subscribed such as battery level or temperature, and `handler` is a method invoked when the event occurs.

We then generate the security policy, starting from the method invoked on event subscription and perform a forward tracing. We first analyze the invoked function's code blocks to determine whether it contains conditional statements, which we analyze immediately. Otherwise, we trace into the callee function. In the condition blocks, we look for (1) what the event is and (2) what the object and action are. The invoked function of the event subscription takes a parameter from the subscription, and the parameter carries information about the event. Combining such information with the variable information from the AST, we identify both the event and associated capability. We further identify the action triggered by the event. For example, an app might control a heater when the measured temperature is above a threshold. Distinguishing whether the app turns the heater on or off is critical. We thus search the result statement for commands that control a device. If so, we continue our analysis to match the capability through variable analysis. Otherwise, we record the event and trace into the callee function.

The previous analysis covers an app's direct access to IoT devices, which we use to identify overprivilege. We also analyze whether the app has remote access to servers other than the SmartThings cloud. We consider two types of remote access: whether the app sends data to the remote server and whether the app works as a web service to take commands from the remote server. Both cases are privacy-invasive and likely violate user expectations. We search the AST to match patterns including OAuth, `createAccessToken`, and `groovyx.net.http`.

Beyond analyzing code, we also examine clues from code annotations (e.g., comments and text strings) to gain further information about the context and states of IoT devices. We apply Stanford POS Tagging and analyze the nouns to determine whether they represent location or time contexts. We find that most context clues in smarthomes relate to a place in the home, such as a bedroom. For example, we can extract that the humidity sensor is associated with bathroom from understanding the annotation in the following code snippet:

```
section("Bathroom␣humidity␣sensor") {
        input "bathroom",
        "capability.relativeHumidityMeasurement",
        title: "Which␣humidity␣sensor?"
        }
```

Analyzing App Descriptions

A key goal of our project is revealing any discrepancy between what the app claims to do and what it actually does. To find such discrepancies, we use NLP techniques to extract the security policy from the app's free-text description and program analysis to compare it with the security policy extracted from the code. We extract and correlate the behaviors in three layers: (1) entity, (2) context and action, (3) condition.

To begin inferring the security policy from human-written, free-text descriptions, we parse the description. We first identify the parts of speech of the words used, then analyze the description's structure to find typed dependencies. Nouns and verbs are often related to entities; for example, movement might be related to a motion sensor. From the structure of the descriptions, we can then infer the relationship between entities by identifying the typed dependencies. For instance, in the phrase "lock the door", the typed dependency is $dobj(lock, door)$, meaning that the noun *door* is the accusative object of the verb *lock*. In other words, *lock* might be the *action* and *door* might be the *target* of the security policy. Usually, cases are more complex, and following we show how we analyze them.

Specifically, we apply the Stanford POS Tagger to identify parts of speech and use the

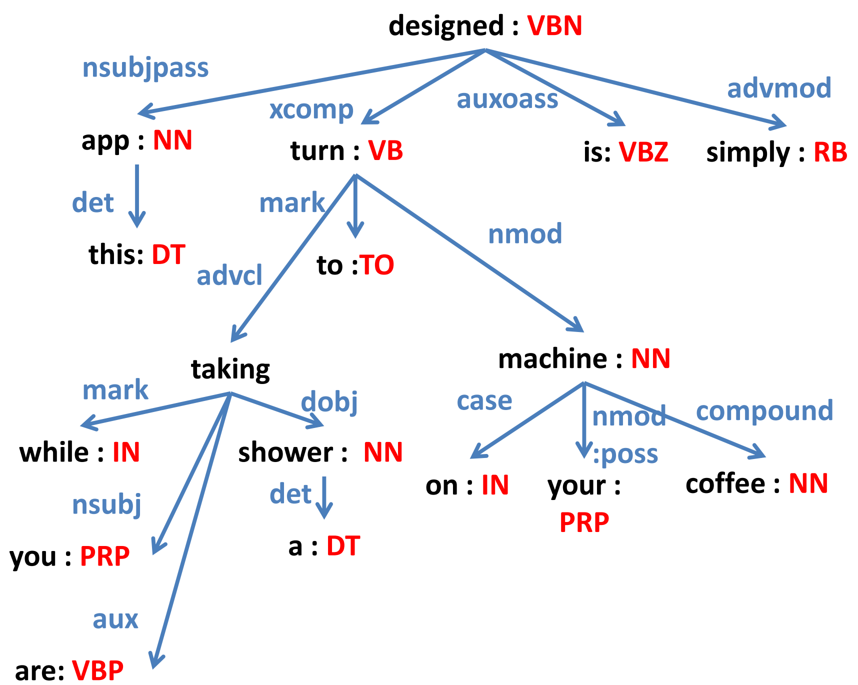


Figure 3.17: As an example, we illustrate NLP analysis of the Coffee After Shower description: "This app is designed simply to turn on your coffee machine while you are taking a shower." Red characters indicate parts of speech (e.g., "VB" stands for verb). Blue characters are typed dependencies (e.g., "advcl" stands for adverbial clause modifier).

Stanford Parser to analyze sentence structure, including typed dependencies. We follow standard NLP practices, such as removing stop words [73] (e.g., "a," "this"). Figure 3.17 provides an example of such analysis.

Because noun phrases and verb phrases usually describe the functionality of an app, we analyze these phrases to pinpoint the relevant entities. However, as the description is written by developers and language can be very complicated, doing so can be challenging. In addition, the device's context can significantly impact the implications of the entities. To overcome these difficulties, we design and implement the following process:

- The most straight-forward case is when the description explicitly includes the name of the entity (e.g., humidity sensors). If so, we match words directly.

- Because of the diversity of language, the first step may not produce meaningful results. However, even when the description does not contain the device name, the description might contain words that are often used in contexts related to the devices. For example, the description may mention detecting a flood, which relates to a humidity or moisture sensor. We identify how related the words used in the description are to the relevant devices through a word distance model that combines Word2Vec with a language model trained from Google News [74]. This language model includes word vectors for a vocabulary of three million words and phrases trained on roughly 100 billion words.

- The most challenging case is when the words in the description are not directly related to the entity in the security policy we generated from code. To address this issue, we compare the description to the context clues from code annotations. Consider the example in Figure 3.18. We first extract the entity "bathroom" (the context clue) from the annotation for the *humidity sensor* (`capability.relativeHumidityMeasu-rement`), as identified through the code analysis (Section 3.2.4). We only identify one entity from one annotation for the applications we analyze. This entity is then compared with the entity "shower" recovered from the description using Word2Vec, which reveals their semantic relation. In this way, we link "taking a shower" to the humidity sensor. Similarly, the clue "coffee" is used to relate "coffee machine" in the description to *switch* (`capability.switch`), a device recovered from the code.

Actually, simply connecting the entity from the description to devices in code is insufficient for determining whether only expected behaviors (as specified in the description) happen. For example, "lock the door when nobody is at home" and "unlock the door when nobody is at home" have starkly different security implications. To compare the semantics of an activity in the description to the operation of a device, we utilize a knowledge-based model. Specifically, we parse the API documentation of SmartThings to generate the attribute model and command model, that is, the sets of keywords for attributes and commands that represent their semantics. Then, we parse the words and phrases in the description connected to the entity-related word. This can be done by going through the typed-dependencies graph. For example, in Figure 3.18, we have identified that “coffee machine” is an entity, and we then parse the related phrase for the coffee machine is “turn on.” Such phrases will be compared with the keywords in the attribute and command models to find matches. During the matching process, we will first identify exact matches, and if we cannot find an exact match, we will compare the words in the phrases with the words in the attribute and command models by word vectors to identify the app’s behaviors.

After comparing the devices actually used in the code to those mentioned in the description, we also need to know whether the actual control flow matches that of the policy model. The causal relationship is critical for multi-device management where devices have impacts on each other. For example, two IoT apps may both ask for access to a door lock, motion sensor on the door, and presence sensor. A benign app might unlock the door when a family member is at the door and locks it when someone other than a family member is there. A malicious app might open the door anytime anyone is there. These two apps use the same devices, but with different control flows.

To perform causal analysis, we analyze the typed dependencies and build knowledge-based models of causal relationships. The causal relationships model is built with sentence structures and conjunctions related to conditional relationships. We apply the initial models to the descriptions to identify which devices caused other devices to change status. For example, the sentence “turn on the light *when* motion is sensed” represents that motion status is the cause, and turn on the light is the result. At the end of this process, we obtain *verified behaviors* that match in code and descriptions and *unexpected behaviors* that exhibit a mismatch. The accuracy of the analysis is in Section 3.2.5.

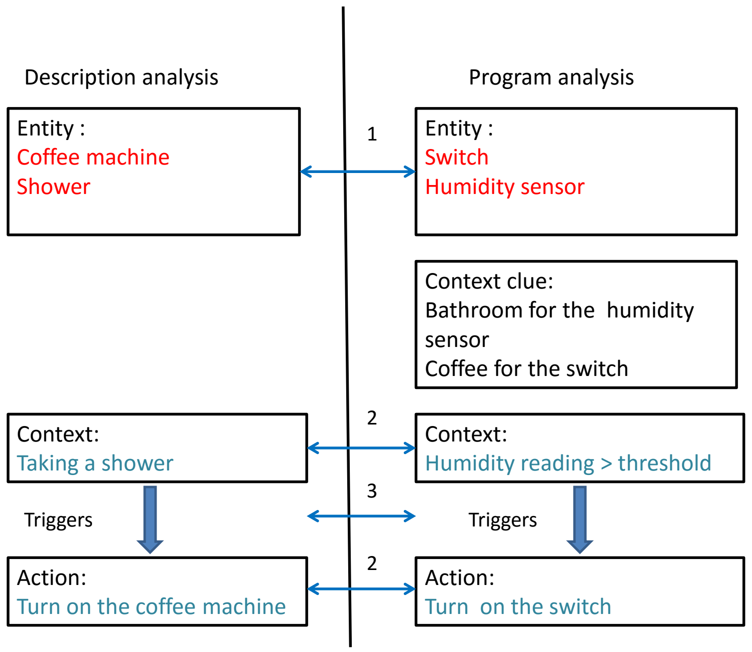


Figure 3.18: We illustrate the three-step policy correlation for the Coffee After Shower app. 1) We apply the context clues “bathroom” and “coffee” for entity correlation. 2) We use the attribute model and command model to extract and correlate the context and action. 3) We use typed-dependency analysis and causal relationship model to correlate the policies generated from the description and program analysis.

Authorization Interface Generation

Towards making usability a first-order concern in designing our authorization scheme, we first conduct an online survey to understand users’ expectations related to IoT app installation and the overprivilege problem. Using MTurk, we recruit adult participants who have experience using smartphones. In July 2016, we post surveys with the title “Smarthome Survey” on Mturk. We tell the participants that the survey is about their understanding of Smarthome. Participants will be compensated for 0.5 dollars after finishing the survey. To avoid biasing participants towards fraudulently claiming experience with SmartThings to participate in the survey, we do not require that participants have used any smarthome platforms to take the survey. However, we only analyze data from the 31.6% of the survey participants who have previous experience with SmartThings.

In the survey, we asked about: (1) experience using IoT platforms and demographics, (2) the factors they consider when installing third-party IoT apps, and (3) their perspective on smarthome capabilities. We received responses from 300 participants who had used SmartThings, identified by an average age of 30.8 years old (age range is 18-60) with a gender breakdown of 32% female, 67% male, 1% unknown.

We asked participants to respond on a five-point scale about how much they care about six factors they might consider when deciding whether or not to install a third-party SmartThings app. App functionality (66% (198 out of 300) strongly care, and 24% (72 out of 300) care) and privacy (57% (171 out of 300) strongly care, and 28% (84 out of 300) care) were the factors participants stated they cared about most in deciding whether to install an app.

To understand participants' perspective on smarthome capabilities, we asked participants to rate the sensitivity of different IoT device functions and to compare the sensitivity of SmartThings capabilities and Android/iOS permissions. To ensure that participants understood what we meant by *smarthome capabilities*, we both formally defined the concept and demonstrated it using an example screen shot from a SmartThings device permission screen.

We asked participants to rate the sensitivity of eight IoT device behaviors on a four-point scale ("not sensitive:1" to "very sensitive:4"). We find that participants have very different risk perceptions for different behaviors of the same IoT device. For example, we find the average sensitivity rating for app's ability to *unlock* their door is 3.28, whereas reading the battery level of their door is only 1.87 (Mann–Whitney $U = 21350, n1 = n2 = 300, P < 0.001$ two-tailed). These sharp distinctions highlight the importance of increasing the transparency to users about what precise behaviors an app will perform in the home, rather than considering all behaviors for a particular device monolithically. Our approach of automatically identifying discrepancies between the actual behavior of an app determined through program analysis and the free-text app descriptions that users generally rely on when considering whether to install apps [57] better supports these distinctions.

To this point, most research on app permissions has focused on smartphones. As a result, we asked participants to select one of four statements indicating whether they considered Android/iOS permissions and smarthome capabilities equally sensitive, Android/iOS permissions to be more sensitive, smarthome capabilities to be more sensitive, or whether they were unsure. Suggesting that smarthome capabilities are a crucial area for research progress, 69% (207 out of 300) of participants indicated that they considered smarthome capabilities to be more sensitive than Android/iOS permissions. Participants provided a free-text explanation of why, and we performed qualitative coding on these responses by two researchers (agreement rate=90.3%). The leading reason participants found IoT apps more sensitive is that they perceived the home environment to inherently present greater risks. For example, one participant wrote, "smarthome compromises can inflict serious damage or injury. Imagine being locked in your house, with the heat cranked up. Or an invader monitoring your location in the house, or studying your patterns. The risk involved in a smartphone knowing your location or accessing the devices, like reading contents, contacts or accessing the camera are far more limited in potential effects by an attacker."

In generating the user interface, we aim to minimize the burden on the user and provide information that matches the user's mental model of the system. We rely on a policy model that links app functionality with authorization. We first automatically summarize the security policy, removing redundant logic, and then create language models to translate the security policy into a human-understandable description using state-of-the-art natural language generation techniques SimpleNLG [75]. SimpleNLG works as realization engine that generates syntactic structures and linearises them. The newly generated description explains to the user in natural language what device attributes and commands are being used, and why. For example, the app monitors the temperature from the temperature sensor and whether someone is at home by the presence sensor to turn on a heater when it is cold and someone is home.

We designed our authorization approach to better align users' expectations with the actual behaviors of smarthome apps, as well as to reduce user burden during the authorization process. Because many users rely on app descriptions, rather than permissions screens, to evaluate smartphone apps [57], one way of reducing user burden is to assume that a user would implicitly grant an app the permission to perform actions stated in the app description. While any assumption that a user's actions with an app perfectly follow the user's intent is necessarily flawed, prior work on smartphone permissions [57] suggests that assuming a user would permit an app to perform the behaviors described in its app description is likely at least as robust as assuming that a user intended to grant the permissions specified on a permissions screen. We therefore minimize users' burden by automatically granting the attributes inferred from the app description. For the attributes that are not described in the app descripton, we present the user with our automatically generated description of the policy model rather than the potentially confusing settings that are currently used.

To highlight for users the most potentially risky unexpected behaviors, we design indicators about the risk levels. We classify behaviors into three categories: verified behaviors that match the claimed functionality, unexpected behaviors that are not sensitive, and dangerous behaviors that are unexpected and risky. We determine these risk levels by asking security experts and average users to rate their perceived risk based on status changes and device operations. An example authorization interface is shown in Figure 3.16.

Policy Enforcement

Once a user sets his or her policy settings through the user interface, we enforce the policy end-to-end by blocking unauthorized command and attribute access. Our policy-enforcement module performs the required filtering operations locally. This module could be integrated directly into the SmartThings Cloud.

We patched existing SmartApps to interact with our policy enforcement module using REST APIs as if they were interacting with the SmartThings Cloud. Every command or attribute function call in the patched app is substituted with an equivalent call to the module that includes the device handler, command or attribute name, and any additional parameters. After the module processes the request, a return value is sent back to the patched app and handed to the code that invokes this command or attribute, which is transparent to the original app. Similarly, the patched app also subscribes to events by connecting to the enforcement module.

The policy enforcement mechanism starts when the user begins to install a SmartApp. The user is directed to our enhanced interface to set up the devices used by the app and the policies that will govern access by the app. This information is transmitted to the policy-enforcement module to ensure that the app can only access what the user allows. Based on the policies, the module will make two type of decisions.

- *Commands and Attributes*: Whenever the module receives a command or attribute request from a patched app, it will extract the device ID and actions and check the associated policies from the database to see if the request is allowed by the app for the specified device. If allowed, the module will forward the request to the cloud service to execute and respond, after which the module will forward the response to the patched app. If denied, the request will be dropped and an error message will be returned to the patched app. We expect that SmartApps will already be designed to handle error messages, so the denial of requests should not impede normal operation. We further analyze compatibility in Section 3.2.5.

- *Events*: Whenever there is an event reported by the SmartThings Cloud, the module will retrieve the associated app IDs and policies from the database and forward the event only to the apps that are allowed to access the event according to the app policy.

The module thus blocks all unauthorized subscribe, command, and attribute requests.

3.2.5 Evaluation

We evaluate SmartAuth in several dimensions, finding SmarthAuth is effective at automatically extracting security policies, significantly helps users avoid overprivileged apps, and adds minimal performance overhead when enforcing users' desired policies.

Effectiveness in Extracting Policies

Our first step is to evaluate SmartAuth's ability to accurately identify unexpected behaviors. To this end, we manually analyze the description and the code of the 180 available SmartApps and compare these manual analyses with the results of the automatic analysis. We first go through the code to check what capabilities the app request, and the actions the app will take when certain things happen. Then we check whether these behaviors match the descriptions. In this process, we do not observe any false negatives, though we identify seven false positives (3.9%) in which SmartAuth flagged a behavior as unexpected, but manual analysis suggests the apps' behaviors match their free-text descriptions.

These false positive cases happen because of the limitations of our NLP techniques. Two of these cases occur because the apps use a product name to represent a device, but the product name is not relevant to its functionality. For example, Mini Hue Controller uses the Aeon Minimote[10] input device with SmartThings. Two cases occur because the app referenced another app to explain its functionality. For example, Keep Me Cozy Two claims that it "works the same as Keep Me Cozy, but enables you to pick an alternative temperature sensor in a separate space from the thermostat." These cases can be improved by named entity analysis to identify the app they refer to and merge the app's behavior into the current analysis. Another case occurs due to complicated logic spread through several sentences, causing the description to be ambiguous. Two cases occur because the correlation of the context is not intuitive, even from the human perspective. For example, it is not clear even

[10]`http://aeotec.com/homeautomation`

for a human to infer that vibration on the floor implies that someone has woken up at night. Note that we manually analyze the code of the smarthome apps to check whether they access information that is not needed for their functionality. Because the ground truth is based on human judgment, we might miss some malicious behaviors, although we have tried our best to identify them.

Impact on Users

We first describe our user study to evaluate how SmartAuth impacted users' app-installation decisions, followed by additional data on the usability of SmartAuth itself.

From September to October 2016, We performed a between-subjects user study in which we recruit 100 participants from Carnegie Mellon University Silicon Valley, Indiana University Bloomington, and Samsung Mountain view office[11]. Each participant came to our lab and used a phone we provided to complete app installation tasks, in addition to answering questions. We require that participants be adults who regularly use a mobile device and are knowledgable about home automation systems. The details of the study is in Section A.2.5 of the appendix. We verify participants understand key concepts of smartphones and home automation using screening questions. For example, we ask them how IoT apps are installed and what purposes IoT apps serve. We also ask questions about demographics, as well as questions about their experiences installing IoT apps. The protocol takes around 20 minutes. For the 100 participants in our study, their ages ranged from 19 to 41 years with a mean age of 25.7 years, and 59% of participants reported as male and 41% as female. The participants have education backgrounds ranging from high school to graduate school (2% high school, 47% bachelor, 51% graduate degree). 68% of participants have a technical background (engineers or students in computer science or related field). We carefully avoid the IoT developers when we recruit in Samsung Mountain view office because they are very familiar with the system and their results might be biased.

[11]We got IRB approvals for these experiments.

The study's primary task is choosing IoT apps to install using the phone we provide. For five different types of IoT apps, the participant chooses between one of two similar apps. Each of the two apps in a pair has identical functionality, yet only one of the two apps in a pair is overprivileged. To prevent this difference in permissions from being the obvious variable of interest, we used apps whose titles and descriptions were roughly comparable. For example, participants choose between "Lights Off with No Motion and Presence (by Bruce Adelsman)" that will "Turn lights off when no motion and presence is detected for a set period of time" and "Darken Behind Me (by Michael Struck)" that will "Turn your lights off after a period of no motion being observed." The apps used in the experiment are listed in the table A.2.

Each participant is randomly assigned into one of two groups, specifying whether they will see Samsung SmartThings' authorization interface or SmartAuth while completing all tasks. For each of the five app-selection tasks, participants saw the app installation page with two choices. We asked the participant to choose only one of the two apps to install, and to explain why.

In each of the five app pairs, if users choose among the two relatively similar apps effectively at random, not realizing that one app is overprivileged, roughly 50% of participants would choose the overprivileged app. Even though the current Samsung SmartThings authorization interfaces shows users a list of the devices the app can access, including potentially unexpected devices, this current interface did not help users avoid overprivileged apps. For each of the five tasks, between 48% (24 out of 50) and 60% (30 out of 50) of participants who saw the current SmartThings interface chose the overprivileged app, as shown in Figure 3.19.

In contrast, between 74% (37 out of 50) and 94% (47 out of 50) of participants who saw the SmartAuth interface successfully avoided the overprivileged app, differing significantly from the current SmartThings interface (Holm–Bonferroni corrected $\mathcal{X}^2$, $p \leq .022$ for all five tasks). Note that for two of the tasks (A and B in Figure 3.19), the overprivilege was a

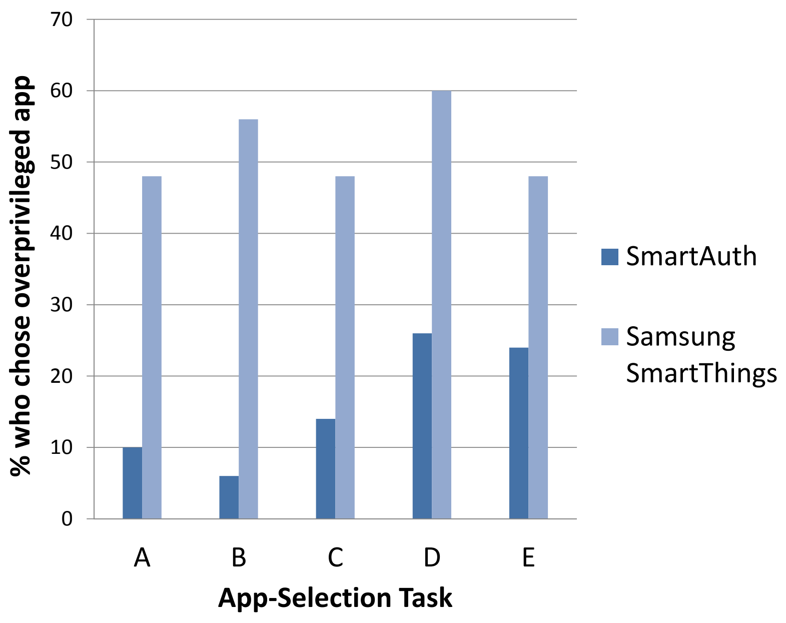


Figure 3.19: For Five tasks, participants chose between two similar IoT apps, one of which was overprivileged. This graph shows the proportion of participants who chose the overprivileged app. Similar to what one would expect from random selection, around half of the participants who saw the Samsung SmartThings interface chose the overprivileged app. In contrast, only between 6% and 26% of SmartAuth participants chose the overprivileged app.

potentially dangerous behavior (e.g., unlock a door), whereas the overprivilege for tasks C–E was potentially less risky (e.g., learn the temperature). For the two tasks with dangerous overprivilege, only 10% (5 out of 50) and 6% (3 out of 50) of SmartAuth participants, respectively, chose the overprivileged app. In contrast, 48% (24 out of 50) and 56% (28 out of 50) of participants who saw the current SmartThings interface, respectively, chose the overprivileged app. Even when they still chose the overprivileged app, we found that many SmartAuth participants were aware of the overprivilege, yet said they either did not care about the unexpected behaviors or thought the app might benefit from these behaviors in the future.

In addition to evaluating SmartAuth's impact on user behavior, we also measure the usability of SmartAuth itself. In the laboratory study, after users choose among pairs of apps and answer questions about privacy, we ask questions to elicit their perceptions of what the interface communicated to them. For some of these questions, participants respond to statements on a five-point Likert scale (from “1: strongly disagree” to “5: strongly agree”).

The first statement gauges the apparent completeness of explanations (“I feel that the app interface explains thoroughly why the app can access and control these sensors/doors”), and participants who used SmartAuth were more likely than those who used Samsung SmartThings to agree (SmartAuth mean 4.06, SmartThings mean 2.40, Mann–Whitney $U = 337.5, n1 = n2 = 50, P < 0.001$ two-tailed). The second statement measures the user’s comfort in making a decision (“I feel confident to make a decision whether or not to install the app after reading the interface”), and SmartAuth participants were significantly more confident in their decisions (SmartAuth mean 4.12, SmartThings mean 2.46, Mann–Whitney $U = 320.5, n1 = n2 = 50, P < 0.001$ two-tailed). The third statement evaluates the perceived difficulty of finding information (“It is difficult to find the information from the interface”), and SmartAuth participants were more likely to *disagree* with this difficulty, meaning they found it easier (SmartAuth mean 2.72, SmartThings mean 3.56, Mann–Whitney $U = 713, n1 = n2 = 50, P < 0.001$ two-tailed).

We also asked open-ended questions about what factors participants consider when deciding to install an app. Both SmartAuth and Samsung SmartThings participants focused on two factors in common: functionality and ease of configuration. However, SmartAuth participants also discussed privacy and unexpected or dangerous behaviors as a major factor. In comparison, only a single one of the 50 Samsung SmartThings participants pointed out a mismatch between the description and the Samsung SmartThings authorization screen as a factor.

Performance and Compatibility

To evaluate the performance impact and ease of deployability for SmartAuth, we collected all 180 open-source SmartApps in the Samsung SmartThings marketplace at the time of research. Our experiments demonstrate that SmartAuth is both lightweight and backward compatible.

We ran two performance tests: pre-processing performance (program analysis, description analysis, behavior correlations, and policy description generation) and run-time performance (authorization interface generation and policy enforcement). For testing the pre-processing performance, we analyzed the 180 apps for 10 times each to generate the policy description. On a 3.1 Ghz Intel Core i7 CPU with 16 GB memory, the pre-processing overhead for an app is 10.42 seconds on average. Since pre-processing is a one-time cost and can be done offline, we believe that the performance is reasonable even for vetting a large number of applications.

For the run-time performance test and compatibility test, we instrumented the SmartApp to interact with our policy server running on the Amazon EC2 cloud, which enforces the rules defined by the user. Given our purpose of evaluating the compatibility of our technique with existing SmartApps, we set the authorization policies (granting permissions to certain commands, attributes and event handlers) ourselves, instead of letting the user do that, as would happen in practice. We designed our experiments to test the technique in the worst-case scenarios. That is, we assume users would reject all unexpected and dangerous behaviors, requiring the maximum amount of policy enforcement. To enable large-scale testing without requiring the purchase of every physical SmartThings device, we used Samsung's online SmartApp simulator platform[12] instead of a mobile device. Instrumented apps are then installed on the simulator, and their functionalities are tested with simulated IoT devices.

As shown in Figure 3.20, we recorded the delay incurred by different command, attribute, and event handler actions. We performed 1800 experiments among the 180 SmartApps on a cloud server with 3.1 Ghz Intel Core i7 CPU and 1 GB memory. SmartAuth incurs an average delay of 35.4 msec, which is small relative to the dominant network latency in cloud-based

[12]`https://graph.api.smartthings.com/`

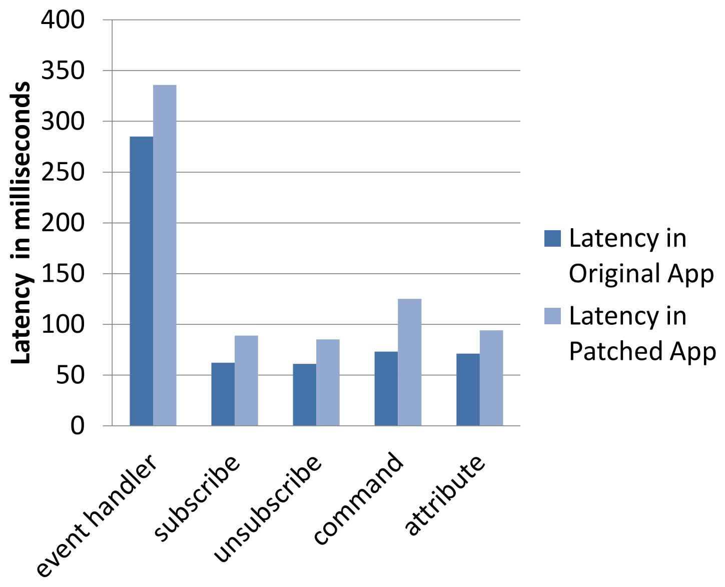


Figure 3.20: We plot the average delay of various functions in the SmartThings platform. The darker bar in each pair represents the delay in the unmodified platform with virtual devices, while the lighter represents the delay in our customized platform with the additional overhead introduced by SmartAuth. Event handlers incur the highest incremental overhead, while commands incur the highest proportional overhead (almost double the base case).

IoT platforms.

Next, we test the degree to which SmartAuth policies to mitigate overprivilege and block third-party remote access impact backward compatibility with existing SmartApps. As with our performance analysis, we test the worst case of users blocking all unexpected and dangerous behaviors and all remote access. We again test patched apps on Samsung's online simulator environment. We trigger events at least five times and insert debug messages into the modified apps' source code to observe apps' behaviors while they gather data from the cloud or when events have been triggered. To evaluate backward compatibility, we both observe app behaviors and analyze the debugging messages. For tests related to overprivilege policy, we focused on the 30 apps that exhibit undisclosed overprivilege according to the behavior correlation analysis. For the interested reader, these 30 apps are listed as Table 3.3. These apps are identified because they either request capabilities not mentioned in their descriptions (unexpected capability), or even worse, request capabilities that could do harm (e.g., unlocking the door). For example, the Smart Security app presents a description: “alerts you when there are intruders but not when you just got up for a glass of water in the middle of the night.” After scanning the source file, this app requests access to `motionSensor`, `contactSensor`, and `alarm` capabilities, satisfying the description, but also requests sensitive commands including turning on/off a switch, which is not mentioned in its description. Therefore, we mark this access as an unexpected behavior. For the remaining 150 apps, we further patch them to comply with our policy enforcement mechanism. Specifically, apps with coarse capability overprivilege and device-app binding overprivilege are also constrained to ensure the least privilege.

In our compatibility tests, **none** of the 180 apps crash after patching when we observe the apps’ behaviors in the simulators, even with overprivilege security rules enforced. Even after they are patched to remove overprivilege, the 180 apps behave the same as their original versions. In other words, patching does not break the functionalities claimed in the app’s description.

We further test how apps function if we block all third-party remote access, an extreme case where the user denies all such requests. Of the 180 apps, only six apps suffer from a loss of valid functionality. For example, Vinli Home Connect allows remote services to control IoT devices, and this functionality breaks entirely when we block remote access. We believe such examples will continue to be rather rare, especially when users are given clear information and useful options to configure the app’s security policy. In addition, we envision the possibility of a cloud-based reference monitor that could check run-time remote access and filter out dangerous access, but such a design is beyond the scope of this work.

3.2.6 Limitations

Although SmartAuth advances user-centered authorization for the IoT, its design has limitations. A malicious developer could use custom-defined methods and properties whose names mirror SmartThings commands and attributes, thus fooling the program analysis. A future version of SmartAuth could better recognize this technique. Our static analysis tool is based on Groovy AST transformation. If handled correctly, the tool can detect obfuscated logic (which cannot evade AST transformation), and obfuscated dynamic variable/function names can be handled with define-use analysis within source code scope. Furthermore, the accuracy of the NLP analysis depends on the quality of the description. A malicious developer could try to craft natural-language descriptions for which SmartAuth's NLP mistakenly extracts a malicious behavior from the description, even when humans would not perceive that the text discusses such a behavior. Future work could focus on recognizing these adversarial descriptions. External services like IFTTT could be the future work for our project. Our approach can be applied if we know the control flow information from IFTTT. External devices, if they are approved by Samsung, will be included in the capability system and covered by our project.

In addition, dynamic method invocation from remote servers is a threat that requires future investigation. However, this is less of a concern because Samsung bans dynamic method execution through code review [60]. In our compatibility analysis, we triggered different events at least five times each. While we did not observe any crashes, this is not proof that crashes will not occur.

Our user studies also have important limitations. While we did not draw attention to this fact, particularly attentive participants might have recognized that SmartAuth was a novel interface. This recognition might have biased participants to be complimentary of an interface they assumed was being tested, as well as to pay particular attention to the interface in the absence of habituation effects. Furthermore, users will not always have a choice between an overprivileged app and a less privileged variant, and it is an open question whether users might still install an overprivileged app if it is the only option. We have one assumption that users will read the app description when they decide to install apps. However, we did not run a formal user study to verify the assumption. We did observe in the lab study that most users paid attention to the app description, but it would be better to verify the assumption formally. Currently, the Smarthome market is still at an early stage, and most of the users are with a technical background. Many participants in our lab study have good technical background, which is representative for the current users. However, when the smarthome systems get much more popular, our participants might not be representative for future users.

3.3 Related work

In this section, we compare our work with previous work on permission systems, as well as privacy for social platforms and smarthome.

3.3.1 Improving the Permission Systems in Client OSes

Researchers tried to improve the permission systems. To design better permission systems that meet users' expectations, researchers first study users' expectations [76, 77]. Felt et al. presented a set of guidelines for requesting permissions in mobile OSes [35], reducing the number of warnings presented to users. The authors evaluated different user interfaces and interactions for granting permissions and compared the effects of different user interfaces on user experience. Based on these experiments, they developed guidelines for building user interfaces for different permissions. One of our goals in this project is also to reduce user burden, and we achieve this goal by applying the OOU primitives to further reduce the number of permissions and prompts. Tiwari et al. developed the "Bubbles" system that allows a user to define a digital boundary (called a bubble) around the data associated with an event and share information in the unit of a bubble [78]. The authors designed a permission model for “Bubbles” where a user can specify privacy requirements for each bubble. For example, a user can specify which other users and applications can access information about a wedding party. With “Bubbles”, a user can make in-context privacy decisions but also needs to specify a policy for each bubble, thereby increasing user burden. Compared with “Bubbles”, our permission model in Somex reduces user burden and still provides more functionality (context-aware recommendations) by applying the OOU primitives. Roesner et al. propose user-driven access control for managing resources in client OSes [42]. We advocate that this technique can also be leveraged in social platforms. Liu et al. propose using users' privacy profiles generated from permission settings from current apps to recommend privacy settings for new apps [79]. Researchers implement a package installer that allows users to add runtime constraints to the permissions [80]. In addition, researchers show that sending user nudges, especially about information access frequency, are helpful to improve the awareness of privacy on smartphone [81].

3.3.2 Improving Privacy for Sharing Information with Third-Party Apps on Social Platforms

Researchers have proposed ways to improve privacy on social platforms. In particular, we focus on the proposals that aim to improve user privacy when users share information to third-party apps. Felt et al. studied the top 150 web applications in Facebook app center and discovered that most applications can provide the desired functionality without accessing a user's private data [82]. They then proposed a “privacy-by-proxy” approach to shield against user identification. This approach uses placeholders for user data and anonymized user identity in applications. Their work was done in 2007 on Facebook web applications, but much of Facebook's functionality has changed, and most apps are now hosted externally. The original proposal for an opaque display feature only worked for apps hosted on Facebook's website, not for today's mobile apps and externally hosted web apps, while our design of opaque display supports mobile and web apps. Moreover, Felt et al. propose an opaque handle only for user identity, whereas we can extend the opaque handle to arbitrary context or even to an abstraction of complex context such as an activity. Our much broader applicability can encompass an ever-growing amount of context that the applications demand. In addition, we have a full-fledged implementation of the permission system applying all principles and many more feature-rich scenarios like Amazon, and knowledge box. Luo et al. propose to send fake data to social platforms to protect user privacy against social platforms and other users [83]. The privacy goal of this work is different because the authors consider social platforms and users as attackers, while our model only considers untrusted applications. This system also does not support context sharing. Singh et al. implemented information flow control to protect user data on social platforms [84]. The authors track the usage of user data and prevent the applications from leaking user information. Their work prevents abuse of information after the permission is granted, which is orthogonal to our work. In this project, we focus on providing limited information to applications while still providing useful functionality. Researchers have proposed many approaches to de-anonymize social network activities [85, 86]. We design and apply our primitives with special attention to the privacy side effects, so that malicious applications cannot utilize our scheme to get any extra information to track use across applications or infer that two users are connected.

Efforts have also been made to build decentralized social networks to protect user privacy. These papers are generally related to our topic but with a different focus on broader problems of social network privacy. Baden et al. proposed using attribute-based encryption to protect user personal data on social networks and enable fine-grained policy for users to control information sharing [87]. They build a prototype wall posting application that simulates Facebook's wall functionality in 2009. Safebook [88] is a decentrailized social network that builds on the real-life trust between users. We focus on a different, practical, and pressing problem, the solution to which helps social networks to protect user data from untrusted third-party applications. Our solution is evaluated systematically in many diversified third-party applications and is effective for modern and emerging social platforms.

3.3.3 Bridging the Gap between User Expectation and App Behaviors for Mobile Apps

Many researchers have studied permission systems for mobile devices and propose methods to bridge the gap between user expectations and app behaviors. While some insights apply to both domains, the unique features of IoT platforms introduce new security and privacy challenges. Most similarly to SmartAuth, the Whyper system identifies Android permissions that might be used from the app's description [89]. The researchers do an extensive analysis of app descriptions and match them with permissions, but they do not evaluate the real security behaviors from the code of the applications. Even for analyzing descriptions, SmartAuth is fundamentally different because Android permissions and APIs have very specific privacy implications. In contrast, reasoning about implications in the IoT is much more context-sensitive, necessitating our further use of NLP. Zhang et al. instead analyzed Android apps using static analysis, generating descriptions for the security behaviors in the applications [90]. These descriptions are helpful for users to understand the app's behavior. However, users are burdened with reading the long logs and still need to use the original Android interface to authorize. In contrast, we remove many overprivilege cases automatically, designing and testing a new scheme that minimizes user burden.

Many approaches build on this prior work. AutoCog [91] compares descriptions with permissions requested. AsDroid [92] analyzes the text in the user interface and the current behavior to see whether it is a stealthy behavior. Appcontext [64] analyzes context that triggers security behaviors and compares the context among apps to differentiate between benign and malicious apps. Other researchers compare app behaviors to app descriptions by clustering applications with similar functionality and finding apps that use uncommon APIs [93]. Besides mobile permissions, researchers also look into privacy policies to identify privacy inconsistency in the code and the privacy policy [94].

Researchers also study users' expectations for permissions, focusing on users' perceived risks [28, 77]. For example, Egelman et al. investigate users' perceptions of sensitive data stored on their phones, including banking information and home address [95]. However, our study about users' mental model about IoT permission makes new a contribution because the perceptions and requirements in IoT platforms are different from mobile platforms. Many researchers have sought to improve mobile permissions. For example, Liu et al. propose privacy profiles to ease user burden [79]. Almuhimedi et al. propose information visualization to improve user awareness of risks [81], Harbach et al. suggest using personal examples to better explain permission requests [96], and Tan et al. suggest using developer-specified explanations for understanding [97]. Researchers have also provided general guidelines for designing permission systems [35, 42]. Users' perceptions of mobile permissions and IoT permissions share some characteristics. For instance, Wijesekera et al. observe through a field study that mobile apps sometimes violate contextual integrity by accessing unexpected resources [98]. However, due to the differing privacy and security implications for IoT platforms, SmartAuth further rethinks the design of authorization systems.

3.3.4 IoT Security and Privacy

IoT security and privacy is an emerging area. Previous research has largely focused on identifying security and privacy vulnerabilities. Naveed et al. discuss the security binding problems of smart devices that are external to the mobile phone [65]. Fernandes et al. run a black-box analysis of Samsung SmartThings, pinpointing the overprivilege problem [56]. We instead reconceptualize overprivilege to be more practical and user-centered. To enhance security and privacy goals in IoT and home automation systems, FlowFence [99] uses information flow control and explicitly isolates sensitive data inside sandboxes. This approach requires intensive modification to SmartApps, and the evaluation is done on Android instead of a real smarthome system. Jia et al. gather information before executing sensitive actions and ask for user approval through frequent run-time prompts [100]. However, in-context prompts cannot satisfy the real-time automation of IoT apps (e.g., users need to be awake to approve when an app needs to alarm the user that there is a fire at night). Users will likely become habituated to approving these prompts, mistakenly approving unexpected behaviors. BLE-Guardian [101] controls who can discover, scan, and connect to an IoT interface. CIDS [102] designs an anomaly-based intrusion detection system to detect in-vehicle attacks by measuring fingerprints from deployed ECUs based on clock behaviors. Sivaraman et al. propose managing IoT devices through software-defined networking (SDN) based on day-to-day activities [103].

Beyond framework or architecture solutions, enhancing the security of smart devices is also a common countermeasure against attacks from remote or near field communication surfaces. For example, Seda [104] proposed their attestation protocol for embedded devices. Through software attestation and showing states gathered from booting sequences, Seda can construct a security model for swarm attestation. Similar approaches to ensure IoT or smart device integrity [105, 66, 106, 102] complement our system.

Some researchers have also examined IoT privacy from a usability perspective. For example, Egelman et al. suggest using crowdsourcing to improve IoT devices' privacy indicators [107]. Further, Ur et al. investigate parents' and teens' perspectives on smarthome privacy [108] and Demiris et al. study seniors' privacy perspectives for smarthome [109]. In contrast, we design and user-test a new usable IoT authorization scheme.

Table 3.3: Compatibility test results among 30 over privelged SmartApps.

App	**Unexpected Capability**	**Dangerous Capability**	**Compatible**
Alfred Workflow	switch	lock	Not if block remote access
Bright When Dark And/Or Bright After Sunset	switchLevel		Yes
Camera Power Scheduler	switch		Yes
Curling Iron		motionSensor	Yes
Forgiving Security	contactSensor, switch	alarm, motionSensor	Yes
Good Night		switch	Yes
Jenkins Notifier	colorControl	switch	Yes
Notify Me When	button, contactSensor, accelerationSensor, presenceSensor, smokeDetector, waterSensor	motionSensor, switch	Yes
Photo Burst When	accelerationSensor, contactSensor	imageCapture, motionSensor, switch, presenceSensor	Yes
Prempoint		imageCapture, switch, lock, garageDoorControl	Yes
Rise and Shine		motionSensor	Yes
Safe Watch	contactSensor, accelerationSensor, threeAxis, temperatureMeasurement	motionSensor, presenceSensor	Yes
Send HAM Bridge Command When	contactSensor, accelerationSensor, switch, waterSensor, smokeDetector	motionSensor, presenceSensor	Yes
Simple Control	switch, lock, thermostat, doorControl, colorControl, musicPlayer, switchLevel	lock, doorControl	Not if block remote access
Smart Light Timer	contactSensor	motionSensor	Yes
Smart Security		switch	Yes
Smart Windows	contactSensor		Yes
SmartBlock Notifier		switch	Yes
Speaker Control	contactSensor, accelerationSensor, switch, waterSensor, button	motionSensor, presenceSensor	Yes
Speaker Mood Music	contactSensor, accelerationSensor, button, waterSensor,musicPlayer	motionSensor, presenceSensor, switch	Yes
Sprayer Controller 2		switch	Yes
Spruce Scheduler	contactSensor		Yes
Talking Alarm Clock	switchLevel, temperatureMeasurement, thermostat, relativeHumidityMeasurement		Yes
The Flasher		presenceSensor	Yes
Turn It On For 5 Minutes	contactSensor		Yes
Undead Early Warning	contactSensor	switch	Yes
Vinli Home Connect		switch,lock	Not if block remote access
Virtual Thermostat		motionSensor	Yes
Weather Windows	contactSensor		Yes
Whole House Fan	contactSensor		Yes

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Rich user information enables platforms to provide various useful services. However, protecting the security and privacy of user data, especially when functionality conflicts with security and privacy, is a challenging problem. We observe that current practices do not adequately solve this conflict; platforms either share information for functionality but violate privacy or sacrifice functionality for privacy concerns. I propose design principles that enable functionality and preserve privacy. Specifically, I use social platforms and smart home platforms as case studies to demonstrate and implement these design principles.

I identify special challenges for privacy-preserving data sharing and propose solutions to these challenges for use in different platforms. For the social platforms, I perform a multi-faceted study of permission models, aiming to design models that provide rich functionality, privacy preservation, and usability. I evaluate the use of opaque handles, opaque display, and user-driven access control primitives toward maximizing functionality and usability without sacrificing privacy. I evaluate the feasibility and efficiency of OOU primitives on Facebook and find that for 87 primitives that incorporate OOU, 95 out of 117 sensitive permission instances can be removed without affecting functionality. I further design a permission model for the next-generation Somex social platform using the OOU primitives. My design enables useful functionality, such as context-aware services, by utilizing contexts from different applications and users without violating privacy. We note that the OOU primitives cannot address cases that involve computation of the data or cases in which the data access cannot be made user-driven. I believe that our demonstration of the value of platform-based mediation of sensitive data will motivate a wealth of future work in this area. For example, a social platform can analyze common data usage patterns and develop customized opaque services to further improve the privacy-preserving sharing of sensitive data. For the smart home platform, I have identified the fundamental gap between how users expect an IoT app to perform and what really takes place. I rethink the notion of authorization in IoT platforms and propose an automated and usable solution called SmartAuth to bridge the gap. SmartAuth automatically collects security-relevant information from an IoT app's code and description using a combination of program analysis and natural language processing and generates a user-friendly authorization interface to inform the user of the unexpected app behaviors. Through manual verification and in-lab human subject studies, I demonstrate that SmartAuth enables users to make better-informed authorization decisions for IoT apps compared to the current approach.

4.2 Future Research Directions

The work presented in this study falls into the broad areas of identifying privacy–functionality conflicts and designing balanced systems. As the platforms and services keep evolving, more problems will appear.

4.2.1 Reliable, Secure and Usable Systems For Emerging Platforms

While working on the mobile OAuth project, I observed that in the computing ecosystem with multiple players, securely integrating the different components was very challenging. For example, users were using devices (e.g., smart vehicles, smart watches, drones, and smart locks) and services (e.g., payment services and third-party apps) from different providers daily. We need to understand the security implications of each component and their combinations to build secure systems. We are facing serious problems, such as how to verify the security behaviors of automated cars and how to manage security and privacy for smart cities. I plan to model the security risks for the emerging platforms formally, and then I will design secure systems that provide the latest techniques and innovations. I will use my previous experiences in security analysis to understand security risks and design systems that consider both security and functionality. Additionally, I plan to utilize hardware security features to dynamically audit the platform and discover security risks. In addition to building the system, we need to consider the human factors when achieving security and privacy benefits. We must examine how to minimize the security and privacy risks in the most user-friendly way, given that minimizing risks through activities like managing sensors at home can create many burdens for users. While working on the smart home permission design project, I found that people tend to manage devices according to the context. We want to build an access control system that organizes the devices via context-based approaches, such as location, time, and user interaction.

4.2.2 Privacy-Preserving Machine Learning

Machine learning and big data analytics boost the success of many services, such as self-driving cars, automatic personal assistants, and health-care informatics. While working on the Facebook application analysis, I observed huge amounts of data collection targeted for machine learning applications. I plan to build a data sharing and machine learning scheme that processes data in a way that preserves privacy and includes hardware security guarantees. Specifically, I have started a project that distributes deep neural networks into sets of executing layers and optimizes the model against an adversary with access to the parameters to infer the neural networks' plaintext inputs.

In addition to exploring the machine learning algorithms, I plan to build a platform that supports such privacy-preserving machine learning applications. I select the IoT platform for designing the platform support because of the large amount of sensor data generated and collected in the IoT platform. We want to explore how the fog mediation model can clarify issues relating to data ownership and data sharing. I plan to address these issues in two stages. First, we will evaluate approaches for analyzing IoT apps in order to align their privacy-sensitive operations (e.g., read sensor data and actuate device) with the minimal subset of privileged permissions required to support a given use case. Secondly, I plan to design different modes of secure data sharing in the fog-computing context that balance stakeholders' data-sharing requirements with application requirements, resulting in a service manifest that may be independently attested to by each stakeholder.

References

[1] J. Constine, *Facebook is shutting down its API for giving your friends' data to Apps*, https://techcrunch.com/2015/04/28/facebook-api-shut-down/, 2015.

[2] D. Bolles, *"JOBS WITH FRIENDS" is shutting down*, http://www.jobhuntersbible.com/for-readers-of-parachute/view/jobs-with-friends-is-shutting-down, 2015.

[3] J. Grossman, *Breaking browsers: Hacking auto-complete*, http://jeremiahgrossman.blogspot.com/2010/08/breaking-browsers-hacking-auto-complete.html, 2010.

[4] Apple Mailing List, *APPLE-SA-2010-07-28-1 Safari 5.0.1 and Safari 4.1.1*, http://lists.apple.com/archives/security-announce/2010//Jul/msg00001.html, 2010.

[5] Bugs@Mozilla, *Bug 527935 - (CVE-2011-0067) untrusted events should not trigger autocomplete popup*, https://bugzilla.mozilla.org/show_bug.cgi?id=527935, 2009.

[6] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery", in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008.

[7] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting frame busting: A study of clickjacking vulnerabilities on popular sites", in *Web 2.0 Security and Privacy*, 2010.

[8] Dakota Shane, *What Medium (and all social media platforms) can learn from YouTube's aweinspiring success*, https://medium.com/the-mission/what-medium-and-all-social-media-platforms-can-learn-from-youtubes-awe-inspiring-success-73021179b605, 2017.

[9] Facebook, *Login review*, https://developers.facebook.com/docs/facebook-login/review, 2016.

[10] Google, Inc., *Incremental authorization*, https://developers.google.com/+/web/api/rest/oauth#incremental-auth, 2017.

[11] Facebook, *Permissions reference - Facebook login*, https://developers.facebook.com/docs/facebook-login/permissions/, 2017.

[12] Twitter, *Application permission model*, `https://dev.twitter.com/oauth/overview/application-permission-model`, 2017.

[13] Facebook, *Login review*, `https://developers.facebook.com/docs/facebook-login/review`, 2017.

[14] Instagram, *Permissions review*, `https://www.instagram.com/developer/review/`, 2017.

[15] Google, *Change your Google+ Sign-In settings*, `https://support.google.com/plus/answer/2980770`, 2016.

[16] Facebook, *Requesting and revoking permissions*, `https://developers.facebook.com/docs/facebook-login/permissions/requesting-and-revoking`, 2016.

[17] ——, *Facebook platform changelog*, `https://developers.facebook.com/docs/apps/changelog`, 2016.

[18] Stackoverflow, *Facebook graph API v2.0+ /me/friends returns empty*, `http://goo.gl/f6QDyX`, 2015.

[19] Torbjorn Nilsen, *Petitioning Facebook:change the way permissions work in Facebook API*, `https://www.change.org/p/facebook-change-the-way-permissions-work-in-facebook-api`, 2015.

[20] E. Vela, *CSS attribute reader proof of concept*, `http://eaea.sirdarckcat.net/cssar/v2/`, 2011.

[21] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: Stealing the pie without touching the sill", in *Proceedings of the 2012 ACM conference on Computer and Communications Security*, ACM, 2012, pp. 760–771.

[22] T. Oda, G. Wurster, P. C. van Oorschot, and A. Somayaji, "SOMA: Mutual approval for included content in web pages", in *Proceedings of the 15th ACM conference on Computer and Communications Security*, 2008.

[23] E. Y. Chen, J. Bau, C. Reis, A. Barth, and C. Jackson, "App isolation: Get the security of multiple browsers with just one", in *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011, pp. 227–238.

[24] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The multi-principal OS construction of the gazelle web browser", in *Proceedings of 18th USENIX Security Symposium*, 2009, pp. 417–432.

[25] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy, "A safety-oriented platform for web applications", in *Proceedings of 27th IEEE Symposium on Security and Privacy*, IEEE, 2006, 15–pp.

[26] Z. Mao, N. Li, and I. Molloy, "Defeating cross-site request forgery attacks with browser-enforced authenticity protection", in *Financial Cryptography and Data Security*, Springer, 2009, pp. 238–255.

[27] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy", in *Proceedings of 8th Symposium on Usable Privacy and Security*, 2012.

[28] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior", in *Symposium on Usable Privacy and Security*, ACM, 2012, pp. 3–17.

[29] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2013, pp. 3393–3402.

[30] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified", in *Proceedings of the 18th ACM conference on Computer and Communications Security*, ACM, 2011, pp. 627–638.

[31] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem", in *Proceedings of the 28th Annual Computer Security Applications Conference*, ACM, 2012, pp. 31–40.

[32] J. Jung, S. Han, and D. Wetherall, "Short paper: Enhancing mobile application permissions with runtime feedback and constraints", in *ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ACM, 2012, pp. 45–50.

[33] S. Motiee, K. Hawkey, and K. Beznosov, "Do Windows users follow the principle of least privilege?: investigating user account control practices", in *Proceedings of the 6th Symposium on Usable Privacy and Security*, ACM, 2010, p. 1.

[34] K.-P. Yee, "Aligning security and usability", *IEEE Security & Privacy*, vol. 2, no. 5, pp. 48–55, 2004.

[35] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission", in *USENIX Conference on Hot Topics in Security*, 2012.

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

[37] A. L. Young and A. Quan-Haase, "Information revelation and internet privacy concerns on social network sites: A case study of Facebook", in *Proc. International Conference on Communities and technologies*, ACM, 2009, pp. 265–274.

[38] Y. Liu, K. P. Gummadi, B. Krishnamurthy, and A. Mislove, "Analyzing Facebook privacy settings: user expectations vs. reality", in *ACM SIGCOMM Conference on Internet Measurement Conference*, ACM, 2011, pp. 61–70.

[39] N. Wang, H. Xu, and J. Grossklags, "Third-party apps on Facebook: privacy and the illusion of control", in *ACM Symposium on Computer Human Interaction for Management of Information Technology*, ACM, 2011, p. 4.

[40] R. Gross and A. Acquisti, "Information revelation and privacy in online social networks", in *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, ACM, 2005, pp. 71–80.

[41] P. Wisniewski, H. Xu, H. Lipford, and E. Bello-Ogunu, "Facebook apps and tagging: The trade-off between personal privacy and engaging with friends", *Journal of the Association for Information Science and Technology*, vol. 66, no. 9, pp. 1883–1896, 2015.

[42] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems", in *Proceedings of the 33th IEEE Symposium on Security and Privacy*, 2012.

[43] H. J. Wang, A. Moshchuk, M. Gamon, S. Iqbal, E. T. Brown, A. Kapoor, C. Meek, E. Chen, Y. Tian, J. Teevan, M. Czerwinski, and S. Dumais, "The activity platform", in *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, Kartause Ittingen, Switzerland: USENIX Association, May 2015. [Online]. Available: `http://blogs.usenix.org/conference/hotos15/workshop-program/presentation/wang`.

[44] W3C, *Frames*, `http://www.w3.org/TR/html401/present/frames.html`, 2014.

[45] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond.", in *Proceedings of 22nd USENIX Security Symposium*, 2013, pp. 97–112.

[46] Facebook, *Comments plugin*, `https://developers.facebook.com/docs/plugins/comments/`, 2016.

[47] D. Chereches, *This project was discontinued*, `https://github.com/loadrunner/Facebook-Contact-Sync`, 2017.

[48] Facebook, *Creating action types*, `https://developers.facebook.com/docs/opengraph/creating-custom-stories/#actiontypes`, 2014.

[49] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2010, pp. 24–43.

[50] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, *Machine learning classification over encrypted data*, Cryptology ePrint Archive, Report 2014/331, `http://eprint.iacr.org/2014/331`, 2014.

[51] SmartThings, Inc., *Samsung SmarThings*, `https://www.smartthings.com/`, 2016.

[52] Google, Inc., *Weave*, `https://developers.google.com/weave/`, 2016.

[53] ——, *Brillo*, `https://developers.google.com/brillo/`, 2016.

[54] Apple, Inc., *Apple HomeKit*, `http://www.apple.com/ios/home/`, 2016.

[55] SmartThings, Inc., *SmartThings marketplace*, `https://support.smartthings.com/hc/en-us/articles/205379924-Marketplace`, 2016.

[56] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications", in *Proceedings of the 37th IEEE Symposium on Security and Privacy*, 2016.

[57] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: Installing applications on an Android smartphone", in *International Conference on Financial Cryptography and Data Security*, 2012.

[58] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process", in *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2013.

[59] H. Nissenbaum, “Privacy as contextual integrity”, *Wash. L. Rev.*, vol. 79, p. 119, 2004.

[60] SmartThings, Inc., *SmarThings code review guidelines*, `http://docs.smartthings.com/en/latest/code-review-guidelines.html`, 2017.

[61] Juniper Research, *Smarthome revenues to reach $100 Billion by 2020*, `https://www.juniperresearch.com/press/press-releases/smart-home-revenues-to-reach-$100-billion-by-2020`, 2008–2016.

[62] Vera Ltd., *Vera: Smarter home control*, `http://getvera.com/`, 2008–2016.

[63] H. M. Levy, *Capability-based computer systems*. Digital Press, 2014.

[64] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, “Appcontext: Differentiating malicious and benign mobile app behaviors using context”, in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, IEEE, vol. 1, 2015, pp. 303–313.

[65] M. Naveed, X.-y. Zhou, S. Demetriou, X. Wang, and C. A. Gunter, “Inside job: Understanding and mitigating the threat of external device mis-binding on Android”, in *Proceedings of the Network and Distributed System Security Symposium (NDSS’14)*, 2014.

[66] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, “Proposed embedded security framework for internet of things (IoT)”, in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, IEEE, 2011, pp. 1–5.

[67] Y. Goldberg and O. Levy, “Word2vec explained: Deriving Mikolov et al.’s negative-sampling word-embedding method”, *arXiv preprint arXiv:1402.3722*, 2014.

[68] G. A. Miller, “Wordnet: A lexical database for english”, *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[69] E. Gabrilovich and S. Markovitch, “Computing semantic relatedness using wikipedia-based explicit semantic analysis.”, in *International Joint Conference on Artifical Intelligence*, 2007, pp. 1606–1611.

[70] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, “The stanford corenlp natural language processing toolkit.”, in *ACL (System Demonstrations)*, 2014, pp. 55–60.

[71] S. N. Group, *The Stanford Parser: A statistical parser*, `http://nlp.stanford.edu/software/lex-parser.shtml`, 2002.

[72] SmartThings, Inc., *SmarThings public*, `https://github.com/SmartThingsCommunity/SmartThingsPublic`, 2016.

[73] C. D. Manning, *Dropping common terms: stop words*, `http://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html`, 2016.

[74] Google, Inc., *Google News Vectors*, `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS`
2016.

[75] SimpleNLG, *SimpleNLG*, `https://github.com/simplenlg/simplenlg`, 2016.

[76] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing", in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ACM, 2012, pp. 501–510.

[77] A. P. Felt, S. Egelman, and D. Wagner, "I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns", in *Proceedings of the second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ACM, 2012, pp. 33–44.

[78] M. Tiwari, P. Mohan, A. Osheroff, H. Alkaff, E. Shi, E. Love, D. Song, and K. Asanović, "Context-centric security", in *USENIX Conference on Hot Topics in Security*, USENIX Association, 2012, pp. 9–9.

[79] B. Liu, J. Lin, and N. Sadeh, "Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help?", in *Proceedings of the 23rd International Conference on World Wide Web*, ACM, 2014, pp. 201–212.

[80] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android permission model and enforcement with user-defined runtime constraints", in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, 2010, pp. 328–332.

[81] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, "Your location has been shared 5,398 times!: A field study on mobile app privacy nudging", in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, ACM, 2015, pp. 787–796.

[82] A. Felt and D. Evans, "Privacy protection for social networking APIs", *Web 2.0 Security and Privacy*, 2008.

[83] W. Luo, Q. Xie, and U. Hengartner, "Facecloak: An architecture for user privacy on social networking sites", in *International Conference on Computational Science and Engineering*, IEEE, vol. 3, 2009, pp. 26–33.

[84] K. Singh, S. Bhola, and W. Lee, "xBook: Redesigning Privacy Control in Social Networking Platforms", in *Proceedings of 18th USENIX Security Symposium*, 2009, pp. 249–266.

[85] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users", in *Proceedings of the 31st IEEE Symposium on Security and Privacy*, IEEE, 2010, pp. 223–238.

[86] A. Narayanan and V. Shmatikov, "De-anonymizing social networks", in *Proceedings of the 30th IEEE Symposium on Security and Privacy*, IEEE, 2009, pp. 173–187.

[87] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: An online social network with user-defined privacy", in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 39, 2009, pp. 135–146.

[88] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust", *IEEE Communications Magazine*, vol. 47, no. 12, 2009.

[89] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “Whyper: Towards automating risk assessment of mobile applications”, in *Proceedings of the 22nd USENIX Security Symposium*, 2013, pp. 527–542.

[90] M. Zhang, Y. Duan, Q. Feng, and H. Yin, “Towards automatic generation of security-centric descriptions for Android apps”, in *ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 518–529.

[91] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, “Autocog: Measuring the description-to-permission fidelity in Android applications”, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 1354–1365.

[92] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, “AsDroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction”, in *International Conference on Software Engineering*, ACM, 2014, pp. 1036–1046.

[93] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, “Checking app behavior against app descriptions”, in *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 1025–1035.

[94] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. Bellovin, and J. Reidenberg, “Automated analysis of privacy requirements for mobile apps”, in *Proceedings of the Network and Distributed System Security Symposium (NDSS’2017)*, 2017.

[95] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner, “Are you ready to lock?”, in *ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2014, pp. 750–761.

[96] M. Harbach, M. Hettig, S. Weber, and M. Smith, “Using personal examples to improve risk communication for security and privacy decisions”, in *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[97] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner, “The effect of developer-specified explanations for permission requests on smartphone user behavior”, in *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[98] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, “Android permissions remystified: A field study on contextual integrity”, in *Proceedings of 24th USENIX Security Symposium*, 2015.

[99] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, “Flowfence: Practical data protection for emerging iot application frameworks”, in *Proceedings of the 25th USENIX Security Symposium*, 2016.

[100] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, “ContexIoT: Towards providing contextual integrity to appified IoT platforms”, 2017.

[101] K. Fawaz, K.-H. Kim, and K. G. Shin, “Protecting privacy of BLE device users”, in *Proceedings of 25th USENIX Security Symposium*, USENIX Association, 2016.

[102] K.-T. Cho and K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection”, in *Proceedings of 25th USENIX Security Symposium*, USENIX Association, 2016.

[103] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, “Network-level security and privacy control for smart-home IoT devices”, in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, IEEE, 2015, pp. 163–167.

[104] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, “Seda: Scalable embedded device attestation”, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 964–975.

[105] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, “Tytan: Tiny trust anchor for tiny devices”, in *Proceedings of ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2015, pp. 1–6.

[106] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, “C-flat: Control-flow attestation for embedded systems software”, *arXiv preprint arXiv:1605.07763*, 2016.

[107] S. Egelman, R. Kannavara, and R. Chow, “Is this thing on?: Crowdsourcing privacy indicators for ubiquitous sensing platforms”, in *ACM SIGCHI Conference on Human Factors in Computing Systemss*, ACM, 2015, pp. 1669–1678.

[108] B. Ur, J. Jung, and S. Schechter, “Intruders versus intrusiveness: Teens’ and parents’ perspectives on home-entryway surveillance”, in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ACM, 2014, pp. 129–139.

[109] G. Demiris, “Privacy and social implications of distinct sensing approaches to implementing smart homes for older adults”, in *Annual International Conference on Engineering in Medicine and Biology Society*, IEEE, 2009, pp. 4311–4314.

A Supplemental Materials

A.1 Supplemental Materials for the Study on Facebook Permission

A.1.1 Facebook Login Permission Details

In table A.1, I provide the information about Facebook login permissions. Specifically, I provide the information whether the permissions enable an app to access content in the future, and whether the app needs to be reviewed to use the permissions [13]. To determine the future content access, I go through the Facebook permission documents [11] to understand what an app can do with a permission.

A.1.2 Survey for Users' Mental Model about Facebook Permissions

We ran user studies for users' expectations about Facebook Permissions. We post surveys about Facebook permissions with a title of "Facebook user experience survey" in Mturk. We told users that we would ask their experiences about using Facebook. In the survey, we asked the participants to answer our questions regarding their understanding of Facebook permissions. For example, we asked them questions about what they think certain permissions will allow the app to do. The survey took approximately 20 minutes, and the results were collected via Mturk. Participants must be older than 18, be literate in English, and use a Facebook account frequently. Each participant was paid 0.5 dollar for the survey. The study was done in October 2014. The recruitment message was as follows:

We are a group of students in Carnegie Mellon University, doing research about Facebook permissions. If you have used Facebook for at least 1 month and shared content on Facebook before and you are an adult(18 years or older) living in the US, you would be very helpful to our study. The survey takes about 30 minutes to finish. We will not collect any identification data from you during the survey. Thanks!

Please read the consent form [Link to the consent form]

Please follow this link to the survey (Please only continue if you read and agree to the consent form). [Link to the survey]

If a user agrees to do the survey, we will instruct the user to click on the link to the survey hosted on Instant.ly to finish the survey to get a code, and then come back to Mturk to enter the code.

We attach the survey on the following pages.

Page 1

Q1: DESCRIPTIVE TEXT

Attention: Please make sure that you are qualified for this survey:
* You must be a Facebook user.
* You must have used Facebook for at least 1 month and shared content on Facebook.
* Your must be an adult, live in the USA and literate in English.

Q2: SINGLE-SELECT LIST

1.Do you have a Facebook account?

1 ◯ Yes
2 ◯ No

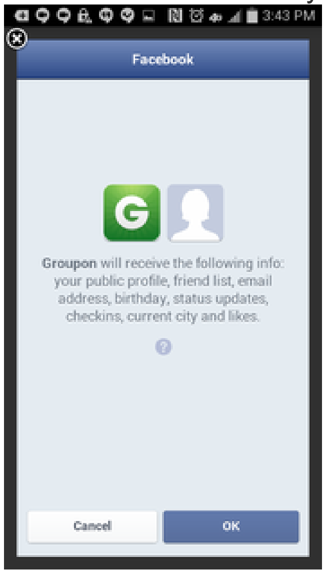
Q3: SINGLE-SELECT LIST

2. How long approximately have you been using Facebook?

1 ◯ 1-6 months
2 ◯ 6-12 months
3 ◯ 1-3 years
4 ◯ 3 and more years

Q4: SINGLE-SELECT LIST

3. Groupon is an app for searching deals. The following figure shows that Groupon asks to access your status when you log into Groupon with Facebook account. Do you expect Groupon to read your status anytime in the future?



Groupon will receive the following info: your public profile, friend list, email address, birthday, status updates, checkins, current city and likes.

1 ○ Yes
2 ○ No
3 ○ I don't know

Q5: SINGLE-SELECT MATRIX

4. Groupon can read all the posts of your Facebook status in the future, including all the content posted by your friends or shared by other applications, how comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q6: SINGLE-SELECT LIST

5. Pandora is a music app. The following figure shows that Pandora asks to access the pages you liked when you log into Pandora with Facebook account. Do you expect Groupon to read your likes anytime in the future?

1 ○ Yes
2 ○ No
3 ○ I don't know

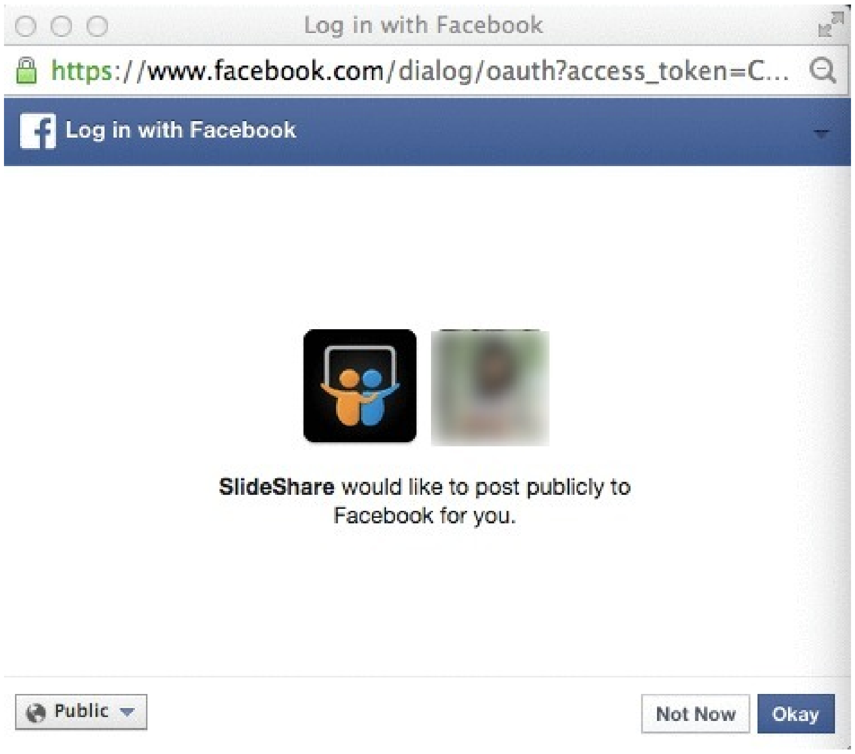
Q7: SINGLE-SELECT MATRIX

6. Pandora can read all the likes in the future, how comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q8: SINGLE-SELECT LIST

7. Some apps will ask your permission to post to your timeline when you log into the apps with Facebook. The following figure shows SlideShare asking your approval to post to your timeline, who do you think the post will be visible to?


Log in with Facebook

https://www.facebook.com/dialog/oauth?access_token=C...

Log in with Facebook

SlideShare would like to post publicly to Facebook for you.

1 ◯ Public
2 ◯ All the Facebook users
3 ◯ All your Facebook friends
4 ◯ Your Facebook friends who use the same app
5 ◯ Some group of your Facebook friends
6 ◯ Only yourself

Q9: SINGLE-SELECT LIST

8. If you log into SlideShare with Facebook, do you expect SlideShare to post to your timeline without your consensus?

1 ◯ Yes
2 ◯ No
3 ◯ I don't know

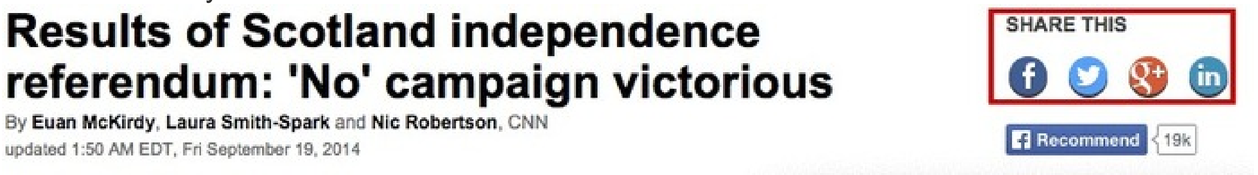
Q10: SINGLE-SELECT MATRIX

9. SlideShare could post to your timeline without your consensus after you log into SlideShare with Facebook. How comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	◯	◯	◯	◯	◯	◯

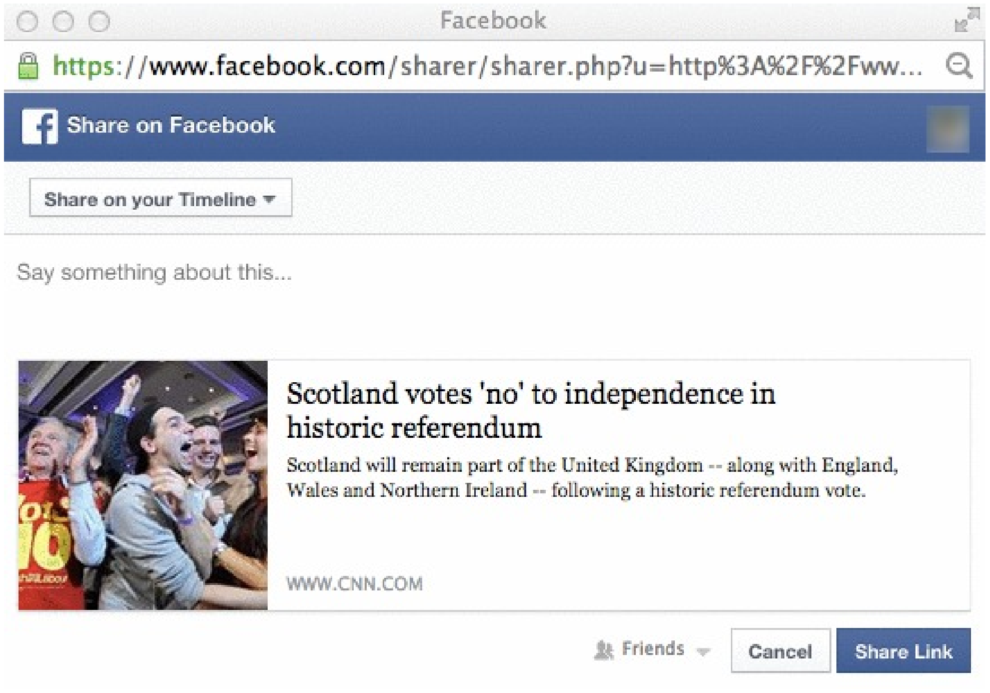
Q11: SINGLE-SELECT LIST

10. Some apps and websites provide a share button where you can choose what content you want to share. The first figure demonstrates the share button on CNN.com and the second figure shows that you are asked to confirm sharing the news after clicking the share button. Who do you think the news will be shared to?



Results of Scotland independence referendum: 'No' campaign victorious

By Euan McKirdy, Laura Smith-Spark and Nic Robertson, CNN
updated 1:50 AM EDT, Fri September 19, 2014



Facebook
https://www.facebook.com/sharer/sharer.php?u=http%3A%2F%2Fww...
Share on Facebook
Share on your Timeline
Say something about this...
Friends
Cancel
Share Link

Scotland votes 'no' to independence in historic referendum

Scotland will remain part of the United Kingdom -- along with England, Wales and Northern Ireland -- following a historic referendum vote.

WWW.CNN.COM

1 ○ Public
2 ○ All the Facebook users
3 ○ All your Facebook friends
4 ○ Your Facebook friends who use the same app
5 ○ Some group of your Facebook friends
6 ○ Only yourself

Q12: SINGLE-SELECT MATRIX

11. CNN will not share anything to your timeline unless you click the share button. How comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

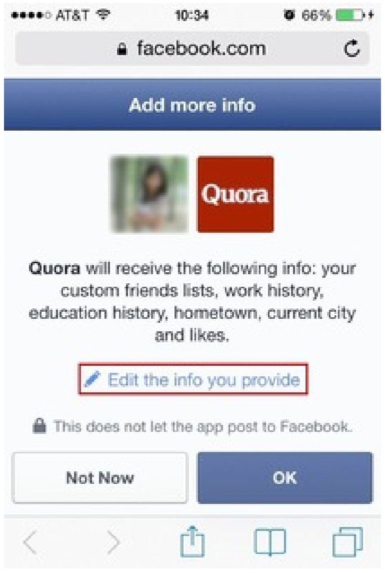
Q13: SINGLE-SELECT LIST

12. Which way of posting to your timeline do you prefer?
SlideShare: ask only at the first time you log in with Facebook and share content in the future without your confirmation
CNN: you can choose what to share anytime and confirm before sharing

1 ◯ SlideShare
2 ◯ CNN
3 ◯ I don't like either

Q14: SINGLE-SELECT LIST

13. When some apps ask your permission to read your Facebook content, they also provide you a way to edit what information you want to share. As is shown in the figure, you can edit whether to share friend list, etc to the app. Will you spend time editing the content?



Quora will receive the following info: your custom friends lists, work history, education history, hometown, current city and likes.

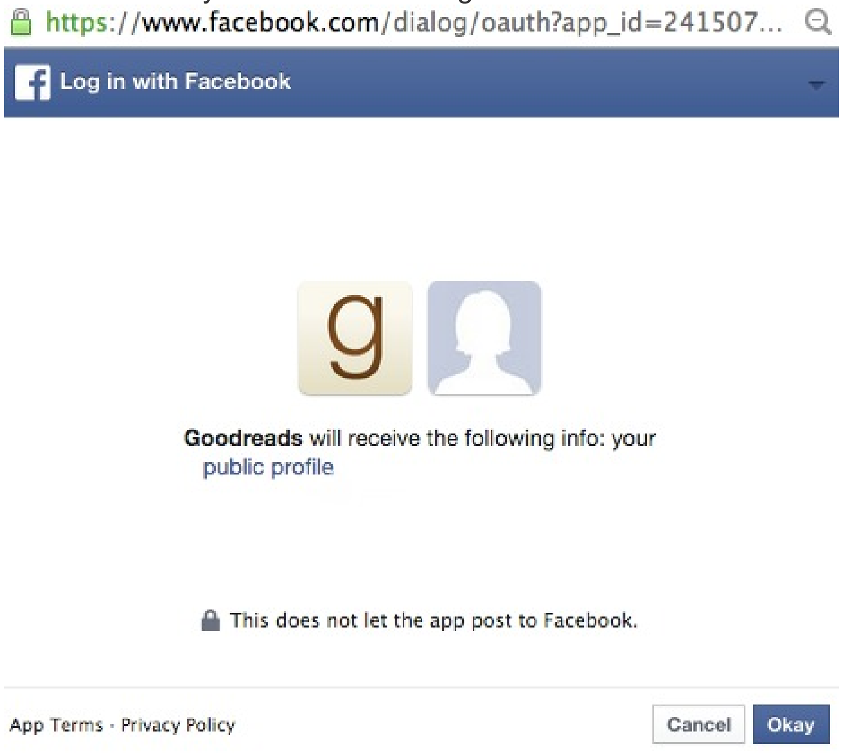
This does not let the app post to Facebook.

1 ◯ Yes
2 ◯ No
3 ◯ Sometimes

Q15: SINGLE-SELECT MATRIX

14. GoodRead is an app which provide book reviews, recommendations and discussions. If you log into GoodRead with Facebook and you see a prompt as illustrated below, how

comfortable do you feel with allowing GoodRead to access these information?


https://www.facebook.com/dialog/oauth?app_id=241507...

Log in with Facebook

Goodreads will receive the following info: your public profile

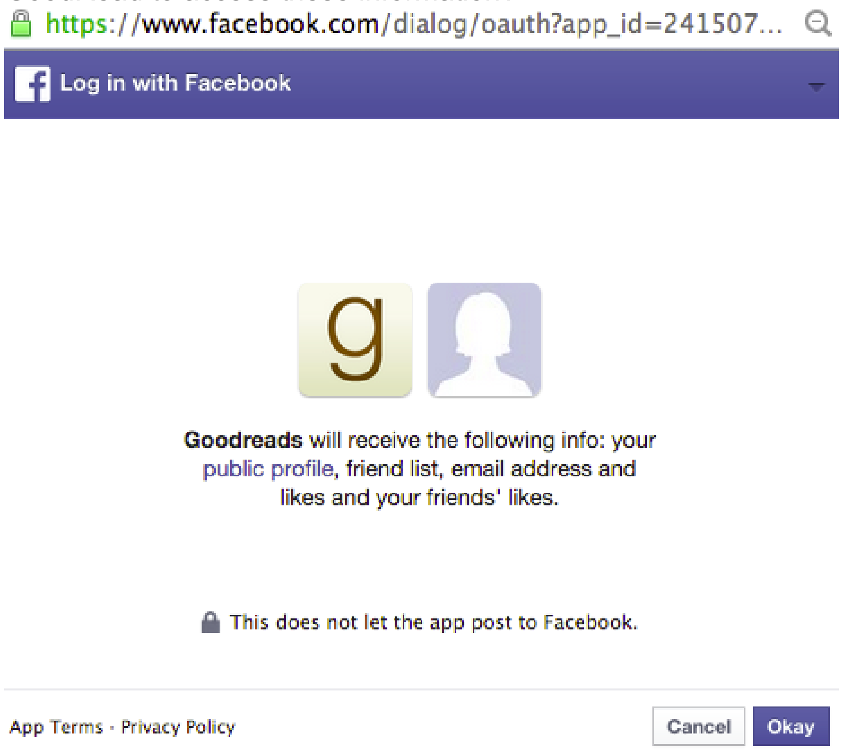
This does not let the app post to Facebook.

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q16: SINGLE-SELECT MATRIX

15. If you log into GoodRead with Facebook and you see a prompt as illustrated below (more permissions than the previous prompt), how comfortable do you feel with allowing

GoodRead to access these information?



https://www.facebook.com/dialog/oauth?app_id=241507...

Log in with Facebook

Goodreads will receive the following info: your public profile, friend list, email address and likes and your friends' likes.

This does not let the app post to Facebook.

App Terms · Privacy Policy

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q17: SINGLE-SELECT LIST

16. Do you expect an application/web site (e.g., TripAdvisor) to be able to access the content shared by another application/website (e.g., Yelp)?

1 ○ Yes
2 ○ No
3 ○ I don't know

Q18: SINGLE-SELECT MATRIX

17. If TripAdvisor can only access **all the content from Yelp** through Facebook, no matter what the content is about(could be very sensitive) , how comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q19: SINGLE-SELECT MATRIX

18. If TripAdvisor can **only access content about the same trip** from Yelp through Facebook, TripAdvisor will not access extra information. How comfortable do you feel with it?

	Very comfortable	Somewhat comfortable	Neutral	Somewhat uncomfortable	Very uncomfortable	I don't know
Comfort level	○	○	○	○	○	○

Q20: SINGLE LINE TEXT

19. What's your age?

Q21: SINGLE-SELECT LIST

20. What is your gender?

1 ○ Female
2 ○ Male
3 ○ Decline to answer

Q22: SINGLE-SELECT LIST

21. What is your highest education level of education you have completed?

1 ○ No high school
2 ○ Some high school
3 ○ High school graduate
4 ○ Some college - No degree
5 ○ Associates / 2 year degree
6 ○ Bachelor / 4 year degree
7 ○ Graduate degree - Master, PhD, professional, medicine, etc

Q23: DROP DOWN LIST

22. Which of the following best describes your primary occupation(Drop-down list)?

1 Administrative support (e.g., secretary, assistant)
2 Art, writing, or journalism (e.g., author, reporter, sculptor)
3 Business, management, or financial (e.g., manager, accountant, banker)
4 Computer engineer or IT professional (e.g., systems administrator, programmer, IT consultant)
5 Education (e.g., teacher)
6 Engineer in other fields (e.g., civil engineer, bio-engineer)
7 Homemaker

8 Legal (e.g., lawyer, law clerk)
9 Medical (e.g., doctor, nurse, dentist)
10 Retired
11 Scientist (e.g., researcher, professor)
12 Service (e.g., retail clerks, server)
13 Skilled labor (e.g., electrician, plumber, carpenter)
14 Student
15 Unemployed
16 Decline to answer
17 Other

Table A.1: Facebook login permission details

Permission	Access Future Content?	Need Review?	Permission	Access Future Content?	Need Review?
public_profile	No	No	user_religion_politics	Yes	Yes
user_friends	Yes	No	user_status	Yes	Yes
email	No	No	user_tagged_places	Yes	Yes
user_about_me	No	Yes	user_videos	Yes	Yes
user_acitivites	Yes	Yes	user_website	No	Yes
user_actions.books	Yes	Yes	user_work_history	No	Yes
user_actions.fitness	Yes	Yes	read_friendlists	No	Yes
user_actions.music	Yes	Yes	read_insights	Yes	Yes
user_actions.news	Yes	Yes	read_audience_network_ingights	Yes	Yes
user_actions.video	Yes	Yes	read_page_mailbox	Yes	Yes
user_actions: (app_namespace)	Yes	Yes	read_mailboxes	Yes	Yes
user_birthday	No	Yes	read_stream	Yes	Yes
user_education_history	No	Yes	manage_pages	Yes	Yes
user_events	Yes	Yes	manage_notifications	Yes	Yes
user_groups	Yes	Yes	publish_actions	Yes	Yes
user_games_activity	Yes	Yes	rsvp_event	Yes	Yes
user_hometown	No	Yes	pages_manage_instant_articles	No	Yes
user_interests	No	Yes	ads_read	Yes	Yes
user_likes	Yes	Yes	ads_management	Yes	Yes
user_location	Yes	Yes	business_management	Yes	Yes
user_managed_groups	Yes	Yes	pages_messaging	Yes	Yes
user_photos	Yes	Yes	pages_messaging_payments	Yes	Yes
user_relationships	Yes	Yes	pages_messaging_phone_number	Yes	Yes
user_relationship_details	Yes	Yes			

A.2 Supplemental Materials for the Study on Smarthome Authorization Systems

A.2.1 Survey for Users' Mental Model about Smarthome Permission

In July 2016, we post surveys with the title "Smarthome Survey" on Mturk. The recruitment message is as follows:

We are a group of students in Carnegie Mellon University, doing research about Smarthome. If you have used Smarthome products or have knowledge about how Smarthome works and you are an adult(18 years or older) living in the US, you would be very helpful to our study. The survey takes about 30 minutes to finish. We will not collect any identification data from you during the survey. Thanks!

Please read the consent form [Link to the consent form]

Please follow this link to the survey (Please only continue if you read and agree to the consent form). [Link to the survey]

If a user agrees to do the survey, we will instruct the user to click on the link to the survey hosted on Instant.ly to finish the survey to get a code, and then come back to Mturk to enter the code.

In the following, I attach a copy of the survey that we distributed to evaluate users' expectations for Smarthome permissions.

Q1: SINGLE-SELECT LIST

Do you own a smartphone (phone / tablet)?

1 ○ Yes
2 ○ No

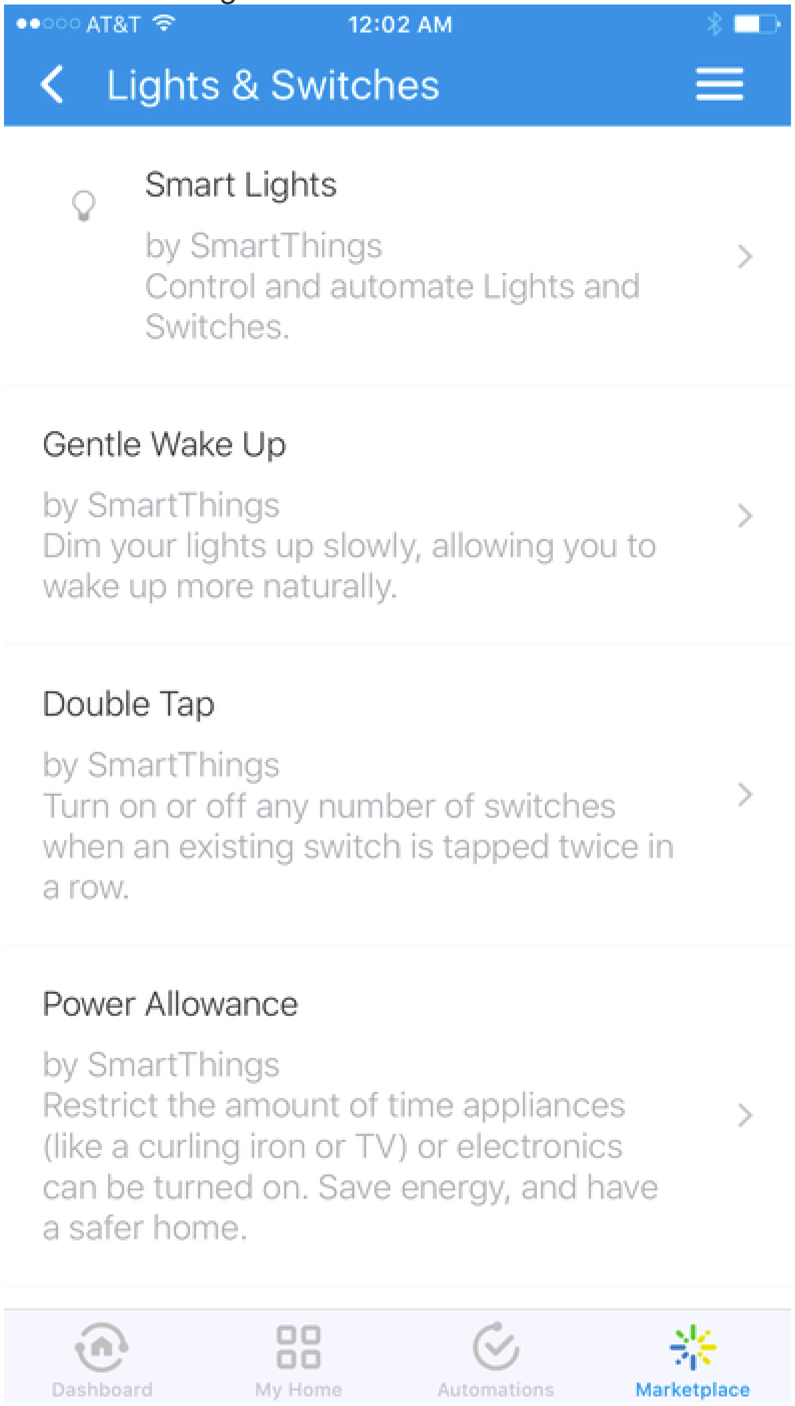
Q2: SINGLE-SELECT LIST

Do you know how to use a smartphone to control smart devices?

1 ○ Yes
2 ○ No

Q3: SINGLE-SELECT LIST

Smartthing is a mobile app that allows you to install third-party apps to manage your smart devices at home. For example, you can install an app "Keep me cozy" to control your air conditioner according to the temperature. Have you used Smartthing before?


●●○○○ AT&T 12:02 AM

Lights & Switches

Smart Lights
by SmartThings
Control and automate Lights and Switches.

Gentle Wake Up
by SmartThings
Dim your lights up slowly, allowing you to wake up more naturally.

Double Tap
by SmartThings
Turn on or off any number of switches when an existing switch is tapped twice in a row.

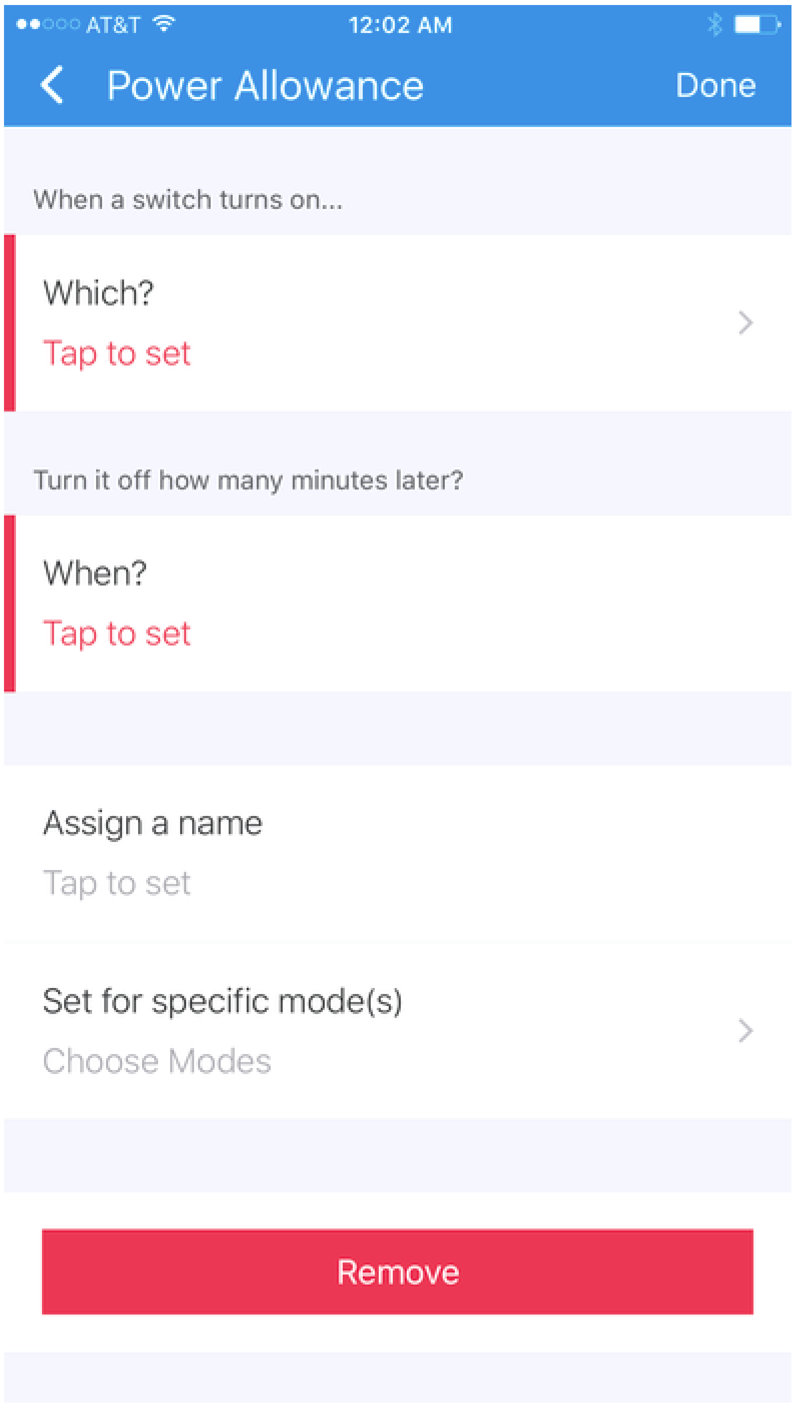
Power Allowance
by SmartThings
Restrict the amount of time appliances (like a curling iron or TV) or electronics can be turned on. Save energy, and have a safer home.

1 ○ Yes
2 ○ No

Page 2

Q4: DESCRIPTIVE TEXT

Third-party app you install with Smartthing will request to access some of your devices. Please read the UI to answer the following questions.


●●○○○ AT&T 12:02 AM

Power Allowance Done

When a switch turns on...

Which?
Tap to set

Turn it off how many minutes later?

When?
Tap to set

Assign a name
Tap to set

Set for specific mode(s)
Choose Modes

Remove

Q5: COMMENT BOX

Please describe, to the best of your knowledge, what "Smarthome Capabilities" are

Q6: DESCRIPTIVE TEXT

Smarthome capabilities represent the access to the devices in the smarthome. For example, an app can request the montionsensor capability to access the montion sensor.

Q7: SINGLE-SELECT MATRIX

What factors will you consider when making decision of whether install a third party app or not? And please indicate how much you care on each factor that you will consider.

	Strongly care	Care	Neither care or not care	Not care	Strongly not care
The source / author of the app	○	○	○	○	○
The popularity of the app	○	○	○	○	○
The functionality of the app	○	○	○	○	○
The privacy aspect of the app	○	○	○	○	○
The smarthome capabilities that the app request	○	○	○	○	○
The relation of capability requests to the app's functionality	○	○	○	○	○
Others	○	○	○	○	○

Q8: COMMENT BOX

What is the other factors that you consider when you install a third party app?

Page 3

Q9: SINGLE-SELECT MATRIX

Third-party apps can access devices in the smart home after they are installed. Please rate the risk levels of the different behaviors to access devices:

	Very sensitive	Sensitive	Slightly sensitive	Not sensitive
Lock your door	○	○	○	○

Unlock your door	○	○	○	○
Read the input of your door lock	○	○	○	○
Read the battery level	○	○	○	○
Read your motion sensor	○	○	○	○
Control your water pump	○	○	○	○
Turn on/off your light	○	○	○	○
Adjust the level of your light	○	○	○	○

Q10: SINGLE-SELECT LIST

Currently, once you select a lock to share to your smarthome third-party app, you give the app the capability to lock/unlock the door, read the input and all the other aspects to read or control the lock, do you hope to grant the app only part of the capability, for example, only to read the lock's status instead of unlock the lock?

1 ○ Yes, I would like to have more control over the app to grant less information/control
2 ○ No, I don't care
3 ○ No, it is too much trouble for me to control the app

Q11: SINGLE-SELECT LIST

Similar to smarthome capabilities, Android or iOS also provide permissions to third-party apps to control the access to resources in the mobile phone such as your location and contact book. Which one do you think is more sensitive?

1 ○ Smarthome capabilties are more sensitive
2 ○ Android or iOS permissions are more sensitive
3 ○ I think they are the same
4 ○ I don't know

Q12: COMMENT BOX

Please explain your reasons for your answers in the last question.

Q13: COMMENT BOX

What is your age?

Q14: SINGLE-SELECT LIST

What is your gender?

1 ◯ Female
2 ◯ Male
3 ◯ Decline to answer.

Q15: SINGLE-SELECT LIST

What is your highest education level of education you have completed?

1 ◯ No high school
2 ◯ Some high school
3 ◯ High school graduate
4 ◯ Some college - No degree
5 ◯ Associates / 2 year degree
6 ◯ Bachelor / 4 year degree
7 ◯ Graduate degree - Master, PhD, professional, medicine, etc

Q16: DROP DOWN LIST

Which of the following best describes your primary occupation?

1 Administrative support (e.g., secretary, assistant)
2 Art, writing, or journalism (e.g., author, reporter, sculptor)
3 Business, management, or financial (e.g., manager, accountant, banker)
4 Computer engineer or IT professional (e.g., systems administrator, programmer, IT consultant)
5 Education (e.g., teacher)
6 Engineer in other fields (e.g., civil engineer, bio-engineer)
7 Homemaker
8 Legal (e.g., lawyer, law clerk)
9 Medical (e.g., doctor, nurse, dentist)
10 Retired
11 Scientist (e.g., researcher, professor)
12 Service (e.g., retail clerks, server)
13 Skilled labor (e.g., electrician, plumber, carpenter)
14 Student

15 Unemployed
16 Decline to answer
17 Other

A.2.2 Smarthome Patching

Our patching script is written in roughly 600 lines of python code to modify the original Groovy source file by the following steps. A toy example for a patched app Turn It On For 5 Minutes is given in Listing A.1.

Listing A.1: We provide a code snippet for patched IoT app Turn It On For 5 Minutes. Text in blue indicates statements that need to be patched, and text in red indicates either new code instrumented by the script or replaced with our wrapped functions. The `appSetting` section added after the definition block is used for OAuth configuration.

```
definition(
        name: "Turn It On For 5 Minutes",
        namespace: "smartthings",
        author: "SmartThings",
        description: "When a SmartSense Multi is opened, a switch will be
                turned on, and then turned off after 5 minutes.",
        category: "Safety Security",
    ... \\
) {
        appSetting "client_idFPS" // used to config app identifier for OAuth.
        appSetting "client_secretFPS" // used to config app secret for OAuth.
        appSetting "http_serverFPS" // we configure cloud server url here.
}
...
mappings { // act as end-points for policy enforcement module to deliver event data
        path("/post_event") {
                action: [
                        POST: "handleEventFromProxyServer"
                ]
        }
}
preferences {
        section("When it opens...") {
                input "contact1", "capability.contactSensor"
        }
        section("Turn on a switch for 5 minutes..."){
                input "switch1", "capability.switch"
        }
}
def installed() {
        log.debug "Installed with settings: \${settings}"
        subscribe(contact1, "contact.open", contactOpenHandler)
        subscribeToServer(contact1, "contact", "open", contactOpenHandler)
}
```

```
def updated(settings) {
        log.debug "Updated with settings: \${settings}"
        unsubscribe()
        unsubscribeToServer()
        subscribe(contact1, "contact.open", contactOpenHandler)
        subscribeToServer(contact1, "contact", "open", contactOpenHandler)
}
def contactOpenHandler(evt) {
        switch1.on()
        sendCommandToProxyServer(switch1, "on", NULL, NULL, NULL, NULL)
        def fiveMinuteDelay = 60 * 5
        runIn(fiveMinuteDelay, turnOffSwitch)
}
def turnOffSwitch() {
        switch1.off()
        sendCommandToProxyServer(switch1, "off", NULL, NULL, NULL, NULL)
}
...
```

To enable authorization in the for policy enforcement module, the script automatically inserts dynamic pages and prepares a URL for the patched app to enable an OAuth authentication flow at install time. The SmartThings platform provides a trigger for an OAuth authorization flow via the URL containing an app identifier and its cloud-generated app secret. When the user navigates to the URL, they will be redirected to the SmartThings login page to enter credentials and receive an authorization token for later use.

The script next scans all devices on the SmartThings capability list[1] by parsing all input labels from the `preferences` section and its corresponding child pages, e.g., `mainPage` page section. The script builds an internal structure called DL, maintaining a pair of information (input label, device capability), for later code substitution for command or attribute statements.

The script then parses event handler subscription and unsubscription statements by scanning the keywords. A subscription statement consists of its input label, associated attributes, and the corresponding event handler function. For instance, `subscribe(motionSensors, "motion.active", motionActive)` means the app subscribes an event handler for status activity of input `motionSensors` which has motion capability, and assign function `motionActive` as callback handler. Therefore, our patching engine replaces this statement with an internal function `subscribeToServer()` to send all corresponding parameters to the policy enforcement module along with its app identifier. The module will determine whether this subscription is allowed depending on user's rules. If successful, the module will forward the event data to the registered SmartApp. Unsubscription is much easier to implement, namely by removing all subscriptions registered on the policy enforcement module.

The last step is to search all statements for possible command issuing or attribute retrieving

[1]`http://docs.smartthings.com/en/latest/capabilities-reference.html`

associated with those device labels collected above. For example, the structure DL may contain an input device called **`switch1`** which has a `switch` capability. When the script parses a statement containing the label switch1, e.g., `switch1.on()`, the script catches the function call `on()` and checks against a capability structure defined based on the list of capabilities and their associated functions and attributes[2]. Once the script confirms the call or attribute, it replaces the original statement with the internal API call `sendCommandToProxyServer()` by sending the request to the policy enforcement module with its app identifier, device label (switch1), command label (`on()`) and any corresponding parameters.

After patching, each Groovy source file will contain around 128 new lines to provide endpoint interfaces for the policy enforcement module.

A.2.3 SmartAuth Working Example

Here we use one example to show how SmartAuth works. The Flasher is an app that claimed to flash a set of lights to notify user when motion, open/close event, or switch event is detected. However, besides subscribing to motion sensor, contact sensor, and switch, the app also subscribes to the presence sensor and the acceleration sensor. To bridge the gap between what the users think the app do and the app's real behaviors, we generate the security policy from the code and from the description. We display the verified capabilities according to their functionality, and notify users about the unexpected behaviors, similar to Figure 3.16. On the interface, we further classify the unexpected actions into "unexpected" and "dangerous", according to the user perception measured through our crowd-sourcing result. We present the security policy and unexpected/dangerous behaviors in a usable authorization interface. After getting the response from the users, we enforce the policy so that the app only gets what it needed for the functionality and what the user understand and would like the app to access.

A.2.4 Apps Used in the Lab Study

We show the participants five group of apps in the SmartAuth and SmartThing interface, as shown in Table A.2.

A.2.5 Details of the Lab Study for SmartAuth

We used e-mails and posters on Carnegie Mellon University Silicon Valley campus, Indiana University Bloomington, and the Samsung Mountain View office to recruit participants for the in-lab experiments of choosing a smarthome application. In the recruiting message, we explained that the experiments would be about selecting smarthome applications to install. To avoid potential biases, we didn't tell users that permissions and functionality matter in the experiments. We looked for participants who meet the following requirements: (1) Participants must be 18 or older. (2) Participants must have experience using a smartphone. (3) Participants should have basic knowledge about smarthome. (4) Participants must live in the United States and be literate in English. The participants was invited to our lab

[2]`http://docs.smartthings.com/en/latest/capabilities-reference.html`

and use our phones to do the experiments. They were asked to review and sign the consent form before they start working on the study. Next we explained to participants the interface of the smarthome to complete the app-selection task, and then we asked them questions about this experience. The participants saw the app-installation pages and compare the apps to choose one that they would like to install. We observed the user's app-installation decisions and asked them questions to see whether our interface helps the user understand the functionality and avoid privacy breach. At the end, we asked the participants some demographic questions. The participant was compensate for a 5-dollar Amazon gift card or two slices of pizza. The total process will took approximately 20 minutes.

If you are interested to learn more details about the study, please find the following the recuirtment message used at Carnegie Mellon Silicon Valley:

We are from Carnegie Mellon University, doing research about app usage. If you have used mobile device for at least 1 month and you are an adult living in the US, you would be very helpful to our study. You will receive a gift card of 5 dollars if you finish the study. Thanks!

Table A.2: Apps in the lab study

App ID	**App Name**	**Description**	**Overprivileged? If so, Behavior Type**
1A	Smart Humidifier	Turn on/off humidifier based on relative humidity from a sensor.	NO
1B	Humidity Alert	Notify me when the humidity rises above or falls below the given threshold. It will turn on a switch when it rises above the first threshold and off when it falls below the second threshold.	YES, Lock (Dangerous)
2A	Virtual Thermostat	Control a space heater or window air conditioner in conjunction with any temperature sensor, like a SmartSense Multi.	YES, Motion Sensor (Dangerous)
2B	Smart Heater	Turn on/off the heater based on the temperature.	NO
3A	Lights Off	Turn lights off when no motion and presence is detected for a set period of time.	NO
3B	Darken Behind Me	Turn your lights off after a period of no motion being observed.	YES, Temperature Sensor (Unexpected)
4A	Flash A Notice	When something happens (open/close, switch on/off, motion detected), flash lights to indicate.	NO
4B	The Flasher	Flashes a set of lights in response to motion, an open/close event, or a switch.	YES, Presence Sensor (Unexpected)
5A	Left It Open	Turn lights off when no motion and presence is detected for a set period of time.	YES, Power Meter (Unexpected)
5B	Smart Window	Compares two temperatures - indoor vs outdoor, for example - then sends an alert if windows are open (or closed!). If you don't use an external temperature device, your zipcode will be used instead.	NO

If you are interested in participating in this research study, you must meet the following requirements:

(1) Participants must be 18 or above

(2) Participants must have experience of using a smartphone

(3) Participants should have basic knowledge about smart home

(4) Participants who live in the USA and literate in English

Your participation in the study is voluntary. The study will be in Building 19, NASA Ames Research Center, CA 94035 and the study takes around 20 minutes. You will read the user interface of Smart Home App Installation to choose Apps to mange the Smart Home devices. You will then be asked survey questions about smarthome devices management.

Please contact Yuan Tian (yt@cmu.edu) if you are interested to join the study.

When the participants came to the lab to join the study, we gave them a paper consent form to sign and the instructions about the study. The instructions wre similar to the purpose of this study section and the procedures section in the consent form. Please find attached the consent form to see more details.

Consent Form for Participation in Research

Study Title: Smart Home Device Management Study

Principal Investigator: Yuan Tian, Ph.D candidate
Electrical and Computer Engineering, Room 1033, Building 19, NASA Ames Research Center, CA 94035
650-862-0576, yt@cmu.edu

Faculty Advisor: Patrick Tague, Associate Research Professor

Purpose of this Study
The purpose of the study is to investigate the users' management of smart home devices.

Procedures
If you decide to participate in the study, first you will first read the consent form of the study and being asked a few screening questions to make sure that you meet the requirements of the study. Then we will ask you to read the user interface of Smart Home App Installation to choose Apps to mange the Smart Home devices. Your major task is to choose one app from five groups of apps. After you finish your major task, we will then ask you questions about the smarthome devices management experiences. In the end, we will ask you some demographic questions.

The study will be in Building 19, NASA Ames Research Center, CA 94035 and the study takes around 20 minutes.

Participant Requirements
[List the requirements for inclusion of participants in the study. Include age requirement.]
(1) Participants must be 18 or above
(2) Participants must have experience of using a smartphone
(3) Participants should have basic knowledge about smart home
(4) Participants who live in the USA and literate in English

Risks
The risks and discomfort associated with participation in this study are no greater than those ordinarily encountered in daily life or during read words from Smart Phone screen.

Benefits
There may be no personal benefit from your participation in the study but the knowledge received may be of value to humanity.

Consent Form for Participation in Research

Compensation & Costs
You will be paid 5 dollars after finishing the study.
There will be no cost to you if you participate in this study.

Confidentiality
By participating in the study, you understand and agree that Carnegie Mellon may be required to disclose your consent form, data and other personally identifiable information as required by law, regulation, subpoena or court order. Otherwise, your confidentiality will be maintained in the following manner:

Your data and consent form will be kept separate. Your research data will be stored in a secure location on Carnegie Mellon property. Sharing of data with other researchers will only be done in such a manner that you will not be identified. By participating, you understand and agree that the data and information gathered during this study may be used by Carnegie Mellon and published and/or disclosed by Carnegie Mellon to others outside of Carnegie Mellon. However, your name, address, contact information and other direct personal identifiers will not be mentioned in any such publication or dissemination of the research data and/or results by Carnegie Mellon. Note that per regulation all research data must be kept for a minimum of 3 years.

We will not record any personal information about you other than your signature on the consent form. The data we collected will be transmitted in an encrypted way and stored in a password protected server in the lab anonymously. Only the investigators of this project can access the data.

Rights
Your participation is voluntary. You are free to stop your participation at any point. Refusal to participate or withdrawal of your consent or discontinued participation in the study will not result in any penalty or loss of benefits or rights to which you might otherwise be entitled. The Principal Investigator may at his/her discretion remove you from the study for any of a number of reasons. In such an event, you will not suffer any penalty or loss of benefits or rights which you might otherwise be entitled.

Right to Ask Questions & Contact Information
If you have any questions about this study, you should feel free to ask them now. If you have questions later, desire additional information, or wish to withdraw your participation please contact the Principal Investigator by mail, phone or e-mail in accordance with the contact information listed on the first page of this consent.

If you have questions pertaining to your rights as a research participant; or to report concerns to this study, you should contact the Office of Research Integrity and Compliance at Carnegie Mellon University. Email: irb-review@andrew.cmu.edu . Phone: 412-268-1901 or 412-268-5460.

Consent Form for Participation in Research

Voluntary Consent

By signing below, you agree that the above information has been explained to you and all your current questions have been answered. You are encouraged ask questions about any aspect of this research study during the course of the study and in the future. By signing this form, you agree to participate in this research study. A copy of the consent form will be given to you.

____
PRINT PARTICIPANT'S NAME

____ ____________________
PARTICIPANT SIGNATURE DATE

I certify that I have explained the nature and purpose of this research study to the above individual and I have discussed the potential benefits and possible risks of participation in the study. Any questions the individual has about this study have been answered and any future questions will be answered as they arise.

____ ____________________
SIGNATURE OF PERSON OBTAINING CONSENT DATE

In the following, I attach a few questions asked in the study.

Please choose how much you agree with the following statements. {Strongly disagree, disagree, neither agree or disagree, agree, strongly agree}.

- I feel that the app description explains thoroughly why the app can access and control these sensors and devices.
- I feel confident to make a decision whether or not to install the app after reading the description.
- It is difficult to find information from the description.

A.2.6 Crowdsourcing for Unexpected Behavior Sensitivity

We evaluated how sensitive the unexpected behaviors are by combining expert reviews and crowdsourcing together. In particular, we had two security experts and 100 Mturkers to look into the apps' unexpected behaviors and evaluated how sensitive the unexpected behavior is given the context of the app. We asked the participants to classify whether these unexpected behaviors are dangerous or not(dangerous is counted as 1, and not dangerous is counted as 0). From the expert and Mturk responses, we assign each security expert a weight of 0.25, and each Mturker a weight of 0.005. If the weighted sum is over 0.5, we consider the behavior as dangerous.