Developer-Specified Explanations for iOS Permission Requests: Adoption and User Effect

Heidi Negrón-Arroyo*, Khanh Nguyen†, Joshua Tan‡, Michael Theodorides§, David Wagner§
*University of Puerto Rico, Mayagüez †University of California, Riverside
‡North Dakota State University §University of California, Berkeley

***Abstract*—In iOS 6, users are prompted by apps for access to a protected resource such as location, contacts, or photos. When a user is using an app feature that requires a protected resource for the first time, the user will be prompted with a permission request that the user can choose to allow or deny. We examine how app developers are using this mechanism through an Internet survey and iPhone app testing. In addition, we examine whether permission requests are helpful to smartphone users in making proactive decisions regarding their privacy. We conducted a usability study with an Internet survey of 807 smartphone users. Our results reveal when users become more engaged in decisions regarding their mobile data privacy. Finally, we present recommendations for how this mechanism could be improved.**

***Index Terms*—iOS, smartphones, mobile phones, security, smartphone permissions, mobile data privacy**

I. Introduction

In September 2012, Apple released its current mobile operating system iOS 6. This version of iOS instated new privacy requirements where users are explicitly asked for access to a protected resource [1]. The protected resources in iOS 6 are Photos & Videos, Contacts, Location, Calendars, Reminders, and Bluetooth. In previous versions of iOS,only the location resource was protected. Prior versions of iOS allowed app developers to specify a custom purpose string, a message explaining to the user why permission to the protected resource should be granted. iOS 6 extends the custom purpose string feature by allowing developers to add a purpose string to the permission request for any protected resource.

When a user is prompted with a permission request, she has the option to allow the request or deny the request and continue using the app, albeit with limited functionality. Some apps heavily rely on access to protected resources and may block further use until the user grants permission. We performed manual testing on 140 iPhone apps to identify how developers are using permission requests and custom purpose strings. Then, we applied statis analysis to a corpus of 4,400 apps to gather similar detail on a larger scale.

Our manual testing and binary analysis revealed several common types of purpose strings. We used an Internet survey to determine which types of purpose strings users find most helpful in making an effective decision about their mobile privacy. The user survey was designed to help us understand the effect of purpose strings on users' behavior. In addition to the user study, we surveyed iOS developers to learn why developers are or are not using custom purpose strings in their apps.

Static analysis revealed that approximately one fifth of all permission requests contain a purpose string. Our findings also show that smartphone users are responsive to purpose strings. Only 23% of developers surveyed reported using a purpose string. From our results, we recommend that developers include a purpose string with their permission requests. To encourage the use of purpose strings, we propose creating a set of predefined strings from which developers can choose.

II. Background and Related Work

A. iOS 6 Permissions

To protect the privacy of iPhone users, iOS 6 apps are required to obtain explicit permission from the user before accessing a protected resource [1]. These resources include Location, Contacts, Photos and Videos, Calendars, Reminders, and Bluetooth. Once the user allows an app to access a certain resource, that app will continue to have access to that resource, unless the user makes changes to the privacy options in the device settings. When asked to grant an app a permission, users see a screen similar to that in Figure 1.

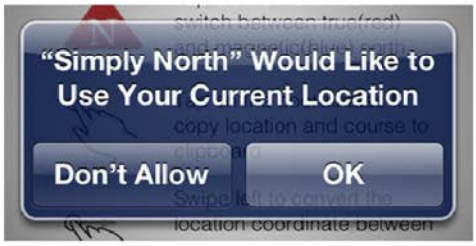

Fig. 1. A location permission request for the Simply North app that does not include a developer-specified purpose string.

In previous versions of iOS, developers could add a custom message to requests for access to location. In iOS 6, developers can add a custom purpose string to requests for any protected resource. Purpose strings offer developers the opportunity to explain or convince users why the app needs access to a particular resource. Developers can explain to smartphone users how the information will be used or what benefit the user would gain from granting the app's permission request. Figure 2 shows a permission request dialog with a developer-specified purpose string.

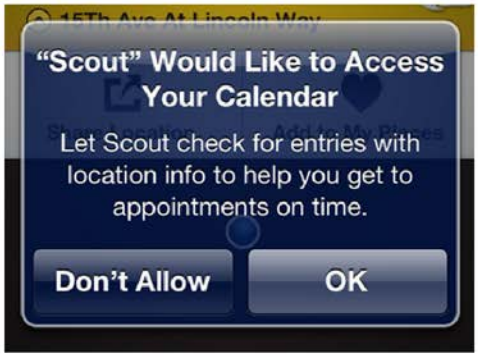

Fig. 2. A calendar permission request for the Scout GPS app that includes a developer-specified purpose string.

To include a custom purpose string, iOS developers can specify it in the either the Info.plist or InfoStrings.plist files of the app. Property list (plist) files are files that iOS uses to store serialized objects such as user or app settings.

For the contacts, calendars, reminders, photos and videos, and bluetooth sharing permissions these plist files are the only way to specify a custom purpose string. However, for the location permission developers have the option to use a deprecated method for specifiy a purpose string by programatically invoking an API from previous versions of iOS.

B. Previous Work in iOS Developer-Specified Purpose String Adoption

Previous research regarding iOS developer-specified purpose strings has been done by Christopher Thompson and Serge Egelman at the University of California at Berkeley in a position paper. In their position paper Serge Egelman and Christopher Thompson report that out of a corpus of 4,395 apps compiled of the top 200 free apps from all categories of the iTunes app store, they found that only 100 apps, or 2.3%, specified any purpose string at all. They estimate the number of apps that request resource permissions by assuming that certain categories of apps require certain resources. For example they assume that all apps in the navigation category request access to location. In our paper we wish to expand on the number of apps that request resource permissions by developing a static analysis technique that can better estimate the number of apps that request certain resources and complementing the static analysis with a manual testing of apps, in which we individually test apps to see whether a runtime permission asks for permission to access a certain resource.

III. Methodology

We analyze the propery list (plist) files of 4,400 apps from the iTunes store to extract developer-specified purpose strings. We decrypted these apps and then analyzed their binaries with a form of static analysis, to search for iOS API methods that trigger resource requests. We also manually tested 140 of these apps to record the same information and we compare these results with those obtained by our static binary analysis. In addition, we conducted two Internet surveys: one administered to iOS developers and the other smartphone users.

A. Purpose String Extraction

In order to quantify the level of developer adoption of purpose strings, we analyzed 4,400 apps, chosen from the top 200 free apps in each category on the iTunes App store, and extracted the plist-specified purpose strings they contained. The location of these strings within an app was predictable, allowing us to automate this task with a Python script. This script recursively searched the unencrypted directories of the app, searching for filenames matching `Info.plist` or `InfoPlist.strings`. Once one of these files was found, it was searched for the existence of one or more usage description keys, such as `NSContactsUsageDescription` [2]. The values corresponding to these keys are the developer-specified purpose strings for resource requests. Each discovered purpose string was added to a database, along with information on the resource it explained and the app in which it was found. This process only finds purpose strings specified using the plist method, but not those using the older deprecated method.

We manually examined purpose strings for the 140 apps that we manually tested and performed a sorting exercise to cluster them into similar types. We found nine distinct categories of purpose strings:

1) Specific benefit/purpose: the purpose string describes a specific benefit to the user that would result form allowing the app access to a particular resource
2) Generic purpose: the purpose string describes broadly why the user should allow the app access to a protected resource
3) Sharing: the purpose string describes how the information may be uploaded and who the information will or will not be shared with
4) Local use: the purpose string describes how the data access will not be uploaded or shared
5) Emotion: the usage string appeals to emotion or the qualitative benefit from allowing resource access in a non-specific way
6) Security: the purpose string describes how the information will be secured or protected against unauthorized use
7) Appropriate use of resource: the purpose string promises that the data access will not be misused
8) Data storage: the purpose string describes how the data access will or will not be stored
9) Developer confusion: the purpose string demonstrates the developer's confusion about how to use the purpose string

B. Static Binary Analysis

We use static analysis of the app binaries to find use of the deprecated purpose string specificiation method, and to find which resouces each app uses - i.e which permission request

will trigger. Purpose string extraction by itself would not allow us to measure the developer adoption of purpose strings. Not all apps request protected resources, and for those that do, not every resource is requested. We then use this information to calculate the rate at which developers are taking advantage of the purpose string mechanism.

We use a very simple form of static analysis, where we dissasemble the app code and search for specific strings in the disassembly output. iOS binaries are stored in the Mach-O file format and contain the information necessary for the Objective C runtime to execute the app. When the compiler compiles Objective-C code, all method invocations are compiled by including the method name as a constant string (the selector) in the compiled binary [1]. We search the binary for these strings. Initially, we used `otool` to search for these selectors in the `__objc_methname` section of the `__TEXT` Mach-O segment. However, limiting the search to this section proved insufficient; at times, selectors were found in other parts of the binary, including the `__DATA` segment. To solve this issue, we instead used the Unix tool `strings`, which enabled us to retrieve printable strings from all parts of the app binary.

iOS apps store their binaries in encrypted form. To enable our analysis, we decrypted the binaries using a tool called *Rasticrac* on a jailbroken iPhone [3]. Due to the lack of a jailbreak method for the current iOS version (v6.1.4) at that time, we used iOS version v6.0.2. Any apps that were incompatible with iiOS v6.0.2, or that failed to decrypt properly, were replaced with by other apps from the top 200.[2] Once the apps had been decrypted, we searched through the strings dump looking for any matches with our search string list. If a match was found, then the resource corresponding to that string was flagged as requested.

The particular strings we searched for in the binary correspond to the selectors for API methods that trigger a resource permission request. We compiled a list of search terms for each of the six protected resources using the developer documentation provided by Apple for each type of resource. For example, to request access to a user's address book (and the underlying iOS Contacts resource), a developer can use the function `ABAddressBookRequestAccessWithCompletion` of the ABAddressBook class. In addition to API functions used to explicitly request resource access, we also added any functions which seemed likely to directly interact with a protected resource. `setProperty:`.

To detect when an app specifies a purpose string for location using the deprecated API we also searched for the selector of that deprecated API, namely,`setPurpose:`.

C. Manual Testing

We chose the top 10 free apps from 14 of the 22 categories in the iTunes store to perform manual testing on. The 140 chosen apps were a subset of the ones we applied binary analysis to. The primary goal during manual testing was to trigger and record all resources that an app might request permission for. Unlike in Android, iOS permission dialogs for accessing resources are triggered and requested at runtime, namely, when an app attempts to access a resource for the first time. Therefore, we tried to explore every menu option and screen while performing manual testing.

A secondary goal of manual testing was to record purpose strings for those not specified in plist files. Although our static analysis predicts whether or not a Location purpose string was specified using the deprecated method, it does not extract the purpose string specified. We recorded screenshotsof the dialogs during the manual testing process. We used the results of manual testing to evaluate the static analysis, by comparing the results of manual testing to the results of static analysis on each of these 140 apps.

D. User Survey

We conducted a survey of 807 smartphone users to learn more about how they view permission requests. We presented them with different types of permission requests and asked them how they would respond. We recruited participatns on Amazon Mechanical Turk.

The survey consists of three scenarios. Each scenario showed them the dsecription of an app, a screenshot of a permission request from that app, and them asked them what they'd do (approve or deny?) and asked them for their opinion about eh scenario. One scenario used a real purpose string screenshot is a screenshot of a real app's permission request that uses a custom purpose string. Another scenario used a no purpose string screenshot: a real purpose string screenshot that we altered to remove the custom purpose string element from the permission request. Finally, the last scenario showed a fake purpose string screenshot, namely, a screenshot of a fake application, Party Planner, which uses a permission dialog from a real purpose string screenshot.

From our manual app testing, we found collected 21 screenshots of custom purpose strings. Of these, 16 were selected to be included in the survey as real purpose string screenshots. The selected screenshots cover an array of resource requests and purpose string types. To compare the effect of a custom purpose string on a permission dialog's usability, each of the real purpose string screenshots have a no purpose string counterpart.

To eliminate the possible effect of branding, we use the fake purpose string screenshots. Each fake purpose string screenshot adapts an existing purpose string so a brand name app's name is removed. For example, Gmail's purpose string "Let Gmail use your contacts to autocomplete email addresses" becomes "Let Party Planner use your contacts to autocomplete email address," where Party Planner is a generic fake app name.

Not all of the Party Planner screenshots have a real purpose string screenshot counterpart in the survey. Included in the 14 fake purpose string screenshot pool are real purpose strings from apps not included in real purpose string screenshots and

[1] At runtime, these selectors are passed as an argument to the method `objc_msgSend`

[2] The iTunes top 200 app charts updated frequently and we were always able to find a new replacement app within the top 200 when the need arose.

real purpose string variants. An example is "Your contacts will be used to find your friends," which is a variant of Snapchat's "Your contacts will be transmitted to our servers and used to find your friends."

The survey shows the user one of each type of screenshot. For each screenshot, the survey taker is first given a description of the app from the iTunes store and then asked to identify and classify the application. This is done as a quality assurance measure to weed out any unqualified survey responses. Then the user is shown a permission dialog screenshot corresponding to that app and is asked to identify the resource the app is trying to access and rate the usability of the permission dialog with 12 Likert statements.

The first two screenshots the survey taker sees are the real purpose string and no purpose string screenshots. The order of the first two screenshots is randomized to avoid any bias from seeing a screenshot with a purpose and then one without and vice versa. Additional control logic in the order is implemented so that a survey taker sees screenshots from three distinct apps in order to avoid any priming. The control logic also prevents survey respondents from seeing a repeat string in the Party Planner screenshot.

E. Developer Survey

A survey targeting developers was designed to study the usage of purpose strings from the developers perspective. This survey gives us insight to their general developer understanding of privacy controls in iOS 6. The three main goals of this survey are:to determine whether developers are aware of the purpose strings feature, to learn why or why not they are using these custom messages and test whether they can effectively communicate with the user. To qualify for the survey a developer must have at least one application approved by the iTunes App Store. Since the responses are anonymous, there may be responses that do not meet this criterion. To reduce the number of unqualified responses, there were questions to confirm that the survey taker had sufficient knowledge of app development. Since it is harder to reach developers, we published ads at iOS app development forums, namely Stack Overflow and iPhone Dev SDK. If the survey was completed, developers were offered an opportunity to enter a raffle to win a $100 Amazon Gift Card.

As part of the survey, developers were provided a screenshot displaying the use of a purpose string and its definition. They were asked whether they had heard of it and if they had ever used it. Open-ended questions followed to get responses about why they decided to use or not use purpose strings in the apps they have developed. This would help us determine if developers find this new feature in iOS 6 is helpful and what information an efficient purpose string should consist of. We study if there are resources that developers are currently specifying more purpose strings compared to the others. Additionally, we can learn from developers their thoughts on the advantages and disadvantages of using purpose strings.

In this survey we also aim to understand the developers thoughts on users privacy decisions. Likert scale questions study how developers believe the users react when purpose strings are shown in a permission request. If the purpose of these strings is to help the user in making more conscious decisions on how they control and their information, we hope to understand what developers are doing to help it. Consequently, we can study what kind of information developers think is relevant to include in a purpose string.

To study what developers are including in their purpose strings, screenshots from two applications were shown to the developer taking the survey. The screenshots were displayed a permission requesting access to contact information. Developers were asked to write a custom purpose string based on the description provided for the application. The description for each application included features and privacy information from the iTunes app store and the privacy policies of each app. Then, the custom message given by the developer is categorized by the information it conveys using 9 categories identified in our study.

IV. Developer Adoption

A. Comparison of Manual Testing and Binary Analysis

To gauge the effectiveness of the static binary analysis in predicting resource requests accurately, we compared the binary analysis results to our results from manual testing. No false negatives were returned by the binary analysis; whenever a resource request was found in manual testing, that request was correctly predicted in the binary analysis. This fact allows us to interpret the numbers returned by it as an upper bound.

Our binary analysis was less successful in avoiding false positives. Table I shows the number of apps flagged as requesting a resource requests when manual testing found no such requests. Although our analysis had a high rate of accurately detecting resource requests for calendar requests, reminder requests, and the existence of binary-sourced location purponse strings, our analysis overpredicts both address book and photos and vidoes requests by a factor of two or more.

Although our binary analysis technique gives a high false positive rate when compared with our manual testing, the actual false positive rate may be lower. In many cases, we repeated manual testing for an app based on reports from the binary analysis and discovered that in our initial manual testing we failed to detect a resource that the app requested. Most of these failures to detect resource requests during initial manual testing resulted because apps only request access to certain resources in rare circumstances that do not arise in basic app use. For example, in the Booking.com iOS app, users are only asked for permission to acces their contacts and reminders once a hotel reservation has been paid for and reserved.

On the other hand, it is also possible that many of the reported false positives are real. Some of the reasons for these results might include the following:

- **Incorrect selector.** The functions specified by the selectors in our search terms may not necessarily lead to a resource request.
- **Correct selector, wrong target object.** Although we may have correctly identified the selector for a function that

	Resource					
	Location	Photo/Video	Contacts	Calendars	Reminders	Location Purpose (Deprecated)
# Apps Flagged	40	55	27	22	5	19
# Apps Not Flagged	40	35	80	112	134	114
% Agreement	50.0	38.9	74.8	83.6	96.4	85.7

TABLE I
BINARY ANALYSIS RESULTS FOR CASES WHERE MANUAL TESTING FOUND NO POSITIVES.

	# Apps With		
Resource	Purpose Strings	Resource Requests	Purpose S. Rate (%)
Location	605	2359	23.8
Reminders	5	82	7.3
Contacts	64	1200	5.3
Calendars	11	508	2.1
Photo & Video	35	2631	1.3
Any	660	3486	18.9

TABLE II
USAGE STRINGS AND RESOURCE REQUESTS BY RESOURCE TYPE

Resource	# Apps Requesting Resource	Requst Rate (%)
Location	1269–2539	20.6–47.7
Reminders	79–82	7.3–7.6
Contacts	897–1200	5.3–7.1
Calendars	424–508	2.1–2.6
Photo/Video	1023–2631	1.3–3.4

TABLE III
RANGES FOR NUMBER OF RESOURCE REQUESTS AND ADOPTION RATE

trigger a resource request, in another class there may be an identically-named function that shares the appearance of the first selector, but does not request a resource. Since our approach does not consider function parameters, we cannot use it to determine the target object, in order to differentiate betweeen these cases.

- **Dead code.** Although the executable for an app may contain a function that necessarily requests permission to access a resource and/or include a purpose string for the request, there may be no actual program execution path that invokes this function.

B. Developer Adoption of Purpose Strings

Using our automated script, we analyzed our entire corpus of 4,400 apps for both purpose strings and resource requests. The results of the analysis are summarized in Table II.

The number of apps that provided purpose strings for the location resource alone was 5 times greater than the number of apps that provided a purpose string for all of the other resources combined. The predicted number of permission requests for the photo & video resource was greater than the the number predicted for the location resource. However, we must evaluate this number in the context of its perceived accuracy. We can use this perceived accuracy to generate probable numbers for each type of resource. Table III shows that the actual value for the number photo & video resource requests likely falls between 1023 and 2631.

Developer adoption of purpose strings was strongest for the location resource, followed by the reminders and contacts resources. Overall, 18.9% of developers specify purpose strings for resource requests. This is significantly different from the percentage reported by Thompson and Egelman in their own analysis of purpose string adoption [4], in which a lower bound of 2.3% was reported. However, the study they performed only considered the purpose strings specified in app's plist files, and did not incorporate purpose strings specified using the deprecated CLLocationManager method, as we do. If we include only plist-specified purpose strings for calculating the developer adoption rate, the adoption rate decreases to 3.6%. However, we choose to include this deprecated method in our calculations, since users are subjected to the same experience, regardless of which of these two methods is used.

C. Developer Consistency for Explaining Resource Requests

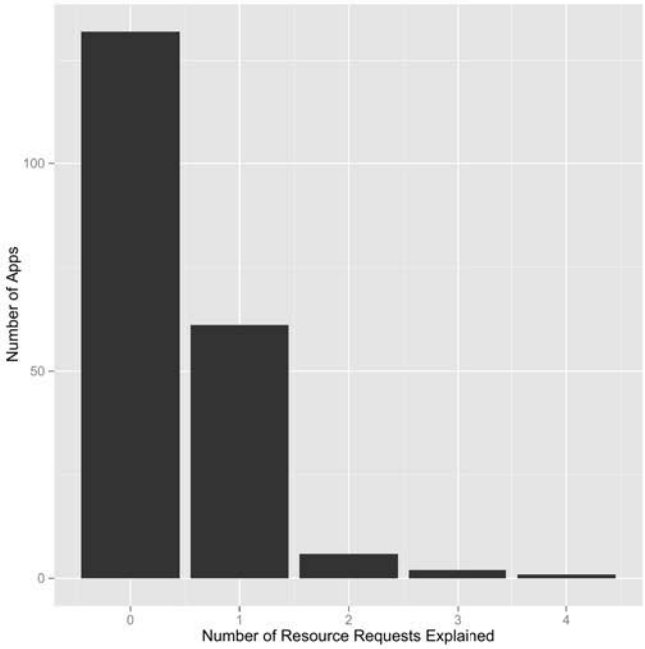

Fig. 3. Resource request explanations for apps that request 4 resources

In addition to determining the overall adoption rate of developers for specifying resource request purpose strings, we also examine the level of developer consistency for explaining these requests for all requested resources within an app. Figure 3 shows distribution for the number of resource requests that are explained for those apps that request access to four resources. From this, we can see that most apps do not provide purpose strings for all four of their resource requests. About half of this number provide an explanation for one resource request, with the number sharply falling for explaining two or more requests. Of the 61 apps that explained only one request, all but one was for the location resource. Furthermore, 57 of these 60 location purpose strings were specified using the deprecated method, instead of using plist files. This pattern was also found in the numbers for apps that request two, three, and five resources.[3]

D. Manual Testing - Results

Through our manual testing of 140 apps for resource requests and usage string use, we found that the plist method of including a purpose string for permission requests dialogs is not as widely used by iPhone apps as the deprecated method of including a location purpose string in the encrypted binary. We found that 9.78% of resource requests used the plist method to include a purpose string, while 17.39% of resource requests used either the deprecated binary method or the plist method. Through our manual testing, we also found that the location resource has a significantly higher adoption rate than any of the other resources. Of all location resource requests made by the 140 apps that we manually tested, 20% included a purpose string for the request dialog.

E. Issues with Bluetooth Permission Testing

Of the 140 apps that we selected for manual testing, none requested permission to access the bluetooth sharing feature. To further explore the bluetooth permission request we installed the first 10 apps from the search result list when searching the iTunes App Store for the keyword "bluetooth". Despite our efforts to trigger a bluetooth permission request from one of the 10 apps, we were unable to trigger the bluetooth sharing permission request dialog. The 10 apps required either special external bluetooth equipment or the use of another iPhone to pair with. We were unable to test apps that required special external equipment. However, even for the apps that only required a second iPhone, we were also unsuccessful at triggering a bluetooth permission request dialog. Our inability to trigger a bluetooth permission request dialog leaves us with no direct method of checking the validitiy of our automated testing for the bluetooth permission resouce request, for which we searched the binary for the methods of the iOS core bluetooth framework.

[3]Zero apps were found to request six resources, since no Bluetooth permission requests were recorded.

F. Developer Results

In the Internet survey focusing on developers, we received 39 responses. Due to inconsistencies in the answers, only 30 were considered in this study. Such inconsistencies included not answering basic questions regarding app development and non-serious responses in the open-ended questions. Most of the developers taking the survey have 1-5 apps published in iOS, making up 56% of survey takers.

From the adoption rates we confirm that developers are not using purpose strings very often. We found that 40% of the developers surveyed have not heard of the purpose strings feature before. Of those that know about this feature in iOS, only 41% have specified a purpose string in an app they have developed. This means that 77% of the developers that took the survey do not use purpose strings. In this paper, we attempt to clarify why the adoption of purpose strings is not significant in comparison to the number of apps requesting access to resources.

We found that the main reason for not specifying a purpose string was because purpose strings were not considered necessary. There were responses such as "because it is obvious" and "default description is good enough". In other cases, the app displayed a seperate dialog box explaining the purpose of the request before triggering a permission request. A survey taker stated that "Because we include an entire screen explaining the purpose ahead of requesting it". These results seem to suggest that developers avoid giving unnecessary privacy information to users.

On the other hand, for developers that have used purpose strings, their main reasons to use them could be classified in three categories:

1) Inform the user: "Users must be notified of information begin collected."
2) Increase the chance of the user enabling the feature: "Make people comfortable."
3) Good practice: "It is the right thing to do."

These categories imply that developers use purpose strings as an attempt to address users' privacy concerns and to encourage users to grant access. The biggest advantage of displaying a purpose string from a developer's perspective is primarily to inform the user. Some try to make users comfortable by gaining their trust and communicate the purpose of the request. This way they can make users grant access when it is requested. In contrast the fear of scaring users and invoking concern retrains some developers from using purpose strings. Other reasons mentioned by developers included the misuse of purpose strings and boring users with messages that are too long.

We find developers specify purpose strings in permission requests from certain resources more often than others. In this study, out of the 30 responses, 86.7% have requested permission to access the Location resource. Contacts, Photos & Videos, Reminders and Bluetooth were 36.7%, 63.3%, 6.7% and 10% respectively. From the developers that have specified purpose strings, all of them indicated they have included pur-

pose strings when requesting access to Location. For the other resources, 57.1% specified purpose strings in Contacts, 71.4% for Photos &, 14.3% for Reminders and 0% for Bluetooth. It is expected that for the most common used resources such as Location, results in more developers specifying purpose strings. However, when sensitive information such as contacts and photos is being accessed without an obvious reason it may become a custom to explain why it is accessing the resource.

When asked to write their own purpose string, based on a brief description and the screenshot requesting access to contacts, we got 29 responses that we classified in one or more of the nine categories that we determined from the manual testing of apps. For this question, they had to write a purpose string for two apps, Vine and Scout. In both apps, the most frequent type of purpose string provided was ppecific benefit/purpose. In the case of Vine, it was very common to also explain data storage information. In Scout, there were more purpose strings that were categorized as Emotion. Since the purpose of Scout using contacts can be considered less obvious than the first case, due to being a navigation app, it can be a reason why some developers try to convince the user in accepting the permission request through words that appeal to emotion such as: "We would need to access your Contacts in order to find their addresses, which will result in an extraordinary experience for you."

The results of the 7-point Likert scale responses show that in general, developers find purpose strings to be somewhat helpful for users and willl make them feel more comfortable when taking privacy decisions. For example, the average score for "Purpose strings make it easier for users to decide on the sharing of their information." was 5.9. Furthermore developers do not think that purpose strings will discourage the users and significantly affect the granting of access to resources. The average score for all Likert responses was 4.95.

The developers would also consider using purpose strings more often if iOS privacy controls included a set of predefined purpose strings that they can choose from. In previous research, Thompson and Egelman concluded that most of the purpose strings tend to vary little in policy semantics and explain similar concepts [4]. Providing a list of purpose strings with standard language could simplify the process, where developers can easily choose a string and avoid confusing the users.

V. User Behavior

We recruited 820 people for our user study. In our data, we removed 13 responses either because the survey taker was not qualified (not a smartphone user) or because the answers were nonsense or incomplete. We used the remaining 807 responses for analysis and to draw our conclusions about the effects of custom purpose strings on user behavior.

The main question our user survey asks is: do purpose strings help users make more effective decisions regarding their privacy? We examine this question two-fold: by comparing reported Likert scores and by comparing the effect of whether or not a screenshot with or without a purpose string was shown first to the survey taker.

Using a Wilcoxon signed-rank test for the Likert statement "It helps me make a more effective decision about the sharing of my information," our study shows that there is a significant difference between user ratings for screenshots with a purpose string and for screenshots without a purpose string ($p < 0.01$). Furthermore, the Likert data shows that priming users with the expectation of a purpose string did not affect the way they scored purpose strings versus non-pupose strings.

In contrast, user permission approval behavior demonstrates that priming does matter. We characterize user permission approval behavior as whether or not a user chooses to accept or deny a permission request given a screenshot. Using Fisher's Exact Test on the data in Table IV, we found there to be a significant difference in the permission request approval distribution of screenshots with a purpose string compared to those that did not have a purpose string when the user was primed by seeing a screenshot with a purpose string first (reject, $p < 0.01$). In contrast, Fisher's Exact Test on Table V showed there to be no significant difference between the with purpose and purposeless screenshot approval distributions when users are not primed (do not reject, $p = 0.1348$). When users are not primed to expect a custom purpose string, they are equally likely to allow or deny a permission request. The results of these tests suggest that user awareness of purpose strings affects their mobile privacy decision making.

	User Decision		
Request Type	Allow	Deny	Total by Type
Purpose String	320	106	426
No Purpose String	268	157	425
Total by Decision	588	263	851

TABLE IV
SCREENSHOT WITH PURPOSE STRING SHOWN FIRST

	User Decision		
Request Type	Allow	Deny	Total by Type
Purpose String	281	105	386
No Purpose String	261	125	386
Total by Decision	542	230	772

TABLE V
SCREENSHOT WITH PURPOSE STRING SHOWN SECOND

The user survey is also used to try to pinpoint successful elements of an effective purpose string. We used the fake purpose string Party Planner screenshots to identify which purpose string type is most effective in helping smartphone users make decisions about their mobile data privacy.

The Party Planner screenshots covered all categories of purpose strings, except for developer confusion. Using the Likert scores of each Party Planner screenshot, we investigated

whether or not particular types of purpose strings are preferred by users more than others. Three different Likert scores were used for analysis: the average score of the statement "It helps me make a more effective decision about the sharing of my information," the average score of the statement "It addresses my concerns over the sharing of my information," and the average of all Likert statements.

No clear best purpose string type could be determined from the data. However, across all three Likert measurements, one purpose string type was consistently ranked at the bottom. The generic purpose string was the weakest performer of the Party Planner screenshots earning an overall Likert score of 4.67. This result is not surprising, since generic purpose strings typically state something that is obvious or offer little information to the user. For example, the MapQuest location request adapted for Party Planner reads, "Contact access is required for this app to work properly." This purpose string offers little information to the user as to why the application requires contact access or how that sensitive information will be used.

In addition to examining the Likert scores by purpose string type, we analyzed the Likert score trends of the variants in the Party Planner screenshot pool. One surprising result from the variant study is performance of the adapted Instagram string "In order to find your friends, we need to send address book information to Party Planner's servers using a secure connection" and its variant which removes the phrase "using a secure connection" from the string. We expected the variant to do worse than the adapted Instagram string since it does not address a user's possible security concerns. However, on all three Likert scores, the variant does better than the original. This seems to indicate that less information is preferred by users.

This same trend where less is more is seen with the adapted Snapchat string ("Your contacts will be transmitted to our servers and used to find your friends.") and its two variants (variant 1: "Your contacts will be used to find your friends. They won't leave your phone," variant 2: "Your contacts will be used to find your friends.") In the overall Likert score and "effective decision" measure, the second variant earns the highest score from users.

VI. Discussion

A. The Impact of Expectations

In our user survey, the decisions made by users on whether to allow or deny permission request lacking explanation differed based on which version of the request they were first exposed to. This suggests that users form expectations for the level of information they require in making decisions that might place their privacy at risk. Our survey also suggests that users find resource requests containing purpose strings to be more helpful than those without explanations, regardless of whether the unexplained request is shown before or after the explained request. These two results combined allow us to infer the following: even if a user is not completely satisfied with the explanation of a resource request, as long as this user is unaware or unaccustomed to receiving more information, the end results of this user's decision-making process will be the same. The lack of alternatives may be all that is needed to compel users to make decisions that could negatively affect their privacy.

Users are not always given enough information to make informed and effective decisions concerning their privacy. This is evident by adoption rates in which less than one out of five resource requests is actually explained. Furthermore, the majority of purpose strings provided by developers pertain only to location resource requests. Responses from the developer survey suggest that developers avoid explaining resource largely because they find it to be an unnecessary task. Since the goals of most developers are accomplished when their customers are satisfied, the best way to solve this issue may be to find a way for users to increase the level of their expectations for privacy security.

B. Predefined Strings

The results from the user survey highly suggest that if users are more aware of permission requests with purpose strings, they will be more proactive in their mobile data privacy decision making. We believe that one way to increase user awareness of purpose strings, is to make them more common place. The challenge here is getting developers to use purpose strings with their permission requests.

Our developer survey shows that developers would be more willing to include purpose strings if a set of predefined purpose strings were available for them to use. We suggest that a set of predefined strings can be determined from seven of our nine purpose string categories (we exclude Generic purpose and Developer confusion because they would be unhelpful to users). The issue with this is that there is a broad range of Specific benefit/purpose type strings. Creating a pool of predefined strings will be difficult because it ideally should be accessible to all developers, but not so wide where it becomes cumbersome to find the right predefined string to accompany their permission requests.

Other issues that need to be addressed are warning fatigue and user comprehension. The current run-time permission prompt system implemented in iOS 6 is conducive to users developing warning fatigue. If users experience warning fatigue, they will be unlikely to read the purpose string the same way many users do not read End-user license agreements. Additionally, in the case that smartphone users do read the purpose string without fatigue, it would be meaningless for them to be proactive in their privacy decisions if they do not comprehend the purpose strings or understand the impact of their actions. For purpose strings to be effective in the future, users will need to learn the fundamentals of the iOS permission granting system and message developers are trying to convey through their custom purpose strings.

C. Verfying Accuracy of Purpose Strings

Although purpose strings can help users make better decisions on their privacy and security, some developers may

use these strings as a way to manipulate users into making decisions that ultimately cause harm to them. To prevent these types of situations, it will be important to have a means of verify the accuracy and truthfulness of different types of purpose strings. This task may be easier to manage if the different types of purposes strings are constrained to specific set of predefined strings.

Our research also shows that when not given enough guidance users do not always make the privacy decisions that are most beneficial to them. In our user survey we found a trend in which purpose strings that contained less information had a higher "effective decisions" likert score than purpose strings that addressed the security of user data. Pre-defined strings can be guide users in making better privacy decisions and aid users in understanding what is best for their privacy.

VII. Conclusion and Future Work

Our research indicates that smartphone users care about their privacy and make decisions to protect their privacy.

A set of pre-defined strings might encourage broader developer adoption of purose strings while providing users with the information that they need to make effective decisions concerning the sharing of their data. However, further research must be conducted to determine the correct level of information that pre-defined strings should provide. This further research should not only use subjective measures, but should also incorporate objective measures of purpose string comprehension. Further work in needed in the accuracy of purpose strings. Research should explore whether developers abide by what they mention in the purpose strings and whether developers are requesting access to excessive resources that they do not need.

Acknowledgment

This work was supported in part by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422), and by Intel through the ISTC for Secure Computing. Thanks to Rebecca Pottenger, Serge Egelman, Christopher Thompson, Erika Chin, and Aimee Tabor for their support and feedback.

References

[1] Apple. (2013) ios sdk release notes for ios 6. [Online]. Available: http://developer.apple.com/library/ios/#releasenotes/General/RN-iOSSDK-6_0/

[2] ——. (2012) Cocoa keys. [Online]. Available: https://developer.apple.com/library/mac/#documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html

[3] (2013) [release] rasticrac v3.0 epsilon 7, clutchpatched to be updated for ios 6. [Online]. Available: http://iphonecake.com/bbs/viewthread.php?tid=106330&extra=&page=1

[4] C. Thompson and S. Egelman, "[position paper] the effects of developer-specified explanations for smartphone permission requests," pre-print.

Appendix

A. Binary Analysis and Manual Testing Resource Agreement

	Static Anaylsis	
Manual Testing	Yes	No
Yes	60	0
No	40	40

TABLE VI
Location

	Static Analysis	
Manual Testing	Yes	No
Yes	50	0
No	55	35

TABLE VII
Photo/Video

	Static Analysis	
Manual Testing	Yes	No
Yes	33	0
No	27	80

TABLE VIII
Contacts

	Static Analysis	
Manual Testing	Yes	No
Yes	6	0
No	22	112

TABLE IX
Calendars

	Static Analysis	
Manual Testing	Yes	No
Yes	1	0
No	5	134

TABLE X
Reminders

	Static Analysis	
Manual Testing	Yes	No
Yes	0	0
No	0	140

TABLE XI
Bluetooth

B. Binary Analysis and Manual Testing Deprecated Location Usage String Agreement

	Static Analysis	
Manual Testing	Yes	No
Yes	7	0
No	19	114

TABLE XII
Location Purpose (Deprecated)