# Context-centric Security

Mohit Tiwari Prashanth Mohan Andrew Osheroff Hilfi Alkaff Elaine Shi[1] Eric Love Dawn Song Krste Asanović
*University of California, Berkeley* *University of Maryland, College Park*[1]

**Abstract.** Users today are unable to use the rich collection of third-party untrusted applications without risking significant privacy leaks. In this paper, we argue that current and proposed *applications* and *data*-centric security policies do not map well to users' expectations of privacy. In the eyes of a user, applications and peripheral devices exist merely to provide functionality and should have no place in controlling privacy. Moreover, most users cannot handle intricate security policies dealing with system concepts such as labeling of data, application permissions and virtual machines. Not only are current policies impenetrable to most users, they also lead to security problems such as privilege-escalation attacks and implicit information leaks.

Our key insight is that users naturally associate data with real-world events, and want to control access at the level of human contacts. We introduce Bubbles, a *context*-centric security system that explicitly captures user's privacy desires by allowing human contact lists to control access to data clustered by real-world events. Bubbles infers information-flow rules from these simple *context*-centric access-control rules to enable secure use of untrusted applications on users' data.

We also introduce a new programming model for untrusted applications that allows them to be functional while still upholding the users' privacy policies. We evaluate the model's usability by porting an existing medical application and writing a calendar app from scratch. Finally, we show the design of our system prototype running on Android that uses bubbles to automatically infer all dangerous permissions without any user intervention. Bubbles prevents Android-style permission escalation attacks without requiring users to specify complex information flow rules.

## 1 Introduction

Mobile application distribution mediums, such as Apple's "App Store" and Google "Play", have been instrumental in the adoption of mobile computing devices, allowing otherwise unknown publishers to sell apps to millions of smartphone and tablet users around the world. Today, these extremely personal devices are used to perform telephony, messaging, financial transactions, gaming, and many other functions. While the popularity of the app store model attests to the benefit and utility of allowing third-party apps, these applications introduce a host of privacy and security risks [6].

A number of different security mechanisms have been used to provide data privacy. For instance, Android's permissions model [1] is an example of *application-centric* security. Android has a static capability-based system where users must decide at installation time whether to grant permissions (including network and device access) or not (and forgo use of the application).

As an alternative, information-flow tracking systems [4, 19] are more expressive than static capabilities and provide *data-centric* security, but require considerable sophistication on the user's part to translate security policies into a lattice of labels. Moreover, they often require applications to be modified with security label assignments so that they do not crash with security exceptions. TaintDroid [5] is an example of an information-flow tracking system for Android, which modifies the Dalvik virtual machine to propagate taint through program variables, files, and IPC messages. While it can track some unmodified Android apps, it cannot track applications which use their own native libraries. Further, information leakage through implicit flows [12] cannot be restricted.

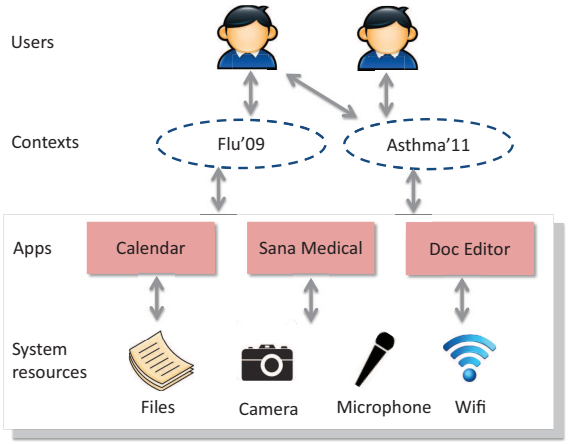


*Figure 1: Traditionally, security policies are expressed in terms of permissions on applications or security labels on system-level features. This makes it hard to capture users' intentions that stem from high-level, real-world contexts, and lead to either static, inflexible permissions as in Android or sophisticated policies and implicit information leaks as with TaintDroid.*

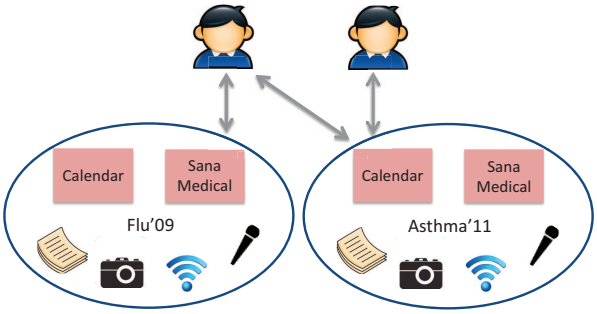


*Figure 2: Bubbles represent real-world contexts that are potentially shared among multiple users, and around which users' data automatically clusters. Users' privacy policies can then be directly represented as an access-control list on bubbles. Applications and low-level peripherals then exist solely to provide functionality and cannot affect who sees the contents of a bubble. Bubbles are thus implemented as tightly constrained execution environments (like a virtual machine, but much lighter weight), and require applications to be partitioned to provide the functionality associated with legacy applications.*

We believe security systems can only be effective if they present a security model that matches the way users' reason about privacy in their real-world interactions. In this paper, we propose a new system, Bubbles, which allows a user to define a digital boundary (called a *bubble*) around the data

associated with a real-world event. For example, the event might be a meeting for a work project or a birthday party. Goffman [9] proposed that people present different *faces* of themselves depending upon the social context. Bubbles implements this sociological aspect of privacy in a digital setting. Similar to attaching access control (ACLs) permissions to files in a file system, the user specifies her privacy requirements for the bubble as a list of people, taken from her contacts list, who have access to the bubble. Different sets of people might have access to different work projects, or to different birthday parties. Bubbles preserves this privacy model across all the user's multiple devices and the cloud. Bubbles thus provides *context-centric* security which isolates all data between bubbles. Bubbles translates these simple user-centric access control rules into the more complex information-flow rules in the underlying system. We have also developed a simple application design model to enable app developers to provide extensive functionality for the user without requiring either the user or the app developer to worry about privacy enforcement. Bubbles effectively factors out privacy features from applications, and puts privacy control into the hands of the user in a natural way.

## 2 Motivation for Bubbles

This paper focuses on the threat to users' privacy posed by untrusted applications and assumes that applications can execute arbitrary code to leak sensitive data to malicious recipients. Bubbles relies on operating system and network level security primitives of isolation, encryption, and anonymized network traffic being implemented correctly.

### 2.1 Android Permissions Considered Harmful

Android is an operating system that includes a modified Linux kernel together with standard user space sub-systems, and is targeted toward mobile devices such as smartphones and tablets. Android provides more than 100 permissions that an application can request at installation time, which a user must explicitly approve or deny. Although, Android warns users by marking some permissions as dangerous, "93% of free and 82% of paid applications request at least one dangerous permission" [7]. The sheer number of permissions being requested causes users to be indifferent about security. Moreover, we find that Android exposes permissions that are foreign to even sophisticated computer users, such as `MOUNT_FORMAT_FILESYSTEMS` (which allows applications to format file systems on removable storage). By asking users questions at the wrong level of abstraction, Android leaves users' privacy in the hands of untrusted applications. Finally, since permissions in Android are statically enforced during installation, applications have these permissions forever. Static permissions allow any application with microphone access, for example, to record one's voice even when he/she is on a phone call.

### 2.2 Many Applications fit Bubbles

We analyzed 750 of the top free and the top paid (375 of each) Android applications from the Google Play store to determine how their functionality relates to users' privacy. On one end, from a privacy point of view, are applications that provide functionality that is not tied to a user's real-world identity. Examples of such applications include flashlights, games, wallpapers, dictionaries, news sites, and browsing for reviews and recipes among others. Such applications can run inside an "anonymous" bubble where the users are expected to not enter sensitive information, and can move data to and from arbitrary locations into the anonymous bubble. We find that 45.6% of the free and 45.3% of the paid applications fit this model.

The second category of applications are where users actively create data (we assume this data is sensitive) and then *explicitly* share this data with other users. Applications that allow storing, editing, and sharing of documents (in formats that range from simple text and images to even audio and video), and real-time communication applications that use SMS, MMS, voice, or video fit this category and account for 47.4% of the free and 52.3% of the paid applications. The common feature of these applications is that users can specify for a given blob of data who they want to share it with – in essence an Access Control List (ACL) for each data blob – and the key insight behind Bubbles is to tie these arbitrary blobs of data to a real-world context.

The remaining 7% free and 2.4% paid applications perform what we term *application-initiated sharing*, where functionality such as a recommendation service requires that users give up their personal data to the application and the application mixes information from multiple users to generate new suggestions or insights. ACLs do not capture the privacy requirement here, because a user has to give up her data to the application, and alternate definitions of anonymity such as differential privacy are required to guarantee that a user cannot be singled out from a dataset by an untrusted application. Most social networking applications include features that implement explicit communication of data, which can be integrated into Bubbles, while features that initiate sharing through aggregate analytics require an anonymizing proxy to enforce privacy through differential privacy. Integrating such a proxy is out of scope of this paper. Note that we only discuss a client-side implementation of the applications; the server side of Bubbles can be assumed to extend the abstraction of a bubble across the network. We defer the discussion of server-side optimizations that access data from multiple users (like deduplication) to a future paper.

## 3 User Abstraction

**Bubbles.** Our core hypothesis is that users want to work in *contexts*, where a context encapsulates information of arbitary types— be it audio, video, text, or application-specific data— and is often tied to a real-world event involving other people. We call such light-weight contexts that have data and people associated with them Bubbles. Applications just exist in each bubble to provide functionality, and any data that a user accesses will trigger its corresponding application.

We also propose that users' privacy policies are inherently tied to such contexts and thus are best stated in terms of Access Control Lists (ACLs) of contacts for each bubble, In one extreme case, if the users are effectively broadcasting information (as when they browse websites or public forum) they want to be aware of this and act accordingly. In the other

extreme case, and by default, users require all information related to a certain context to be private.

**On creating new bubbles.** A bubble is effectively the minimum unit of sharing, because when all apps that act on data inside a bubble are untrusted, they can mix data arbitrarily among files that exist within a bubble. The implication of this is that sharing even a part of the data in a bubble is equivalent to sharing any data from the bubble.

As a result, we recommend that bubbles be tied to very light-weight contexts in order to facilitate flexibility in future re-sharing decisions. A coarse classification of all personal data into, say, a "Home " bubble and a "Work" bubble will lead to violation of privacy guarantees when the user moves even a single file across this Home-Work boundary. On the other hand, a light-weight event could be simply a single meeting, or even only a part of a meeting (e.g. the technical discussion as opposed to financial discussions), and putting these light-weight contexts into separate bubbles allows a user to share these smaller units of data independently. In the example above, all developers may be included in the technical discussion, but only the program managers may have access to the financial details of the project.

**Navigating the foam.** We call the collection of bubbles visible to a user, their *foam*, which replaces the conventional user-visible file system. The Bubbles system only supports a flat *foam* of bubbles without hierarchical order. The system executes bubbles inside independent, mutually isolated containers, and does not support nested bubbles, in order to prevent complications that arise in constraining untrusted applications. Since untrusted applications can operate on the data in an arbitrary manner inside a bubble, they can mix information among all data items in sub-bubbles. Sharing any data item to a new person will thus leak information from potentially all sub-bubbles to the person. Traditional systems propose fine-grained information flow analysis to control how applications mix information, but tracking implicit flows at run-time leads to considerable performance penalties. As a result, we eschew fine-grained information flow tracking in Bubbles, and build Bubbles around the basic primitive of an isolated container.

While bubbles cannot be nested, users can assign an arbitrary *tag* to a collection of bubbles, e.g., to group bubbles as belonging to some longer-term project, and can even overlay a hierarchy on the underlying foam of bubbles. Further, the system tags bubbles with time, location, nearby contacts, and other contextual information that may help the user identify or index the bubble for future reference. For instance, a user can view her bubbles as a time-line to give a calendar view, or by geographic coordinates overlaid on a map view.

**Staging Area.** Users often create data that is not immediately associated with any existing bubble or tag, for example, a phone-camera photo or a web-page downloaded for future reading. In such cases, instead of forcing the user to assign this data to a bubble, the system automatically assigns the data item to a new bubble and assign location and time-based tags to the bubble. This ensures that the user has flexibility of copying such data items into any (even multiple) bubbles later on. This can be used to implement a Photo Gallery application for example, allowing browsing of images without mixing information from one image to another. One fallout of the staging area is that the system will have a *lot* of bubbles, motivating the need for a very light-weight implementation of containers in Bubbles.

**Usage Flow**. We now use the example shown in Figure 3 to illustrate how a user works in a context-driven rather than application-driven manner. We argue this adds little cognitive overhead to regular operation and that it is worth the price of privacy.

One usage flow could be to start with the trusted *Viewer* application that allows a user to browse data from her entire foam within a single bubble, classified either by application, such as Sana or Calendar in Figure 3, or by data type, such as images. Clicking on Sana takes the user to a listing of all medical records classified by bubble name (each patient is stored in a separate bubble).

The Viewer app is discussed in more detail in Section 5. Clicking on the New Procedure then takes the user to the staging area, where the user can enter data related to a patient's visit. At the end of the procedure, and before the user moves back to the viewer mode, Bubbles prompts the user to enter a new name and tag for the bubble and other contacts who this bubble is shared with (e.g. a remote doctor). In case of an existing patient, the record can be assigned to an existing bubble. The last image on the right shows that a user, while she is in a bubble, can switch among applications (from Sana to the Calendar here) while staying within the same bubble.

Apart from the Viewer, the two trusted applications that a user interacts with in Bubbles are the Bubbles and Contacts managers. Both the Bubbles and Contacts applications are available on the Home screen (which can also include some recent and favorite bubbles for easy navigation) to navigate to an existing bubble or to create a new bubble and invite contacts into it.

## 4 System Design

In this section, we describe the design of our Bubbles prototype implemented on top of Android. All applications are available to each bubble, with all but a few permissions (explained in Section 4.3). Applications effectively execute as though each bubble was an entire OS installation.

### 4.1 Isolation between Bubbles

In order to provide file-system state isolation between each bubble, we choose an approach similar to Cells [2]. We mount a unioning file system that uses the base operating system files as a read-only copy, and a read-write scratch directory to hold any file system modification created by applications in a bubble. Note that OS files cannot be modified by any application running inside a bubble, and that scratch directories are maintained on a per-bubble basis to prevent sharing of file-system state between bubbles.

Control groups [14], supported on both Linux and its smartphone fork Android, are used to provide UTS, PID, IPC and mount namespace isolation, to further ensure that applications cannot communicate with each other apart from via the Binder IPC mechanism. We also make use of the same techniques used in the Cells system [2] to provide device iso-

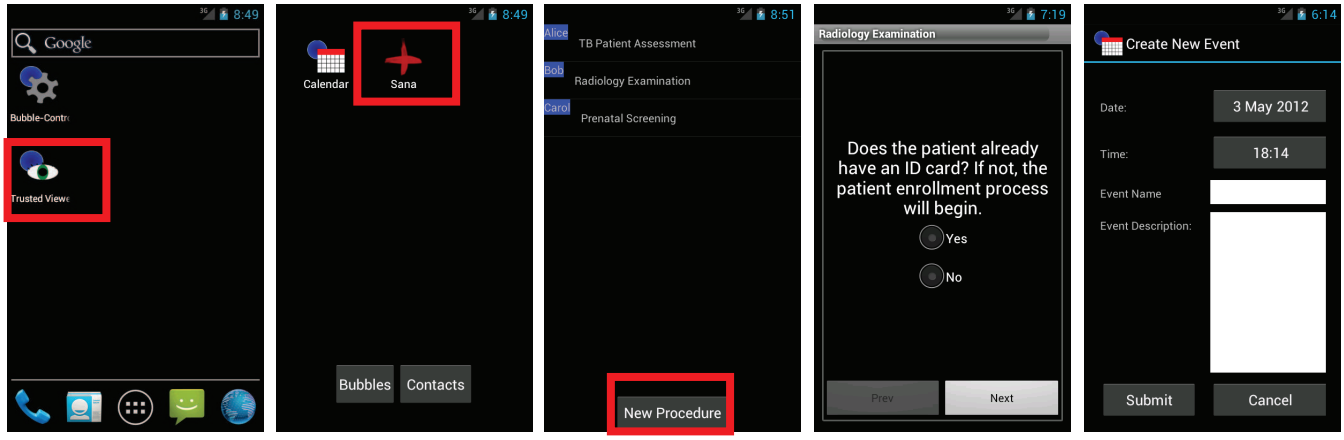


*Figure 3: Usage Flow in a Bubbles System: User's Home screen shows trusted system applications to manage Bubbles and to launch the Viewer. The Viewer allows a user to see all installed applications, such as a Calendar or the Sana medical application. Clicking an application in this mode takes the user to browse cross-bubble data, i.e. all data attached to Sana or the Calendar. Within the View mode of an application, the user can initiate new data creation; either in a Staging area (i.e. for which the system assigns a unique bubble), or by first using the Bubble service to transition into a bubble and then going to the edit screens for Sana or the Calendar (the last two screens on the right).*

lation, i.e., because we could potentially have multiple applications accessing the same device resources, it is important that devices understand the namespace abstractions.

We also enable all Android middleware services that allow persistent state to separate data between different bubbles. For instance, `SQLiteOpenHelpers` respond to `getReadableDatabase()` or `getWritableDataBase()` calls from applications with a bubble-specific instance of a database. The SD card and preferences are also virtualized in a similar manner.

### 4.2 Copying Data between Bubbles

A user may copy data from one bubble to another, e.g., when an image from the staging area is copied into an existing bubble. Such copying requires applications in the receiving bubble to be able to assimilate the incoming data structure instances with existing ones. Because the Bubbles system is agnostic of application data structures, the applications register call-back functions for handling inter-bubble data transfer.

The Android IPC mechanism is modeled upon the OpenBinder functionality in BeOS [3] wherein a `Binder` device mediates communication between different processes. In Android, this communication can be effected by using the `Context.registerReceiver()` routine to register for incoming messages (called `Intents` in Android). Bubbles's modified `Binder` driver implementation automatically modifies the `IntentFilter` to restrict messages sent to the application.

The source bubble initiates data transfer by sending a message to the `Sharing Service` implemented as part of Bubbles. In our prototype, this generates a trusted prompt asking for user confirmation about the application-initiated data share. (This prompt can potentially be removed by replacing the `Sharing Service` with trusted and isolated widgets similar to the access control gadgets [16] architecture). Finally, the applications can transfer data in any serialized format it prefers such as Google Protocol Buffers.

### 4.3 Intuitive Permissions Model

In Bubbles, instead of the applications statically requesting resource permissions at install time, permissions are automatically inferred from the access control rules placed on the bubble within which the application is executing. To achieve this, Bubbles relies on explicit user input and on virtualizing the Android resources among different bubbles. Most permissions fall into *three* broad categories and of the remaining 5 dangerous permissions, 3 have been deprecated already and 2 (writing to contacts and to APN settings) are not allowed for any untrusted application.

**Explicit User Decision.** Users are provided with a trusted UI to explicitly enable 7 permissions, in order to input audio, video, location, and contacts into a bubble. These resources have the common feature that users are familiar with these concepts outside of a computing device context and can thus make intelligent decisions about when to share them within a bubble. Writing to the contact list is however limited to the trusted address book application.

**Per-Bubble Resources.** Internal and external storage, logs, calendar, application caches, history, various settings like animation scale and process limit are all low-level resources that need not be exposed to users. Bubbles allows all applications to have access to the 27 permissions that control these resources, while maintaining a unique, per-bubble copy of each resource (e.g. isolated folders in storage).

**Concurrently Shared Resources.** Communication resources like telephony, wifi, and internet access are all shared among various bubbles. Hence Bubbles enables all 17 communication-related permissions to all applications in a bubble, but with firewall rules to ensure that all communication follows ACL rules specified by a user. Hence applications in a private bubble can only communicate with Bubbles servers (or only send SMS messages to a contact who can access the bubble), while an application in an anonymous bubble can talk to arbitrary servers. While not relevant for

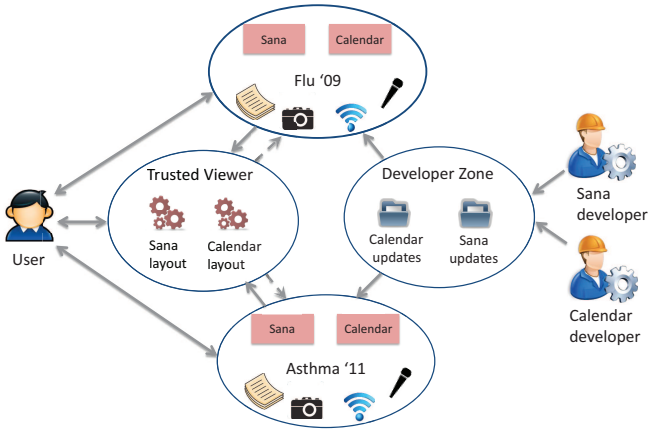


*Figure 4: Applications in Bubbles: Most application functionality is included in its* editor *component with one editor instance inside each bubble (e.g., inside Flu'09 and Asthma'11 bubbles). A* Viewer *bubble provides cross-bubble functionality by combining trusted code that receives and processes data from multiple bubbles and a layout file specified by the application statically that determines how such data is laid out. Finally,* Bubbles *provides developers with their own bubble to send in application updates that a user's personal bubble can only read from and never write to.*

privacy, additional rules can be imposed to prohibit communications that cost money.

## 5 Applications

**Simple Programming Model for Developers**. As shown in Figure 4, legacy application functionality can be partitioned into two components: *Editor* and *Viewer* modes. The editor mode includes most user-facing functionality that one typically associates with a legacy application, e.g., adding and editing patient records in Sana or new events in the Calendar. Bubbles then ensures that there is a unique editor instance for each bubble so that untrusted editor code will only ever see data from one bubble.

The viewer mode of an application is required for the user to be able to browse data from multiple bubbles, e.g. medical records for all patients in Sana. There is thus only one viewer instance on a client device (or one per user). The viewer is responsible for capturing the user's intentions when she uses the application's interface (e.g. clicking a patient record or searching for a particular illness), and forwarding the resulting query to individual bubbles. This requires a message to be passed from the viewer to one or more editors to convey, for example, a specific data item to be opened or a specific term to be searched for. Since the viewer accesses cross-bubble data and then generates this message, an untrusted viewer can encode information in the message. Thus, we only allow fixed builtin *trusted viewers* to prevent information leaks among different bubbles.

**Cross-bubble Functionality**. To enable application-specific functionality in the viewer mode, an application specifies a layout file and a call-back function at install time. Applications register themselves with the trusted Viewer Service by sending an intent containing 2 XML files: `ui.xml`, which informs the Viewer Service how to display the layout of the application's viewer mode (our current prototype supports the LinearLayout ViewGroup and so far the only supported Views are `ImageView`, `TextView`, `GridView`, `ListView` and `EditText`.); and `app_data.xml`, which contains basic application information (name, icon, etc) together with a callback to be called whenever any entry in the viewer mode is acted on by the user. Sana enables a user search (for any text string) by registering a "search" button as an action item in `app_data.xml` and using the call-back function to receive the search string as a message. The user can control which bubbles the message is sent to.

We have so far ported two applications— Calendar and Sana —without requiring substantial code changes. The editor mode had to be changed to call Bubbles wrappers around Android services (e.g. SQLiteOpenHelper replaced by DBSwapper), a call-back function implemented to parse messages from the viewer component, and XML files that indicates how the viewer should display calendar or medical records from multiple bubbles. We needed 500 lines of code to port Sana, including all wrapper classes, and implemented the Calendar app in about 800 lines of code.

**Developer Zone for Ads and Updates**. We introduce Developer Zone as a storage area to support advertisements and software updates (Figure 4). Its key feature is that it can be written to by developers, but only read by application editor instances in each bubble.

Bubbles is complementary to several prior works on privacy-preserving advertising [18, 10, 8] – basically, Bubbles can be readily used to provide a secure client-side implementation for these privacy-preserving advertising systems. A dedicated ad retrieval bubble retrieves a set of ads from the ad network, based on information (e.g., a broad interest category) a user explicitly shares with the ad retrieval bubble. An ad selection bubble (i.e., a viewer service) can potentially perform user profiling on sensitive, fine-grained personal information or behavioral traces of the user; it reads the ads retrieved by the ad retrieval bubble, and selects and displays the most relevant ad.

Software updates, as shown in Figure 4, requires new information from the developer to be made available to the application. The server conducting the updates (e.g., Google Play) will be able to write and read from the Developer Zone, but for the applications' editor instances, this will be a read-only area. An application can provide a button for the user to indicate when she wants to check for updates and then, the application just uses the Bubbles API to check, read and apply the updates from the DevZone.

## 6 Related Work

The idea of isolating applications and data on a mobile based on the user context has been widely explored in literature [17, 11]. Bubbles extends existing literature by providing developers with an easy to use design pattern for developing fully functional applications that abide by the users' privacy policies at the same time.

Helen Nissenbaum argues that privacy is closely linked to prevailing social, economic, and judicial norms, i.e. the prevailing social context [15]. We agree with this premise; in fact, Bubbles system allows a user to be clear about what she has shared with whom, and thus will form a basic primitive for the system Nissenbaum envisions. We use the term con-

text to mean any small event (a meeting, a browsing session) that generates some arbitrary data.

MobiCon is a system that provides context detection services to context-aware applications and implements the detection algorithms in energy-efficient ways [13]. However, Bubbles uses the term contexts to indicate a real-world event that creates data that has some access control attached to it. Bubbles could, in addition to user input, use MobiCon as an underlying service to better infer automatic tags, or even the start/end of a bubble's lifetime.

The Cells [2] system maintains parallel Android execution environments running unmodified Android applications, where it creates "virtual phones" that are completely isolated from each other. It makes devices namespace-aware by introducing wrappers around drivers, as well as modifying the device subsystem and the device driver itself. Bubbles uses the concept of 'virtual phones' proposed in the paper to generate bubbles since they serve the same purpose. In addition to the Cells infrastructure, Bubbles also performs permission inference using the peripheral device virtualization and provides an API for untrusted applications. In contrast to Cells, where a few virtual phones are envisioned to execute all the time, Bubbles will create a large number of bubbles that need to be managed using trusted indexing services.

There has also been work to reduce access permission prompts by providing trusted widgets called 'Access Control Gadgets' that an untrusted application can insert to get access to privileged resources [16]. These gadgets are isolated from the application and their integrity ensured. ACGs can thus be used in Bubbles for the user to convey a carefully chosen set of permissions. Unlike Bubbles, once an application has access to the data, ACGs do not allow a user to enforce context specific privacy policies on the data.

TaintDroid [5] is closest in spirit to Bubbles. As explained earlier, TaintDroid ensures that private sensitive data sources are not exposed by propagating taint through files, variables and IPC messages. However, information leakage is possible in TaintDroid through implicit flows. Further, because information flow control requires source code to be annotated with security labels, TaintDroid disallows application-specific native libraries. Bubbles guards against both of this by isolating all state between bubbles.

## 7 Conclusions and Challenges

We propose Bubbles, a new *context*-centric security paradigm that empowers users to retain control of their data while still benefitting from the broad spectrum of *untrusted* third-party applications on mobile devices. Bubbles provides end users with an intuitive interface through which users can configure their privacy policies with respect to *contexts*. Bubbles transparently translates these simple access control rules into information flow policies. If the application developers inadvertently creates a privacy breach, the application functionality will be curtailed, but the privacy policy will still be enforced. There is thus a strong incentive for the developer to follow the suggested design pattern to ensure the full functionality of their applications running on Bubbles.

Bubbles poses several important technical challenges and opens up several directions for future research. One important challenge is how to realize a framework for light-weight containers that incurs low run-time overhead, and how to spawn and destroy containers extremely rapidly. Another important task for future work is to conduct user studies to evaluate the user-friendliness of Bubbles.

## Acknowledgments

The authors would like to thank the anonymous reviewers for providing useful comments on this paper. This material is based upon work supported by the AFOSR under MURI award FA9550-09-1-0539, by the National Science Foundation under Grant #1136996 to the Computing Research Association for the CIFellows Project, by NSF awards CPS-0932209 and CPS-0931843, and by Intel through ISTC for Secure Computing.

## References

[1] Android security overview. `http://source.android.com/tech/security/`.

[2] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: a virtual mobile smartphone architecture. In *SOSP*, 2011.

[3] M. Brown. *BeOS: porting UNIX applications*. Morgan Kaufmann, 1998.

[4] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[5] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[6] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *SPSM*, 2011.

[7] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *WebApps*, 2011.

[8] M. Fredrikson and B. Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *S & P*, 2011.

[9] E. Goffman. *The presentation of self in everyday life*. Garden City, NY: Doubleday Anchor Books, 1959.

[10] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *NSDI*, 2011.

[11] M. Johnson and F. Stajano. Implementing a multi-hat pda. In *Security Protocols*, pages 295–307. Springer, 2007.

[12] D. King, B. Hicks, M. Hicks, and T. Jaeger. Implicit flows: Can't live with 'em, can't live without 'em. volume 5352 of *Lecture Notes in Computer Science*. Springer, 2008.

[13] Y. Lee, S. S. Iyengar, C. Min, Y. Ju, S. Kang, T. Park, J. Lee, Y. Rhee, and J. Song. Mobicon: a mobile context-monitoring platform. *Commun. ACM*, 55(3):54–65, Mar. 2012.

[14] P. B. Menage. Adding Generic Process Containers to the Linux Kernel. In *Linux Symposium*, June 2007.

[15] H. Nissenbaum. Privacy in context: Technology, policy, and the integrity of social life. 2009.

[16] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *IEEE S&P*, 2012.

[17] J. Seifert, A. De Luca, B. Conradi, and H. Hussmann. Treasurephone: Context-sensitive user data protection on mobile phones. *Pervasive Computing*, pages 130–137, 2010.

[18] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.

[19] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in histar. In *OSDI*, 2006.