Autom Softw Eng (2015) 22:333–366
DOI 10.1007/s10515-014-0166-y

User-aware privacy control via extended static-information-flow analysis

Xusheng Xiao · Nikolai Tillmann · Manuel Fahndrich · Jonathan de Halleux · Michal Moskal · Tao Xie

Received: 3 September 2013 / Accepted: 7 August 2014 / Published online: 12 September 2014

Abstract Applications in mobile marketplaces may leak private user information without notification. Existing mobile platforms provide little information on how applications use private user data, making it difficult for experts to validate applications and for users to grant applications access to their private data. We propose a user-aware-privacy-control approach, which reveals how private information is used inside applications. We compute static information flows and classify them as safe/unsafe based on a tamper analysis that tracks whether private data is obscured before escaping through output channels. This flow information enables platforms to provide

This work was primarily done when Xusheng Xiao was at Microsoft Research as an intern.

This paper is an extended version of our previous work published at ASE 2012 (Xiao et al. 2012). Our previous work introduced the concept of user-aware privacy control via extended-information-flow analysis. In this work, we present a performance evaluation of our analysis and a user survey of the deployed system built based on our analysis.

X. Xiao (✉)
NEC Laboratories America, Princeton, NJ, USA
e-mail: xsxiao@nec-labs.com

N. Tillmann · M. Fahndrich · J. de Halleux · M. Moskal
Microsoft Research, Redmond, WA, USA
e-mail: nikolait@microsoft.com

J. de Halleux
e-mail: jhalleux@microsoft.com

M. Moskal
e-mail: micmo@microsoft.com

T. Xie
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: taoxie@illinois.edu

default settings that expose private data for only safe flows, thereby preserving privacy and minimizing decisions required from users. We build our approach into TouchDevelop, an application-creation environment that allows users to write scripts on mobile devices and install scripts published by other users. We evaluate our approach by studying 546 scripts published by 194 users, and the results show that our approach effectively reduces the need to make access-granting choices to only 10.1 % (54) of all scripts. We also conduct a user survey that involves 50 TouchDevelop users to assess the effectiveness and usability of our approach. The results show that 90 % of the users consider our approach useful in protecting their privacy, and 54 % prefer our approach over other privacy-control approaches.

Keywords Mobile Application · Privacy Control · Information Flow Analysis · Static Analysis

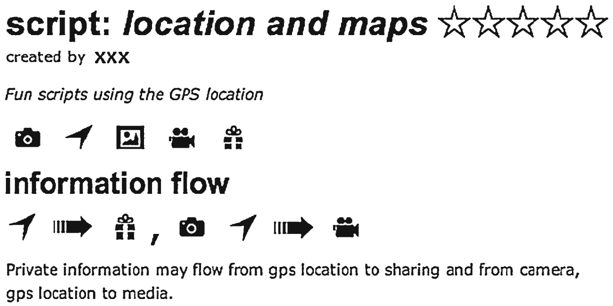
1 Introduction

Modern mobile-device platforms like iOS, Android, and Windows Phone provide a central place, called app stores or marketplaces, for finding and downloading third-party applications. A common problem faced by these mobile-device platforms is that the published applications in the marketplace may leak private user information through output channels. Many of these applications access mobile-device resources, such as pictures and GPS that may contain and expose private information, and share them using remote cloud services or web services without notifying users (Enck et al. 2010).

To mitigate these problems, privacy control mechanisms employed by mobile-device platforms include two major parts: (1) manual app validation by experts: experts employed by an app store manually exercise the functionality provided by an app and observe its behaviors for validation; (2) access-control granting by users: app stores ask for permissions before users can install applications (Android and Windows Phone), or an app requests permissions before it can access users' private information (iOS). The manual validation process is costly and delays publishing of apps. It is also incomplete, since it cannot examine every execution path to detect violations of privacy policies (Gilbert et al. 2011). Access-control granting provides information about *what* private information these applications may access, rather than *how* these applications use private information, causing users to make uninformed decisions on how to control their privacy. These privacy control mechanism lead to a situation where users simply install applications without questioning the requested permissions, even if the applications may silently leak private information (Felt et al. 2011c; Vidas et al. 2011; Enck et al. 2010).

To improve the privacy-control mechanism of these mobile platforms, we provide a user-aware-privacy-control approach that reduces efforts for app validation and access-granting by computing information flows and classifying information flows as safe/unsafe.

Our approach automatically computes information flows of private information via static analysis and visualizes the flows, as shown in Fig. 1. We use the term *Source* to



script: *location and maps* ☆☆☆☆☆

created by XXX

Fun scripts using the GPS location

information flow

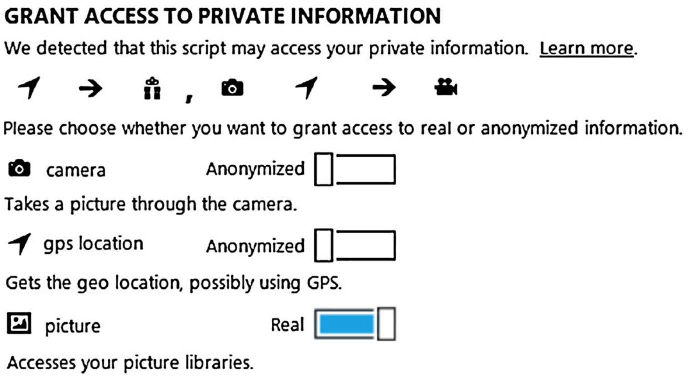
Private information may flow from gps location to sharing and from camera, gps location to media.

Fig. 1 Information flow view of a sample script

refer to an origin of private information and *Sink* to refer to a point where information may leak from an app. The example in Fig. 1 shows that the app uses 5 capabilities (Camera, Location, Pictures, Media, and Sharing). Among these, the first 3 are sources, and the last two are sinks. Among the 6 possible flows (3 sources to 2 sinks), our analysis shows that the Location flows to the Sharing sink, and that Camera and Location flow to the Media sink.

Given the computed information flows, our approach employs the mechanism of user-driven access control (Roesner 2011). When the application is executed for the first time, our approach allows users to choose among *real* information, *anonymized* information, or *abort* execution, as shown in Fig. 2 (the *abort* option is not yet implemented in TouchDevelop). These settings provide flexible choices for users: (1) using anonymized information (e.g., a fixed picture or a fixed geolocation), users can experiment with applications before granting access to real information; (2) aborting an execution prevents unintended access to a resource and is helpful for diagnosis.

To assist experts and users in better understanding how apps handle private information and improve privacy control, our approach further classifies information flows based on a tamper analysis. We define a policy to classify information flows as safe or unsafe: an information flow is safe if only *untampered* private information flows to a *vetted* sink. A vetted sink is a sink that presents an explicit dialog requesting the user's permissions before the information being shown escapes. In Myer's terminology (Askarov and Myers 2010), this dialog corresponds to a declassify step and tampered data has low integrity. For example, in TouchDevelop (2011), the sharing of a picture taken camera shows a dialog for users to review the picture before it leaks from the device. Such information flows do not leak private information without notifying users and should be safe. However, a malicious app could encode the user's phone number into the color intensity of some pixels inside a picture to be shared. The information flow will reveal that private information from the camera and contact sources flow to the share sink, but a user may be hard pressed to recognize any changed pixels in the picture being posted. Our analysis detects such obscure flow by observing whether the information is tampered with before reaching the sinks. Based on the safe/unsafe classification of flows, our policy is to use real information for sources only appearing in safe flows, and anonymized information for all other sources.



GRANT ACCESS TO PRIVATE INFORMATION

We detected that this script may access your private information. Learn more.

Please choose whether you want to grant access to real or anonymized information.

camera Anonymized

Takes a picture through the camera.

gps location Anonymized

Gets the geo location, possibly using GPS.

picture Real

Accesses your picture libraries.

Fig. 2 Grant access to private information

Our privacy-control approach strives for a balance between security and user involvement. By employing user-driven access control, our approach ensures that apps gain permissions from users for private information accessed by apps. To avoid overwhelming the users with access granting—which may annoy users and cause users to blindly grant every permission—our approach does not ask users to grant access to private information accessed by an app but not flowing to sinks. Furthermore, our technique provides default settings that are safe to run a script without further user decisions, thereby reducing risk and user burden.

We build a prototype of our privacy control into TouchDevelop, a novel mobile platform that enables users to write apps directly using touch screens. In TouchDevelop, apps are written using a scripting language that is expressive enough to create applications or games, utilizing most features of mobile devices (Tillmann et al. 2011). We call apps written in TouchDevelop "scripts". Users can publish their scripts in a "script bazaar", where other users can install and run them on their own devices. TouchDevelop is thus similar to other mobile-device platforms, except that we use no manual validation, only automatic information flow analysis. Our approach works well with the TouchDevelop platform for several reasons: (1) all code is made available through the script bazaar as source; (2) the expressiveness of the language enables apps to be created in fewer lines, allowing efficient static analysis on whole scripts; (3) the language does not allow reflection, eval, or native calls to platform APIs, making code analysis easier (Howard 2011).

To evaluate the usability and effectiveness of our approach, we conduct a user survey with TouchDevelop users who have been using the TouchDevelop App. Our survey involves 161 users, and 50 of them answer all the questions. Each participant of the survey is asked to complete questions regarding the effectiveness of different techniques in our approach and provide comments on how to improve our approach.

This paper makes the following contributions:

- We propose a user-aware-privacy-control approach, which reveals information flows and their classification to users to assist app validation and user-driven access control. Our approach is a first step towards improving privacy control on mobile devices via automatic analysis.

- We present an extended static analysis to compute information flows and check tamper information for classifying information flows as safe/unsafe flows.
- The automatic computation and classification of information flows and resuling safe default settings enable our script bazaar to operate *without any manual script validation*.
- We built a prototype of our privacy control into TouchDevelop, both for analyzing published scripts, and to present user privacy settings to the user based on our analysis and policy.
- We studied 546 scripts published by 194 users to evaluate the effectiveness and performance of our information flow analysis. The results show that among the 546 scripts, 172 use a private source, but only 78 scripts (14.29 %) flow private information to a sink. Among these 78 scripts, our approach classifies 24 as safe, reducing the need to make access granting choices to a mere 10.1 % (54) of all scripts. Alternatively, users need to grant access to only 63 sources (41.4 %) among 152 sources appearing in scripts together with sinks.
- We conducted a user survey that involves 50 users who have been using TouchDevelop App built with our privacy control approach. The results show that 90 % of the users consider our approach useful in protecting their privacy, and 54 % prefer our approach over other privacy-control approaches. We also summarized users' feedback on the future improvement of our approach.

2 TouchDevelop language

TouchDevelop allows users to create applications using an imperative and statically typed language (Tillmann et al. 2011). A TouchDevelop script consists of a number of actions (procedures) and global variables. The body of actions consists of: (1) expressions that either update local or global variables (assignments), invoke another action, or invoke a predefined property; (2) conditional statements **if**–**then**–**else**, (3) loop statements, **for**, **while**, and **foreach**, that iteratively execute a block of statements. The global variables are statically typed and their current value is persisted and accessible across multiple script invocations.

As a statically typed language, TouchDevelop defines a number of data types (e.g., **Number** or **String** for s, or **Picture** for p in Fig. 3). Each data type provides a number of properties (e.g., $p \rightarrow$ share). For the sake of the simplicity, the language does not provide features that allow users to define new types or properties.

2.1 Classified information flow

In this section, we illustrate several examples to show how scripts written in TouchDevelop may leak private information (referred to as *classified information*). Figure 3 shows an example of how classified information flows among values, such as **Number** and **String**. At line 4, variable loc becomes classified since it contains the geolocation information obtained via the GPS. Here, we refer to the property senses→current location as a *Source* of geolocation information. At line 5, the location is transformed into a string and assigned to s, thereby making s classified. At line 6, the location string

```
1 action foo() : Nothing {
2       var s := "unclassified";
3       var p := media → create picture();
4       var loc := senses → current location; // classified
5       s := loc → describe(); // classified
6       p → draw text(s); // p's mutable state is classified
7       p → share("facebook");
8 }
```

Fig. 3 Example of classified information flow

```
 1 action foo(msg : Message, msgs: MessageCollection,
        i: Number) : Nothing {
 2     var pic := senses → take camera picture;
 3     pic → share('facebook','share a pic');
 4     var s := currLoc(); // classified
 5     msgs → add(msg);
 6     msg → set message(s); // classified
 7     var msg2 := msgs → at(i); // classified via
           reference−type flow
 8     msg2 → share('facebook');
 9     var y := false;
10     if s → contains('Seattle') then {
11         y := true; // classified via implicit flow
12     }
13 }
14
15 action currLoc() returns r : String{
16     var l := senses → current location; // classified
17     r := locations → describe location(l); // classified
18 }
```

Fig. 4 Implicit and reference-type information flow

s is rendered as text into the picture p, causing p to be classified. At line 7, the share action of p leaks the classified information of the user's geolocation to facebook. Here we refer to the property share as a *Sink*. One thing to note is that if line 5 were moved to after line 6, then p would not be classified. The later update of s would not affect p.

Now let's look at another example shown in Fig. 4. At line 5, the message msg is added to the message collection msgs. The message collection msgs keeps a reference to msg, which means that msg can be accessed from msgs at a later time. At line 6, msg becomes classified, which causes msgs to be classified indirectly. At line 7, msg2, the i-th message in msgs, may contain the information of msg or other messages. Thus, msg2 should also be considered as classified. We refer to this type of information flow as reference-type flow, since it occurs through objects such as message collections that contain references to other objects.

Another type of information flow that can potentially leak private information is implicit flow (Denning 1976; Denning and Denning 1977). Implicit flow arises from conditional control structures such as *if* statements where the condition depends on classified information. The statements in the branches of the conditional statement can leak the outcome of the condition, which allows later code to determine the classified information indirectly. Consider the example of implicit flow shown in Fig. 4. The

Table 1 Capabilities provided by the TouchDevelop APIs

	Capability	Description
Source	Camera	Takes a picture through the camera
	Location	Gets the geo location, possibly using GPS
	Picture	Accesses the picture libraries
	Music	Accesses the music library
	Microphone	Accesses the microphone
	Contacts	Accesses emails or phone numbers of contacts
Sink	Contacts	Saves an email or phone number of a contact to the device
	Media	Saves pictures to the phone
	Sharing	Share information through social services, email or short messages
	Web	Accesses the web, downloading or uploading data

classified local s is used at the **if** statement at line 10. By observing the values of y, users can guess whether the geolocation information stored in s contains the substring Seattle. Thus, to track implicit information flows, we need to consider y as classified.

3 Capability identification

The application capabilities tell users what kinds of mobile-device resources (such as personally-sensitive information and wireless network) an application uses, which is useful information for users to decide whether to install the application. These resources can be classified as sources (such as camera or geolocation) and sinks (such as web or facebook sharing). To use these resources, application developers need to use the APIs provided by the device-specific development environment, also called software development kit (SDK). Table 1 shows the kinds of sources and sinks provided by the TouchDevelop APIs. Among these sinks, the sink *Sharing* prompts users with the sharing information, which makes it a vetted sink. For the other three kinds of sinks, only the sink *Web* is considered as an unvetted sink. The reason is that the pictures from the sink *Picture* and emails or phone numbers from the sink *Contacts* are all considered as sensitive private information, and if these kinds of private information would flow to *Web*, our approach would identify the flow as an unsafe flow.

Automated capability identification To provide the accurate and complete information of what resources are accessed by applications, our approach provides a static analysis that scans through the application script to automatically identify application capabilities. We have manually annotated all TouchDevelop APIs with source and sink information. We use a fixpoint algorithm to compute the capabilities used by each action of a script. For each action in a script, our approach parses the action into an abstract syntax tree (AST), and automatically scans each statement node in the AST to identify what sources and sinks are used. If a statement in an action a_1 is a call to another action a_2, our approach adds the sources and sinks of a_2 to a_1. A fixpoint is reached if the computed sources and sinks for each action do not change.

Since application developers in TouchDevelop can only use the APIs provided by the device-specific SDK for accessing mobile-device resources, our analysis results are guaranteed to be accurate and complete.

4 Information flow analysis

In this section, we first present an overview of our static information flow analysis, and then follow it up with full technical details.

4.1 Overview

Our approach statically computes information flows using abstract interpretation (Cousot and Cousot 1977). Our approach maintains the abstract state of the script and updates the state according to the simulated execution of a statement. The state maps local variables to sets of sources. In addition it maps a single mutable location for each kind[1] to a set of sources. Finally, the state maps *sinks* to sources flowing to that sink. Sinks can be thought of as additional mutable locations that accumulate what flows into them. Information flow from a source s_1 to a sink s_2 arises whenever source s_1 appears in the abstract state of sink s_2. The sources in our maps are represented as a set of value elements consisting of constant sources and input parameter names. Input parameter names are used to represent symbolic information that allows us to determine where parameters flow.

Implicit flows In order to handle implicit flow arising from control flow statements that branch on classified information, we use an additional special local variable named pc. The pc variable is assigned (augmented) with source information at conditionals at the entry of both branches. At each basic block, the pc is defined by the value of pc at the immediate dominator block instead of all predecessor blocks as is the case for normal locals.

Inter-procedural analysis Our approach uses a fix-point algorithm to iteratively compute the summaries of basic blocks in an action and then uses these summaries to compute summaries of actions. At call-sites, summaries are instantiated with concrete values for symbolic parameter names, thereby computing the effect of the call without re-analysis of the action. This approach also handles recursive actions.

Mutable and immutable values We map the TouchDevelop concepts to a simpler model for information flow analysis. We can think of each kind of value as having two separate parts: (1) *an immutable part*, and (2) *a mutable part*. Many types of values have only an immutable part and no mutable parts, e.g., **Number**, **String**, and **GeoLocation**. Other types of values have both immutable parts and mutable parts. E.g., **Picture** has an immutable part that is associated with whether the picture is valid (i.e.,

[1] Data types in TouchDevelop are called kinds.

whether the pointer is null). The mutable part of a picture consists of the actual pixel colors at each coordinate of the picture.

We track information flow separately for the mutable and immutable parts of values. The immutable part of an object is copied whenever a value is assigned from one local to another, passed as parameter, returned from a method, stored or loaded from a global variable. The immutable part of a value is tracked precisely at each program point and assignments are strong assignments that replaces the original values.

The mutable part of an object is affected only by pre-defined property invocations (i.e., primitive methods). We track the mutable part of values using an abstraction where we have a single mutable location per kind. Every value of that type shares that same mutable location in the analysis. All updates to the mutable part are weak updates, meaning they are accumulated.

Primitive properties are annotated with information that indicates from which parameters (and thus which kinds) the mutable state is read, and also what mutable parts are written (parameters and return values).

Embedded references Because values may have embedded references to other values that could be mutable, we also keep track of such embedded references using directed edges from one mutable location to another. The model currently does not accommodate references from immutable parts to mutable parts, but we have not found a need for that. Establishing a reference from one value to another implies a write to the mutable state of the first.

Globals To simplify the description in the remainder of the paper, we eliminate global variables from the model. Global variables are treated as extra parameters and return values from each action. One can easily transform a program with globals to a program without globals by adding all globals used in an action (and actions called) as extra parameters, and all globals modified by an action as extra return values. As a result, inside an action, accessing a global is no different than accessing a local variable. We will thus no longer explicitly talk about global variables henceforth.

Parameters Parameters of an action are treated as ordinary locals inside an action. They are pre-initialized by the action invocation, but otherwise act no differently than normal local variables.

Results Result variables are treated as ordinary locals inside an action. Upon return, their immutable parts (values) are copied to the caller's locals that receive the results of the invocation.

4.2 Simplified language

We assume that our input program consists of a number of actions, where each action has any number of parameters and any number of results. The body of an action consists of a control flow graph of basic blocks, with a distinguished entry block and a distinguished exit block. Conditionals branching on condition `c` are transformed into non-deterministic branches to the **then** and **else** blocks, where the target blocks are augmented with a first instruction of the form assume(c) and assume(**not** c).

The instructions inside a block have the following forms:

$$\begin{gathered} Instruction ::= x := y \mid r := p(x_1..x_n) \\ \mid r_1..r_n := a(x_1..x_m) \mid assume(x) \mid assume(\neg x) \end{gathered}$$

An instruction is either a simple assignment from one local to another, a primitive property invocation of parameters $x_1..x_n$ binding the result to a variable r, an action invocation with parameters $x_1..x_m$ binding the results of the action to $r_1..r_n$, or a special *assume* statement arising from conditional branches. We assume that primitive operations always return a value, even if it is the **Nothing** value.

4.3 Summaries of basic blocks and actions

We separate the state into three parts: (1) local variable information, (2) pc information for implicit flow, and (3) mutable state information. The first two are program point specific, but the mutable state is not. The mutable state consists of one classification per kind, and a set of edges between kinds representing possible references from the mutable state of objects of one kind to objects of another kind.

$$\begin{array}{ll} Atom & ::= Sources(i) \mid Parameter(i) \mid PC_{in} \\ Classification & ::= Set\,of\,Atom \\ LocalMap & ::= Block \rightarrow Local \rightarrow Classification \\ SinkMap & ::= Block \rightarrow Sink(i) \rightarrow Classification \\ PCMap & ::= Block \rightarrow Classification \\ MutableState & ::= Kinds(i) \rightarrow Classification \\ References & ::= Set\,of(Kinds(i) \times Kinds(i)) \end{array}$$

The fixpoint computation computes the following data structures:

$$\begin{aligned} L_{pre}, L_{post} &: LocalMap \\ PC_{pre}, PC_{post} &: PCMap \\ S_{pre}, S_{post} &: SinkMap \\ M_{pre}, M_{post} &: Block \rightarrow MutableState \\ R_{pre}, R_{post} &: Block \rightarrow References \end{aligned}$$

L_{pre} contains the local information on entry to a particular block, whereas L_{post} contains the corresponding information at exit of the block, and similarly for PC_{pre} and PC_{post}. The sink maps S_{pre} and S_{post} contain the classification of the predefined sinks on entry and exit of blocks. M_{pre} and M_{post} contain the mutable state classification and R_{pre} and R_{post} contain the reference links between mutable states.

4.3.1 Block summary

We initialize L_{pre} for entry blocks of actions to map each parameter local i to the singleton $\{Parameter(i)\}$ and to the empty set for all other locals. Similarly, we initialize PC_{pre} for entry blocks to the singleton $\{PC_{in}\}$ which allows computing symbolic summaries of actions that can be applied in contexts where the PC is classified differently. The sink map S_{pre} for the entry block is empty. These maps will not change during the global fix point of the analysis.

The information for R_{pre} and M_{pre} for the entry block keep track under which assumptions the action has been analyzed. It is initially empty, but may grow as the action is invoked in a context with larger M or R, causing the blocks of the action to be re-analyzed.

For non-entry blocks, the starting state is defined as follows:

$$\begin{aligned}
L_{pre}(b) &= \bigsqcup_{b' inpred(b)} L_{post}(b') \\
S_{pre}(b) &= \bigsqcup_{b' inpred(b)} S_{post}(b') \\
M_{pre}(b) &= \bigsqcup_{b' inpred(b)} M_{post}(b') \\
R_{pre}(b) &= \bigcup_{b' inpred(b)} R_{post}(b') \\
PC_{pre}(b) &= PC_{post}(dom(b))
\end{aligned}$$

The locals on entry to a block are simply the union of the post local state of all predecessor blocks, where union is defined point-wise on the map (similarly for the sinks, mutable state, and reference links). For the PC classification is obtained by the post PC classification of the immediate dominator of block b.

4.3.2 Action summary

We assume each action has a single exit block. The summary of an action is simply the post state of the exit block of the action. For each action, we keep track of the initial M and R under which it was analyzed in the information for its entry block. If we see a call to the action with a larger M or R, we update that information for the entry block and propagate the changes through the blocks of the action. For example,the summary of action foo in Fig. 4 is:

$$\begin{aligned}
State =\{ & \\
& L = \{s \rightarrow \{\textbf{Location}\}, pic \rightarrow \{\textbf{Camera}\}, \\
& \quad y \rightarrow \{\textbf{Location}\}, msg \rightarrow \{\textbf{Location}\}, \\
& \quad msg2 \rightarrow \{\textbf{Location}\}\}, \\
& S = \{\textbf{Sharing} \rightarrow \{\textbf{Camera}\}\},
\end{aligned}$$

$$\begin{aligned} &PC = \{\},\\ &M = \{Picture \rightarrow \{\textbf{Camera}\},\\ &\qquad Message \rightarrow \{\textbf{Location}\}\}\\ &R = \{< MessageCollection, Message >\}\ \} \end{aligned}$$

Here the state of locals L shows that the local s contains the geolocation data, pic contains the camera data, y contains geolocation data due to the implicit flow from s to y, and the local msg gets geolocation data from s at line 5. The state of mutable locations M shows that the mutable state of **Picture** contains the camera data and the mutable state of **Message** contains the geolocation data. The state of references R contains a pair showing that **MessageCollection** is linked to **Message**. Due to this link, msg2 reads the mutable data of msgs and is considered to contain the geolocation data. The state of sinks S shows that the sharing sink contains camera data. The set PC is empty, since the pc does not carry the camera data after the **if**–**then**–**else** block.

4.4 Classified information propagation

In this section, we describe how APIs are annotated and how information flow is tracked at the instruction level.

4.4.1 Property annotations

We assume that every primitive property p is annotated with a set $\textbf{ReadsMutable}_p$ consisting of the parameter indices of parameters whose mutable state is read by p. Similarly, the set $\textbf{WritesMutable}_p$ consists of the indices of parameters whose mutable state is written by p. Additionally, we use index 0 in $\textbf{WritesMutable}_p$ to indicate whether the mutable state of the result depends on the classification of the inputs to property p. By default, we assume that all immutable parts of all parameters are read by a property and that all read parts flow into the result's immutable part. Additionally, the set $\textbf{EmbedsLinks}_p$ contains the set of edges between kinds representing possible references established by invoking property p.

A set $\textbf{Sources}_p$ indicates which predefined sources flow into the result value when invoking property p. Finally, $\textbf{Sinks}_p$ contains the set of sinks to which information flows on invoking p.

4.4.2 Statement-based propagation

The following rules show the propagation of the state for each kind of instruction. We assume L, PC, M and R are the initial states, and L', PC', M' and R' are the post states.

Case $x := y$

$$L' = L[x \mapsto L(y) \cup PC]$$

$$
\begin{aligned}
PC' &= PC \\
M' &= M \\
R' &= R \\
S' &= S
\end{aligned}
$$

Note how the PC classification flows into the new classification of x. This is needed to keep track of implicit flow.

Case $r := p(x_1..x_n)$. First we compute the input classification, which consists of the classification of all input parameters, the classification of all kinds for which there is a parameter annotated with **ReadsMutable**.

$$
\begin{aligned}
Common = {} & PC \cup \mathbf{Sources}_p \cup \bigcup_i L(x_i) \\
& \cup \bigcup_{j \in \mathbf{ReadsMutable}_p} Cl(M, R, kind(x_j))
\end{aligned}
$$

The helper function $Cl(M, R, i)$ computes the union of the classification of all kinds j reachable from i via edges in R. Note that $Reach(R, i, i)$ is true for all R.

$$Cl(M, R, i) = \{M(j) \mid Reach(R, i, j)\}$$

With this information, we update the result and the mutable state.

$$
\begin{aligned}
L' &= L[r \mapsto Common] \\
PC' &= PC \\
M'(i) &= \begin{cases} M(i) \cup Common & \text{if } \exists j \in \mathbf{WritesMutable}_p \\ & \text{and } Reach(R, kind(x_j), i) \\ M(i) & \text{otherwise} \end{cases} \\
R' &= R \cup \mathbf{EmbedsLinks}_p \\
S'(i) &= \begin{cases} S(i) \cup Common & \text{if } i \in \mathbf{Sinks}_p \\ S(i) & \text{otherwise} \end{cases}
\end{aligned}
$$

Case $assume(x)$ or $assume(not\ x)$.

$$
\begin{aligned}
L' &= L \\
PC' &= PC \cup L(x) \\
M' &= M \\
R' &= R \\
S' &= S
\end{aligned}
$$

Assume statements cause the PC classification to be augmented with the classification of the condition.

Case $r_1..r_n = a(x_1..x_m)$. First, we update $M_{pre}(entry_a)$ to $M \sqcup M_{pre}(entry_a)$ and $R_{pre}(entry_a)$ to $R \sqcup R_{pre}(entry_a)$. If necessary, propagate changes through blocks of a. We use the state at the exit block of a as the summary of a to be applied at the current invocation. Since the summary contains some symbolic information for parameter classification and pc classification, we first instantiate the exit block information with the invocation site information. Let σ be the substitution

$$\sigma = [PC_{in} \mapsto PC, Parameter(i) \mapsto L(x_i)]$$

Now we compute instantiated versions of the exit block summaries:

$$\begin{aligned} L_s &= \sigma(L_{post}(exit_a)) \\ M_s &= \sigma(M_{post}(exit_a)) \\ R_s &= \sigma(R_{post}(exit_a)) \\ S_s &= \sigma(S_{post}(exit_a)) \end{aligned}$$

Note that no PC information flows out of the action. Let $r'_1..r'_n$ be the result locals in action a. The final states after the invocation of action a is then:

$$\begin{aligned} L' &= L[r_i \mapsto L_s(r'_i)] \\ PC' &= PC \\ M' &= M \sqcup M_s \\ R' &= R \cup R_s \\ S' &= S \sqcup S_s \end{aligned}$$

5 Tampered information

The source to sink information flow we compute so far may not be enough to make good policy decisions about which scripts are good and which scripts are bad. For example, a script taking a picture with the camera and then posting it to facebook may be a reasonable script, especially since posting to facebook will prompt the user and display the text and picture that will be posted. The user thus has a way to *vet* the information being posted.

However, a malicious script could try to encode the user's phone number into the color intensity of some pixels in the posted picture. From an information flow perspective, we would simply see that sources Camera and Contacts flow to Sharing. Users looking at the picture being posted will likely not notice changed pixels containing the hidden phone number.

Can we distinguish somehow between these two cases? Our attempt to do so is based on the following assumption: for sinks that prompt the user to review the information (e.g., emails, sms, phone calls, facebook posts), we want to distinguish if the information being posted is recognizable by the user as containing sensitive information or not. In the case where pixels in the picture taken by the camera are modified based on classified contact information, we want to consider the information in the picture as tampered and thus apply a harsher policy than if the information is not tampered with.

In order to track tampering, we introduce an operator $Tamper$ that can be applied to the existing sources.

$$\begin{aligned} Atom ::= {} & Sources(i) \mid Parameter(i) \\ & \mid PC_{in} \mid Tamper(Atom) \end{aligned}$$

Note that the set of atoms is not unbounded, as this is not a free algebra. Indeed, $Tamper(Tamper(s)) = Tamper(s)$ for all s. Additionally, we annotate all properties p with a single bit $\textbf{Tampers}_p$, indicating whether any input classifications are transformed into tampered output classifications for the result and writes to the mutable store.

The rule for handling the flow at property invocations then needs to be modified insofar as the classification $Common$ now becomes:

$$\begin{aligned} InFlow = {} & PC \cup \bigcup_i L(x_i) \\ & \cup \bigcup_{j \in \textbf{ReadsMutable}_p} Cl(M, R, kind(x_j)) \\ Common = {} & \textbf{Sources}_p \cup \begin{cases} InFlow & \text{if } \neg\textbf{Tampers}_p \\ Tamper(InFlow) & \text{if } \textbf{Tampers}_p \end{cases} \end{aligned}$$

Applying $Tamper$ to an entire classification, just means applying the operator pointwise to the set elements.

6 User-aware privacy control

By applying the static analysis, we compute information flows on a per action and per script basis and show summaries of which sources flow to which sinks in each action and in the script as a whole. As an example, Fig. 1 shows the summary of the script named *location and maps*, which can send a text message containing the user's current location or take a picture with the user's current location embedded in it and save the picture into the media storage library of the mobile device. This flow summary shows the information flows of the application: by looking at the information flows at install time, users can understand what private information the application uses and where this private information may escape to. To minimize the efforts of experts in validating applications and users in granting accesses to sources, we further define a policy that classifies flows into safe and unsafe flows.

Classification of safe and unsafe flows Our policy is based on the assumption described in Sect. 5: we consider a flow as a *safe flow* if it is an untampered flow to a *vetted sink*. Recall that a vetted sink results in an explicit dialog at runtime, presenting the particular information flowing to the sink and requesting permissions from the user before the information escapes from the mobile-device. For example, a post to facebook would prompt the user to review the information before the actual sharing happens. Our approach considers all other flows as unsafe, including untampered flows to unvetted sinks (Web) and all tampered flows. We may evolve the policy of what constitutes a safe flow based on user feedback, and update the policy when more sources and sinks are added into the system.

Granting accesses When running the script for the first time, the user is presented with all sources appearing in information flows along with a radio button group for each source that allows the user to choose among *anonymized* or *real* information (Fig. 2). Anonymized information means that the runtime provides the script with anonymized information (a fixed picture or a fixed geolocation etc.), real information means the script gets access to the real information on the users' device, and abort execution means that the runtime stops the execution at the access point. By using anonymized information, a user can safely experiment with an application to determine if it does something useful prior to even considering whether to allow access to real information.

Default settings To keep users safe and minimize efforts in granting access, our approach provides default settings. We guarantee that running a script with the default settings does not leak private information, except through vetted sinks where the user is presented untampered information to review. Sources appearing in no flows use real information and are not shown. For sources that appear only in safe flows, the default setting is to use real information; for other sources appearing in flows, the default setting is to use anonymized information.

7 Evaluation

This section presents experiments we conducted to evaluate the effectiveness and performance of our extended static information flow analysis. We chose TouchDevelop as a platform for our evaluations due to three major reasons: (1) **privacy concerns**: scripts written in TouchDevelop can access private information on the phone, and scripts can be downloaded easily by other (unsuspecting) users through the script bazaar; (2) **source code availability**: the source code of a script is made available as part of the publishing process; (3) **simplicity**: the expressiveness of the TouchDevelop languages enables applications to be created in much fewer lines, reducing the complexity of static analysis; the TouchDevelop language does not allow reflection or native calls to platform APIs, enabling complete annotation of the APIs with source, sink, and flow information; TouchDevelop only allows importing of external scripts through the script bazaar and does not allow generating code at runtime.

7.1 Subjects and evaluation setup

To conduct the experiments, we collected 546 scripts (all publications prior to Oct 6th, 2011) published by 194 TouchDevelop users, excluding scripts published by ourselves.

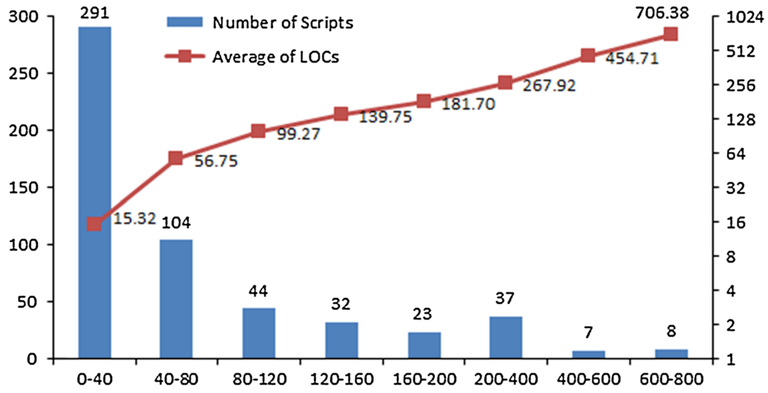


Fig. 5 Sizes of 546 published scripts in TouchDevelop

Figure 5 shows the number of scripts in different ranges of lines of code (LOC) [2] and the average LOCs in these ranges. Among these scripts, 395 (72.34 %) scripts have LOCs ranging from 0-80, and the scripts *Termini 3 Final*[3] and *Termini 3 Beta 1.4*[4]) have the maximum LOCs of 738. The major reason why these scripts are of relatively small size is that the expressiveness of the TouchDevelop language enables users to create applications using fewer lines of code than using traditional programming languages for mobile devices. For example, the script *Termini Include Edition 1.0.2*[5] published by the user Pouya Animation[6] creates a UNIX emulator (Terminal) for TouchDevelop in just 407 LOC.

7.2 Information flow evaluations

To show the effectiveness of our information flow analysis, we posed the following three research questions about the 546 subject scripts:

- **RQ1**: What is the advantage of using information flow from sources to sinks to classify scripts, as opposed to the mere presence of both sources and sink (capability usage)?
- **RQ2**: How many more scripts can we classify as safe using our tamper analysis, thus eliminating the need to ask users to grant access?
- **RQ3**: How many more sources can we classify as safe using our tamper analysis, further reducing the number of sources that require users' decisions?

[2] Meta data and comment statements are excluded for LOC computation.

[3] http://touchdevelop.com/pycw

[4] http://touchdevelop.com/xwgl

[5] http://touchdevelop.com/hllw

[6] https://www.touchdevelop.com/ntqe

Table 2 Information flow summary of 546 published scripts

# Total	# Cap (242)			# Flow
	/w Source	/w Sink	/w Both	
546	172	159	89	78

```
 1 action OBRE(OpenBarcode1: Picture) returns pic: Picture {
 2     ...
 3     pic := media → create picture(480, 120);
 4     if SN → to number = 010010 then {
 5         pic → draw text(0, 0, 'Comicbar 2', 80, 0, color → accent);
 6         pic → draw text(0, 85, 'Serial Number: ' // SN, 30, 0, colors → white);
 7         web → browse('http://comicbar.basekit.com');
 8     }
 9     ...
10 }
```

Fig. 6 Code snippet of *OpenBarcode 1.0.2*

7.2.1 RQ1: Information flow summary

To address RQ1, we compare the number of scripts that are classified as information-leaking using information flows with the number of scripts that are classified as information-leaking using capabilities. Table 2 shows the information flow summary of the published scripts. Column "# Total" shows the total number of scripts. Column "# Cap" shows the number of scripts that either have at least one source or one sink. Column "/w Source" shows the number of scripts that have at least one source. Column "/w Sink" shows the number of scripts that have at least one sink. Column "/w Both" shows the number of scripts that have both sources and sinks. Column "# Flow" shows the number of scripts that have computed information flows.

The results show that in 546 published scripts, 242 (44.32 %) either have sources (access private information) or have sinks (can leak information from the script). To form an information flow, a script must have at least one source and one sink. As shown in Table 2, 457 (83.70 %, #Total − #Both) scripts have either no sources or no sinks, which can be classified as non-information-leaking by either using information flow or capabilities usage. For the remaining 89 scripts that have both sources and sinks, our information flow analysis detects that 11 scripts have no information flows. Thus, using potential flow (presence of both source and sink), reduces prompting by 48.26 % (from 172 to 89) over the traditional capability approach (presence of sources). Using actual information flows, as computed by our analysis, further reduces prompting by 12.36 % (from 89 to 78).

Example OpenBarcode 1.0.2[7] is a script using the Picture source and the Web sink, but without actual information flow (Fig. 6). At line 4, variable SN contains private information from the user's picture library (not shown in Fig. 6). Due to indirect flow in the **if**-branch, variable pic becomes classified. However, at line 7, the sink web→browse

[7] https://www.touchdevelop.com/hejn

Table 3 Information flow vs. source-sink pairs

	Contacts	Media	Sharing	Web	*Any*
Camera	0/0	22/22	11/ 21	0/30	33/36
Contacts	1/3	0/11	30/ 39	0/19	30/41
Location	0/0	6/10	12/12	27/29	30/34
Microph.	0/0	0/2	1/1	0/3	1/6
Music	0/0	0/1	1/1	0/1	1/3
Picture	0/0	18/29	2/15	14/21	24/32
Any	1/3	29/39	44/48	40/51	78/89

uses a constant string that does not depend on classified information. Thus, there is no information flow from the Picture source to the Web sink in this script and users need not grant explicit access to pictures.

Table 3 shows the information flow summary of the published scripts based on source-sink pairs. Each column represents a kind of sink and each row represents a kind of source. The first number in each table cell is the number of scripts for which our analysis determines information flow from the given source to the given sink, whereas the second number in each cell is simply the number of scripts that use the corresponding source and sink. The two numbers presented in each table cell compare our approach of computing actual information flow, to a naïve capability analysis that simply presumes an information flow for each used source-sink pair.

For example, the cell for Camera and Web shows that a naïve capability approach would classify 30 scripts as having information flow from the camera to the web, whereas our information flow analysis proves that none of these scripts actually leak camera information to the web, completely removing the concerns of leaking pictures taken from the camera through the web.

Similarly, the naïve capability approach would consider 19 scripts to leak contact information through the web, while our analysis shows that none of these scripts would do that.

These results show that information flow analysis effectively computes a much finer granularity of the potential flows between sources and sinks used in a script.

7.2.2 RQ2: Safe scripts

To address RQ2, we apply our static analysis on the 78 subject scripts that have information flows, referred to as *flow scripts*, and measure the number of flow scripts that have safe flows. We assume only sink *Web* is an unvetted sink, while all others are vetted sinks. Table 4 shows the safe/unsafe flow summary of the 78 scripts that have information flows. Column "# Safe" shows the number of scripts that have safe flows. Column "# Unvetted" shows the number of scripts that have information flows from sources into unvetted sinks. Column "# Tampered" shows the number of scripts that have tampered information flows. Column "# Both" shows the number of scripts that

Table 4 Safe/unsafe flow summary of 78 flow scripts

# Safe	# Unsafe (54)		# Both	# Mix
	# Unvetted	#Tampered		
45	40	47	21	0

have both safe and unsafe flows. Column “# Mix” shows the number of scripts that have both safe and unsafe flows from a common source (*mix scripts*).

The results show that 45 (57.69 %) flow scripts have safe flows and 54 (69.23 %) flow scripts have unsafe flows. Among these 54 unsafe flow scripts, 40 flow scripts have flows from sources into unvetted sinks and 47 have tampered information flows. Based on this safe/unsafe flow summary, we know that 24 (#Safe − #Both), or 30.77 % of flow scripts have only safe flows. For these 24 scripts, users are perfectly safe to use the scripts granting full access to private information without prompting or reduced functionality.

Among the 21 flow scripts that have both safe and unsafe flows, none are mix scripts. In all the TouchDevelop scripts, only 2 flow scripts published by ourselves (*tag stuff*[8] and *location and maps*[9]) have both safe and unsafe flows from a common source to sinks. Our current access granting allows users to grant access based on sources only, instead of flows. Users cannot choose real information for one flow and anonymized information for another flow from the same source. As we found only 2 scripts where this limitation matters, our approach seems to be a good trade-off that avoids giving users too many choices.

7.2.3 RQ3: Safe sources

To address RQ3, we look at how many times a user would have to change the default setting for a source if she were to give full access to all scripts. Table 5 shows the total number of times a source appears in a given context. Column “Naïve” shows the number of scripts that use this source and any sink. Column “Flow” shows the number of scripts that have information flows from this source to any sinks. Column “Safe” shows the number of scripts for which this source is safe. The last three columns explain why some flows are unsafe. Column “Unvetted” shows the number of scripts where information flows from this source to unvetted sinks. Column “Tamper” shows the number of scripts where information from this source is tampered before it reaches a sink. Column “Both” shows the number of scripts that have common sources in Columns “Unvetted” and “Tamper”.

Among 33 scripts that have source Camera appearing in flows, 24 scripts (72.73 %) have source Camera as a safe source and 9 scripts (27.27 %) have source Camera in tampered flows. Similarly, 25 scripts (83.33 %) have safe sources of Contacts, leaving only 5 scripts having source Contacts appearing in tampered flows.

[8] https://www.touchdevelop.com/qdjt

[9] https://www.touchdevelop.com/qvci

Table 5 Categorization of sources

	Naïve	Flow	Safe	Unsafe due to		
				Unvet.	Tamp.	Both
Camera	36	33	24	0	9	0
Contacts	41	30	25	0	5	0
Location	34	30	0	27	26	23
Microph.	6	1	1	0	0	0
Music	3	1	0	0	1	0
Picture	32	24	6	14	15	11
Total	152	119	56	41	56	34

In summary, our analysis detects that 47.06 % (56) of 119 sources are safe sources. These safe sources are allowed to use real information directly based on our default settings, eliminating the need for access granting. Among the remaining 63 unsafe sources (# Unvetted + # Tamper − # Both), 7 (# Unvetted − # Both) are solely due to flow to unvetted sinks, and the remaining 56 sources appear in tampered information flows. These results show that using the naïve classification, a user would have to make 152 changes to settings to use real data in all scripts. Using information flow alone, this number is reduced to 119 changes. Using tamper analysis and vetted sinks in addition to information flow, our approach reduces the burden to 63 changes to settings, an overall reduction of 58.6 %.

Effort reduction In the 78 flow scripts, averagely each script has 1.5 sources for users to make decisions if using the information flow approach (119). Thus, the effort reduction of our approach is also about 50 % effort reduction for each script. Also, Table 4 shows that our approach identifies 24 scripts having only safe flows. Since safe flows do not require users' decisions, our approach completely eliminates the need of users' decisions for these 24 scripts. Such effort reduction is useful for users since the users typically install many applications, and have to make decisions each time they install an application.

In Android, users typically make one decision for all permissions of an application. Although our approach detects more potential privacy leaks, our approach may require users to make multiple decisions for permissions in unsafe flows for an application. Therefore, reducing as much manual effort as possible is important in our approach, and our evaluations show promising results in reducing more than 50 % of user decisions compared to naïve classification that uses source-sink pairs.

7.3 Performance evaluation

To analyze all the scripts submitted by TouchDevelop users, the static analysis must have acceptable performance even if it can be deployed in cloud servers. We apply our static analysis on the evaluation subjects to compute information flows and record the

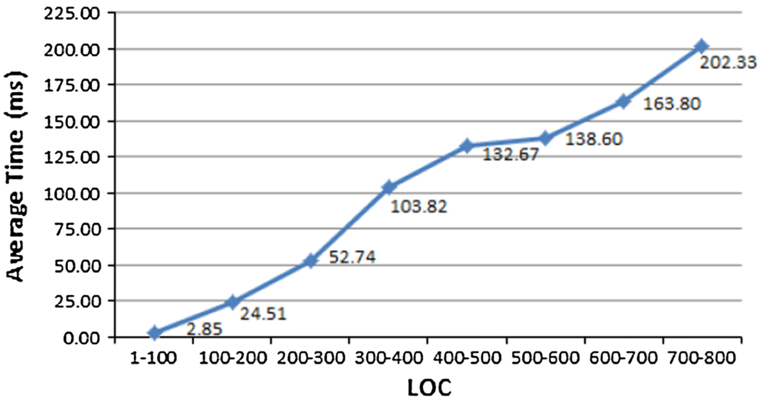


Fig. 7 Analysis time of 546 published scripts in TouchDevelop

analysis time for each script. The performance evaluation is conducted on a Windows 7 x64 PC, with Intel Core 2 Quad CPU Q9400 at 2.67GHz and 4 GB memory. We repeat this performance evaluation 5 times and compute the average analysis time. The average analysis time for analyzing all 546 scripts is 8144 ms, and the maximum average analysis time of a script (*Termini 3 BETA 1.2*[10], whose LOC is 708) is 282 ms. Figure 7 shows the average analysis time for scripts of different sizes. Based on Fig. 7, the analysis time increases linearly based on LOC. These results suggest that our analysis can further scale to larger TouchDevelop scripts.

8 User survey

To understand the usability of our privacy-control approach and to guide future improvement of our approach, we conduct a user survey involving more than 150 TouchDevelop users. We ask each participant to answer 8 questions that assess the effectiveness and usability of our approach. In particular, we seek to answer the following research questions:

- **RQ1**: Do users find the extra information provided by information flow useful in helping them understand more about how scripts will use their personal information?
- **RQ2**: What do users like about our privacy-control approach?
- **RQ3**: Do users think our privacy-control approach is effective in protecting their privacy, and whether our approach is better than other privacy-control approaches? And what should we do to improve our approach?

[10] https://www.touchdevelop.com/eooo

Table 6 50 participants' knowledge of privacy-control mechanisms

Platform	# Users
iOS	16 (32 %)
Android	17 (34 %)
Windows phone	40 (80 %)
Windows 7/8	34 (68 %)

8.1 Study setup

Our static-information-flow analysis is integrated into the server part of the TouchDevelop environment, and every submitted script is analyzed automatically. The resulting flow information is shown to users in the script installation page. The TouchDevelop App also presents all computed flows to users the first time an installed script is run and suggests safe default settings.

Our analysis has been running for nearly 2 years, and computes information flow information for about 17,000 scripts published by more than 4,000 users (Li et al. 2013). To collect feedback on our approach, we conduct a user survey with the TouchDevelop users who have been using the TouchDevelop App v2.11 for Windows Phone, since our privacy control is not fully supported for the web version of the TouchDevelop App.

We design a survey that consists of 3 general questions to ask users about their emails and familiarities with different privacy-control approaches, and 8 specific questions to ask users about their feedback on our privacy-control approach. We distribute the survey by prompting a link to our web-based survey via the TouchDevelop App and Facebook. The survey starts on July 1, 2013. Users who complete all the questions enter a raffle for a $20 Visa gift card, and we randomly select 10 users who complete the survey by July 31, 2013 as winners. Using a random raffle rather than giving awards to every user who fill the survey greatly reduces the incentives for users who never use the TouchDevelop App to fill the survey for winning gift cards, mitigating the threat of the subject representative for our survey.

Participants. After the raffle ends, we keep the survey links active and there are still more users who fill the survey to provide us feedback. For this paper, we collect the results on Aug 19, 2013. In total, there are 161 users who participate in our survey and 50 of them answer all the questions. Among these 50 users who complete the survey, 38 users enter the survey via the web link distributed through the TouchDevelop App and 12 users using the link posted on Facebook. This result indicates that the participants of our survey are mostly users who have been using the TouchDevelop app.

Among these users (Table 6), 40 of them (80 %) are familiar with the privacy-control mechanism of Windows Phone and 34 of them (68 %) are familiar with the privacy-control mechanism of Windows 7/8, while about 30 % of them are familiar with the privacy control mechanisms of iOS and Android. Such distribution is expected, since most of the participants are users of the TouchDevelop app that is available in only Windows Phone. The privacy-control mechanism of Windows Phone includes a permission list that lists the permissions required by an app during installation (the same as Android), and pops up a dialog to explicitly ask for permissions that access

Table 7 Survey questions

RQ	ID	Type	Question
RQ1	Q1	Single-select	When you try to install a script, how often do you inspect the capabilities requested by the script?
	Q2	Single-select	How often do you find the information flows useful in helping you understand more about how the script will use your personal information?
RQ2	Q3	Multi-Select	What do you like best about setting of granting access? – Pop-up dialog that show the information flows – Granting anonymized/real data to different permissions
	Q4	Multi-select	What do you like best about experimenting the script using anonymized data? – Anonymized data that allows you to experience the script without exposing your real information – Change the privacy setting anytime in the script's privacy tab
	Q5	Multi-Select	What do you like best about such privacy control mechanism? – Show only capabilities involved in information flows – Classify information flows as safe/unsafe flows – Default setting of using real information for safe flows and anonymized data for unsafe flows
RQ3	Q6	Single-select	Do you think that the privacy control mechanism in TouchDevelop is useful in protecting your privacy?
	Q7	Text-ranking	How would you rank the following privacy control mechanisms: – User-aware privacy control: TouchDevelop – Permission list: Android and iOS – Popup dialog: iOS and Windows
	Q8	Comment	Do you have any suggestion on improving the current privacy control mechanism in TouchDevelop?

the users' sensitive private information (the same as iOS). Thus, although most of these users are not familiar with the mobile platforms other than Windows Phone (i.e., iOS and Android), these users are familiar with their privacy control mechanisms (i.e., prompting dialogs and permission lists).

8.2 Survey design

Our survey consists of questions that ask users about their experiences of using our privacy-control approach, and how they may compare our approach with other existing approaches.

Table 7 shows the major questions and the corresponding RQs. Every single-select question and multi-select question have an option *Other*, which allows users to provide free-form answers. For the single-select questions of RQ1, users can select one of the 6-scale ratings: *Always (1)*, *Often (2)*, *Occasionally (3)*, *Rarely (4)*, *Never (5)*, and *I did not know about this feature (6)*. Users can provide comments if they select the

Table 8 Results of RQ1: *Information-flow visualization*

Question	*Always (1)*	*Often (2)*	*Occasionally (3)*	*Rarely (4)*	*Never (5)*	*No Idea (6)*
Cap.	17 (34 %)	17 (34 %)	10 (20 %)	4 (8 %)	1 (2 %)	1 (2 %)
Info.	16 (32 %)	16 (32 %)	10 (20 %)	1 (2 %)	4 (8 %)	3 (6 %)

rating of 4, 5, or 6. For the multi-select questions of RQ2, besides options listed in the Table 7, users can also select *None* as the answer. For the single-select question of RQ3, users can select one of the 7-scale ratings: *Very useful (1)*, *Useful (2)*, *Somehow useful (3)*, *Don't Care (4)*, *Not that useful (5)*, *Ineffective (6)*, and *Not at all (7)*. Users can provide comments if they select the rating of 7.

8.3 Study results

In this study, we consider answers from 50 users (out of 161 users) who complete all the questions, since these users are more likely to be users who have been using our privacy-control approach for some time and would like to spend time in providing feedback. Overall, we have received positive feedback on different aspects of our approach, and obtain insightful feedback to guide future improvement of our approach. We next present the results of the survey.

8.3.1 RQ1: Information-flow visualization

RQ1 focuses on the assessment of the visualization of information flows during script installation. Table 8 shows the results of the two questions (Q1 and Q2) for RQ1, with the first row for the ratings of capabilities and the second row for the ratings of information flow.

Based on the results, we can see that more than 60 % of the users would inspect the capabilities and information flows before installing scripts. Such results show that most of the users are aware of the risks of leaking their private information by installing some scripts, and would like to know *what* and *how* these scripts may use their private information. These results provide positive supports for showing extra information such as capabilities and information flows during script installation.

The remaining users inspect only capabilities (32 %) or information flows (36 %) occasionally, or do not look at them at all. By investigating their comments, we know that some of them do not install many scripts and rarely encounter scripts that have information flows; some of them state that they do not care about their privacy. Such results show that there are a portion of users who would simply install scripts without questioning the requested private information, consistent with findings from previous studies (Felt et al. 2011c; Vidas et al. 2011; Enck et al. 2010). Thus, a safe default setting is necessary in the first place to protect such users. Moreover, users tend to ignore information flows more often than capabilities. Such tendency may imply that more information may overwhelm users, and users may not understand what information flows are used for.

8.3.2 RQ2: User-aware privacy control

RQ2 focuses on the assessment of different techniques proposed by our privacy-control approach.

First-time access granting When users first run a downloaded script, a dialog of granting accesses pops up if there is any information flow that may leak the users' private information. To allow users to safely experiment with a script, users can grant real or anonymized data to different permissions. We design Q3 to gather feedback on our access granting.

The results of Q3 shows that 82 % (42) of the users like the setting of granting different access to sources, while only 40 % (20) of the users like the setting of showing information flows in the dialog. Such results show that although most of the users enjoy the flexibilities enabled by allowing real/anonymized data, they may not always use the information flows to help them make the decisions of granting accesses. This result implies that presenting information flows is not intuitive or informative enough under the context of granting accesses to sources. There are also 2 users like neither of these two settings because they state that they do not care about privacy. We may consider to provide an option that allows users to turn off pop-up dialogs and always use the default setting.

Changable settings Unlike Android where users have to grant all permissions at the installation time, our privacy-control approach allows users to freely change the granted accesses even after script installation. Note that the settings of our approach allow either real or anonymized data for each capability (i.e., permission in Android) that is involved in any information flow. We design Q4 to gather feedback on the changable settings.

The results of Q4 show that 40 % (20) of the users enjoy the flexible setting of changing the granted accesses anytime, and 74 % (34) of the users vote for the settings of granting real or anonymized data. Such results imply that only a portion of the users have the needs of making multiple changes of the granted accesses; the setting of real or anonymized data is still welcome by most of the users due to its flexibility and safety guarantee.

Safe/unsafe flows and default settings When users are asked to grant accesses to sources for a script, only sources involved in information flows are shown to users. The computed information flows of the script are then classified by our approach as safe and unsafe flows. To minize users' efforts in granting accesses, for sources that appear in only safe flows, the default setting is to use real information; for other sources, the default setting is to use anonymized information. We design Q5 to gather feedback on the safe/unsafe-flow classification and our default settings.

The results of Q5 show that about half the users like the safe/unsafe-flow classification (27, 54 %) and our default settings (25, 50 %), and only 32 % (16) of the users like the approach of showing only sources involved in information flows. The benefits of all the three techniques studied by Q5 are shown in Sect. 7. These benefits cannot be easily presented to users, and thus lower rates for these techniques are expected.

Thus, the results of Q5 are good enough to indicate that users appreciate the benefits brought by these techniques.

However, the technique of showing only sources involved information flows receives a surprisingly low rate—32 % (16). Such a low rate has two implications: (1) the additional concept of filtering sources based on information flows overwhelms users and they tend to ignore it; (2) users may want to use anonymized data for sources that do not appear in information flows, and thus our current setting should be improved to allow such flexibilities. Moreover, for those users who do not care about their privacy, the default settings provide a safety guarantee for them to safely ignore access granting.

8.3.3 RQ3: Overall assessment and future improvement

RQ3 focuses on the assessment of our privacy-control approach in general, and how users may compare our approach with existing approaches.

The results of Q6 show that 90 % (45) of the users consider our approach useful or very useful in protecting their privacy, and the results of Q7 show that 54 % (27 = 9 + 18[11]) of the users rank our approach to be better than other privacy approaches based on permission lists (Android and iOS) and pop-up dialogs (iOS and Windows). The results show that most of the users (90 %) give positive feedback on our privacy-control approach, and prefer our approach over other approaches in terms of usability and effectiveness. Such results also indicate that those users who do not even look at capabilities or information flows consider our approach useful as well. The main reason is that our default settings allow these users to safely run scripts without making any access choice, providing protection on their privacy and yet preserving usability.

Q8 asks users to provide comments on improving our approach, and we receive insightful comments from 20 users. We next present summaries of these comments.

Centralized/specific settings Our current default setting acts as a centralized setting for every script, and specific settings for each script are presented to users when the script is run for the first time. One user suggests that *"Having a centralized 'default settings', and being able to change them for specific apps, would enhance the experience. Set settings once and any deviation is set on a case by case basis."* Such suggestion indicates that users may be bored by granting accesses for every script, especially when these scripts share the similar sources or information flows. To improve our current default setting, we can make default setting configurable, and allow users to set finer-grain rules. For example, a rule may instruct the default setting to always use anonymized data for information flows from contact to sharing. Users can change such settings freely and can provide specific settings for each script.

Summary of access granting When the number of installed scripts is large, users may have difficulties in maintaining their settings of granting accesses. One user suggests

[11] Due to a bug of the online survey system, we lost part of the data for Q7. We can recover only part of the results based on our best efforts, and cannot provide exact rankings for the other two privacy-control approaches.

that *"It might be helpful to have an audit feature that, on user request, would summarize all of the permissions given and allow a quick review."* Such quick review can also allow users to easily see settings of scripts that have similar sources and help them make better decisions on access granting.

Support for file access One limitation of our approach is lack of support on addressing another type of implicit flow, covert channels (Shieh and Gligor 1990). One user points out that *"Should be better if touchdevelop scripts can ask for permission like iOS does when accessing local files. I mean when the script runs 1st time in a phone."* To address this issue, we plan to investigate the supports of dynamic taint tracking of files from the underlying operating system (Enck et al. 2010).

9 Threat to validity

We next summarize main threats to validity to our research findings.

Threats to external validity The threats to external validity for the evaluation of our approach include the representativeness of the subject scripts and the TouchDevelop platform used by the current implementation of our approach. As shown in Sect. 7 and the recent study (Li et al. 2013), TouchDevelop scripts are typically small, but there are still relatively large scripts with more than 500 LOC. The major reason is that TouchDevelop APIs include rich sets of convenient methods and wrappers for common mobile usage on sensors and touchscreen-related actions, enabling users to create applications using fewer lines of code. Moreover, TouchDevelop users create and publish scripts that cover various categories, such as games, utilities, and tools. Rich sets of APIs used in various kinds of scripts and abilities to publish and share scripts make TouchDevelop similar to other mobile-device platforms. To reduce the threat of subject representativeness, we collect 546 scripts published by 194 TouchDevelop users after the TouchDevelop App was released for several months. The threat could be further reduced by implementing our approach in more types of mobile devices.

The threats to external validity for our user study is the representativeness of participants. Our survey targets at users who have experiences of using the TouchDevelop App, and we consider only users who complete all the questions in the survey. To reduce the threat, users who complete the survey by July 31, 2013 enter a raffle, instead of getting awards every time they fill the survey. Such raffle greatly reduces the incentives for users who fill the survey just for wining the gifts and prevents them from spamming our survey.

Threats to internal validity The threats to internal validity include tool faults in computing information flows and human errors for interpreting the results of the evaluations for information flows and the answers given by participants in the user survey. To reduce the tool faults, we provide unit tests on every function in the computation of information flows, and achieve nearly 100 % of statement coverage. We also provide invariants and pre/post conditions to verify the developed tool. After the tool is deployed to the server part of the TouchDevelop environment, we further collect bugs reported by users and fix them in subsequent versions. To reduce the human-error threats, we ensure that the results are individually verified and agreed upon by at least two authors.

10 Discussion and future work

In this section, we discuss generalizations and limitations of our approach.

Information flow characteristics in TouchDevelop and other mobile platforms In our approach, information flows show a summary of what data types flowing to what output channels, as shown in Fig. 1. These data types represent information and resources proteced by permissions in a mobile platform. Although mobile applications may have many permissions, only some permissions protect sensitive and user-understandable information and resources, and our approach is designed to identify the information flows for such permissions. In order words, the data type in the sources must be an information source or resource that is in the domain of knowledge of general smartphone users, as opposed to a low-level API known to only developers. The data types studied in this paper (i.e., Contact, Microphone, Location and so on) fall within this domain. Table 3 shows the summary of the information flows for the TouchDevelop scripts.

In Android (Enck et al. 2010, 2009; Felt et al. 2011b), the sources that contain sensitive information and the sinks that may leak sensitive information are the similar to the ones shown in this article. According to recent studies of information flows on Android market applications and malware, there are only a handful of information flows that are prevalent in popular Android applications and have potential privacy violations: (1) sending sensitive information such as phone information (e.g., IMEI) and location to remote servers, and (2) saving sensitive information in log files and preference files. Our information-flow techniques can detect all these malicious behaviors by customizing the capabilities (sources and sinks shown in Table 1) to the Android platform. Also, Felt et al. (2011b) classify 46 pieces of malware in iOS, Android, and Symbian into 7 distinct categories by their behaviors, and our information-flow techniques can detect the malicious behaviors from 4 major categories out of the 7 categories: exfiltrating user information, premium calls or SMSs, sending SMS advertising spam, and exfiltrating user credentials. These 4 major categories account for about 60 % of the malicious behaviors in their study.

The granularity of our information-flow techniques can be adjusted to detect more detailed information flows if needed. Based on the results shown in Table 4, there are very few information flows (2 out of 78) that require finer granularity of information flows, since these flows have both safe and unsafe flows from a common source to sinks. The studies of Android and other mobile platforms (Enck et al. 2010, 2009; Felt et al. 2011b) also show that there is little need for detecting finer granularity of information flows.

Generalization to other mobile-device platforms To generalize our approach to other mobile-device platforms, such as Windows Phone, Android, and iOS, several points need to be addressed: (1) these platforms provide a much larger API surface than TouchDevelop and annotating these APIs with source, sink, and flow information is a major effort, (2) the languages used (Java, C#, or assembly code) provide more ways to obscure flow than in our scripting language, in particular through indirect calls, or via reflection. The static analysis would have to be extended to account for these (Enck et al. 2011; Felt et al. 2011a). (3) Indirect flow through mutable storage will require a finer grained heap model than we currently employ (one abstract location per data kind). The static analysis might need to be complemented with dynamic analysis (Enck et al. 2010; Zhu et al. 2011) to address this issue.

Limitations of static information flow analysis Due to the way our approach handles implicit flows, our approach may produce false positives as described by Kang et al.'s work (Kang et al. 2011). However, our evaluation results show that even with these potential false positives, our approach still achieves a significant reduction in access granting for users. To improve our approach when migrating to other mobile-device platforms, our approach can be combined with DTA++ techniques (Kang et al. 2011).

Another type of implicit flow, covert channels (Shieh and Gligor 1990), may cause false negatives of our approach. For example, a script can store a classified picture into the media library, and then later share it through facebook via a different application. Our flow analysis would indicate that a picture is stored into the media library (and the user has to agree with that flow), but our approach does not contemplate what could happen to the picture in the library after that. To address such issues, the operating system would have to provide dynamic taint tracking (Enck et al. 2010), since such flows involve more than one application or even OS built-in functionality.

11 Related work

In this section, we compare our work with other related approaches.

User-aware application capabilities Mobile-device platforms like Android and social-network platforms like Facebook use manifests to show application capabilities and request permissions at install time. Other mobile-device platforms like iOS and research approaches like TaintDroid (Enck et al. 2010) report application capabilities the first time applications try to access a resource. The capabilities shown in the manifests are either claimed by developers (Saltzer and Schroeder 1975) or only present part of the requested application capabilities. Felt et al. (2011a) proposes an approach that uses static analysis to map API calls used by applications to permissions, which is similar to our approach. However, they adapt automated testing methodology to test the applications and identify APIs that require permissions, while our approach annotates the APIs with permissions and uses static checking.

Wetherall et al. (2011) propose a concept called *privacy revelation*, which requires that (1) users must be aware of the spread of personal information based on user-relevant context; (2) users should be able to give feedback before information exposure; (3) users can learn from other users' experiences. Part of our approach can be considered as one instance of their concept, since our approach reveals information flows to users and requests users to grant access to private information. However, our approach adapts static information flows analysis to expose information at both the install time and the first time users run the applications, while their developing systems are all based on dynamic analysis, which cannot provide information before users even installs an application. Moreover, our anonymize/real/abort setting encourages users to try out applications with safe default settings, while their approach encourages sharing of privacy revelations.

Information flow analysis Xie and Aiken (2006) present an approach that statically computes summaries of blocks and procedures of PHP and detects security vulnerabilities at the block level, intraprocedural level, and interprocedural level. Their approach does not handle reference-type flows shown in Fig. 4, and would lose track of flows after built-in procedure calls (e.g., senses→take camera picture) that cannot be analyzed by their approach. To address these problems, our approach uses mutable locations to simplify analysis of reference-type flows and tracks untampered- and tampered-classified information for classifying safe and unsafe flows.

The closest work related to ours is PiOS (Egele et al. 2011), which studies private information leakage in actual iOS binaries. The PiOS approach statically computes data flow along control flow paths from sources to sinks to determine if there exists a user prompt along that path. PiOS emits warnings if such a flow is found without a user prompt. For the purposes of safe-guarding users of the TouchDevelop bazaar, the PiOS approach is insufficient because: (1) the PiOS analysis is not conservative, it misses flows that are too long or use indirect flow, (2) the prompts PiOS identifies may be unrelated, show nothing of the leaked information, or show tampered information. PiOS also does not use the static information to control user prompting and privacy settings as our approach does.

Language-based information flow (Sabelfeld and Myers 2002) allows developers to annotate variables with security attributes. These attributes can be used by compilers to enforce information flow controls. For example, Heintze and Riecke (1998) shows that information flow labels can be applied to a simple language with reference types and Jif (Myers and Liskov 2000; Myers 1999) extends Java language with statically-checked information flow annotation. Laminar (Roy et al. 2009) allows developers to specify security regions and provide information flow controls on both language and JVM/OS levels. Although these language-extending approaches are effectively in guaranteeing information flow controls, they impose additional burdens on developers when writing applications, which is undesirable for writing scripts on mobile devices in the context of TouchDevelop, especially for beginners.

Dynamic taint analysis (Enck et al. 2010; Zhu et al. 2011) has been applied to track information flows on both mobile platforms like Android and desktop platform like Windows. These approaches track tainted data during runtime, providing accurate runtime information about leaks. However, to reduce runtime overhead, these approaches usually ignore implicit flows raised by control structures. Moreover, dynamically executing all execution paths of these applications to detect potential information leaks is impractical. These limitations make these approaches inappropriate for computing information flows for all submitted applications.

Access granting Mobile-device platforms like Android and social-network platforms like Facebook use manifests to request permissions at install time. Once permissions are given by users, the permissions cannot be changed. iOS and Windows User Account Control (MICROSOFT 2011) prompts a dialog to request permissions from users when applications try to access a resource or make security or privacy-related system-level changes. Instead of only showing information about the access to resources, our approach presents information flows to describe what applications may do with private user information. Our access granting also provides a way for users to try out applications before using private information, and these settings can be changed at will.

Zhu et al. (2011) propose an approach that uses dynamic taint analysis to track user data as it flows through applications. Their approach allows users to choose among logging the action, blocking the system call, or randomize the tainted data. Chen et al. also propose an approach that shadows data that the user wants to keep private and blocks network transmissions that contain data the user made available to the application for on-device use only (Hornyack et al. 2011). Our anonymized/real/abort setting is inspired by their approach, but we use static information flow analysis extended with tampered information to classify flows as safe/unsafe flows and provide default access settings, rather than runtime information.

Automated security validation of mobile apps Gilbert et al. (2011) present a vision of making mobile apps more secure via automated validation. They propose using commodity cloud infrastructure to emulate smartphones and run the submitted apps to dynamically track information flows and actions. Based on the information flow and action tracking, they propose to automatically detect malicious behavior and misuse of sensitive data via further analysis of dependency graphs (Ferrante and Ottenstein 1987) or natural language processing. Such an approach is akin to automated testing and suffers from the same problems, namely coverage. It is difficult to drive applications automatically into exercising all data and control paths. Thus, in the end, such an approach only gives a partial view of the behavior and does not safe-guard users.

Anonymization techniques to protect privacy Castro et al. (2008) propose an approach that uses anonymization techniques to protect private information in bug reports delivered to vendors when programs crash on computers of customers. Clause and Orso (2011) propose an approach that sanitizes the inputs for causing failures in field and releases the sanitized inputs to help developers debug. There also exist approaches (Grechanik et al. 2010; Taneja et al. 2011; Budi et al. 2011) that study how to release private data for testing and debugging by combining the k-anonymity techniques with program-behavior preservation. Our approach allows users to use anonymized information for a source, and currently uses a static list of fixed information for such purpose. In future work, we plan to investigate how to use the anonymization techniques proposed by these approaches to anonymize the sensitive information while preserving the behaviors.

12 Conclusion

We presented a user-aware privacy control approach based on static information flow analysis extended with tamper analysis. We compute information flows from private sources to sinks and classify them as safe/unsafe flows. We conducted evaluations on 546 scripts published in TouchDevelop to study the effectiveness of our static information flow analysis. The results show that our approach computes useful information flows and can be used to automatically provide default privacy settings for each script that keeps users safe without any user intervention, thereby obviating the need for manual script validation. We also conducted a user survey on 50 TouchDevelop users. The results show that 90 % of the users consider our approach useful in protecting their privacy and 54 % of them prefer our approach over other privacy control approaches.

Our approach is the first step towards employing a better privacy control mechanism in mobile-device platforms based on automatic validation of applications in the marketplace and user-driven access control.

References

Askarov, A., Myers, A.: A semantic framework for declassification and endorsement. Programming Languages and Systems. LNCS, vol. 6012, pp. 64–84. Springer, Heidelberg (2010)

Budi, A., Lo, D., Jiang, L., Lucia: Kb-anonymity: a model for anonymized behaviour-preserving test and debugging data. In: Proceedings of PLDI, pp. 447–457 (2011)

Castro, M., Costa, M., Martin, J.-P.: Better bug reporting with better privacy. In: Proceedings of ASPLOS, pp. 319–328 (2008)

Clause, J., Orso, A.: Camouflage: automated anonymization of field data. In: Proceedings of ICSE, pp. 21–30 (2011)

Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252 (1977)

Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**, 236–243 (1976)

Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM **20**, 504–513 (1977)

Egele, M., Kruegel, C., Kirda, E., Vigna, G.: PiOS: detecting privacy leaks in iOS applications. In: Proceedings of NDSS (2011)

Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of OSDI, pp. 1–6 (2010)

Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S.: A study of android application security. In: Proceedings of USENIX Security Symposium (2011)

Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: Proceedings of CCS, pp. 235–245 (2009)

Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of CCS (2011)

Felt, A. P., Finifter, M., Chin, E., Hanna, S., and Wagner, D.: A survey of mobile malware in the wild. In: Proceedings of SPSM, pp. 3–14 (2011)

Felt, A.P., Greenwood, K., Wagner, D.: The effectiveness of application permissions. In: Proceedings of WebApps (2011)

Ferrante, J., Ottenstein, K.J.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. **9**, 319–349 (1987)

Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J.: Vision: automated security validation of mobile apps at app markets. In: Proceedings of MCS, pp. 21–26 (2011)

Grechanik, M., Csallner, C., Fu, C., Xie, Q.: Is data privacy always good for software testing? In: Proceedings of ISSRE, pp. 368–377 (2010)

Heintze, N., Riecke, J.G.: The SLam calculus: Programming with secrecy and integrity. In: Proceedings of POPL, pp. 365–377 (1998)

Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: Proceedings of CCS, pp. 639–652 (2011)

Howard, F.: Malware with your mocha: obfuscation and anti-emulation tricks inmalicious javascript. http://www.sophos.com/security/technical-papers/malware_with_your_mocha.pdf. Accessed Sept 2011

Kang, M.G., McCamant, S., Poosankam, P., Song, D.: DTA++: Dynamic taint analysis with targeted control-flow propagation. In: Proceedings of NDSS, San Diego, CA, February (2011)

Li, S., Xie, T., Tillmann, N.: A comprehensive field study of end-user programming on mobile devices. In: Proceedings of VL/HCC (2013)

MICROSOFT: What is user account control? http://windows.microsoft.com/en-US/windows-vista/What-is-User-Account-Control (2011)

Myers, A.C.: JFlow: practical mostly-static information flow control. In: Proceedings of POPL, pp. 228–241 (1999)

Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Trans. Softw. Eng. Methodol. **9**(4), 410–442 (2000)

Roesner, F.: User-driven access control: a new model for granting permissions in modern operating systems. Qualifying Examination Project, University of Washington, June (2011)

Roy, I., Porter, D.E., Bond, M.D., Mckinley, K.S., Witchel, E.: Laminar: practical fine-grained decentralized information flow control. In: Proceedings of PLDI, pp. 63–74 (2009)

Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Select. Areas Commun. **21**, 5–19 (2002)

Saltzer, J. H., Schroeder, M. D.: The protection of information in computer systems. In: Proceedings of the IEEE, pp. 1278–1308 (1975)

Shieh, S.-P., Gligor, V. D.: Auditing the use of covert storage channels in secure systems. In: Proceedings of Oakland, pp. 285–295 (1990)

Taneja, K., Grechanik, M., Ghani, R., Xie, T.: Testing software in age of data privacy: a balancing act. In: Proceedings of ESEC/FSE, pp. 201–211 (2011)

Tillmann, N., Moskal, M., de Halleux, J.: Touchdevelop - programming cloud-connected mobile devices via touchscreen. Microsoft Technical Report MSR-TR-2011-49 (2011)

TouchDevelop. http://research.microsoft.com/en-us/projects/touchdevelop/ (2011). Accessed 21 Aug 2014

Vidas, T., Christin, N., Cranor, L.: Curbing Android permission creep. In: Proceedings of W2SP, Oakland, CA, May (2011)

Wetherall, D., Choffnes, D., Greenstein, B., Han, S., Hornyack, P., Jung, J., Schechter, S., Wang, X.: Privacy revelations for web and mobile apps. In: Proceedings of HotOS, pp. 21–21, Berkeley, CA, USA (2011). USENIX Association.

Xiao, X., Tillmann, N., Fähndrich, M., de Halleux, J., Moskal, M.: User-aware privacy control via extended static-information-flow analysis. In: Proceedings of ASE, pp. 80–89 (2012)

Xie, Y., Aiken, A.: Static detection of security vulnerabilities in scripting languages. In: Proceedings of USENIX Security (2006)

Zhu, D.Y., Jung, J., Song, D., Kohno, T., Wetherall, D.: TaintEraser: Protecting sensitive data leaks using application-level taint tracking, pp. 142–154. SIGOPS Operating Systems Review (2011)