Duvel: Enabling Context-driven, Multi-profile Apps on Android through Storage Sandboxing

Sharath Chandrashekhara, Taeyeon Ki, Karthik Dantu, and Steven Y. Ko
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
{sc296,tki,kdantu,stevko}@buffalo.edu

ABSTRACT

We present a novel technique to achieve a dynamic, context-driven, multiple-profile manager for individual apps on stock Android. Our system allows users to use a single app with any number of accounts, allows incognito modes for every app, and allows a context-driven dynamic switching between the profiles (e.g., based on geolocation). Our technique achieves this by creating a sandboxed storage environment within each app through byte-code instrumentation. This allows for a clean separation of profile specific data and allows users to run personal and business accounts on the same phone, or sandbox an app in incognito mode without sharing any data between them. We present many more use cases where our solution can be used to improve user experience on mobile systems.
In contrast to many of the existing solutions, our solution eliminates any modifications to the platform, does not require any special SDK to develop apps, and can use a context-driven policy to dynamically switch between profiles. We realize a storage sandbox environment called *Duvel* on Android, based on our previous work BlueMountain, and show how *Duvel* can enable using multiple accounts and incognito mode in popular apps.

KEYWORDS

Mobile systems; Data management; Bytecode instrumentation; BYOD; Enterprise Mobility Management

1 INTRODUCTION

Mobile devices have become an essential part of people's lives and are being used in increasingly complex scenarios. Part of the complexity is driven by the various virtual roles we take using online accounts on our mobile devices. Applications of such complexity include: (1) Using an office suite for creating presentations, (2) Sharing a mobile phone with family members, especially children, (3) Publishing photos online on platforms such as Instagram or Google Photos, (4) Social media accounts such as Twitter and Facebook. These situations require systems which can dynamically adapt to a given context.

Each of the above scenarios can be used by the multiple virtual roles assumed by a single person. For example, one person can use an office suite to make presentations for her job (role = professional) or for the local residential community meeting (role = personal). Similarly, a person can maintain a social media account for herself and multiple others for business/personal interests. Until recently, it was common practice for business people to keep two separate devices to provide a clear separation between personal and business activity.

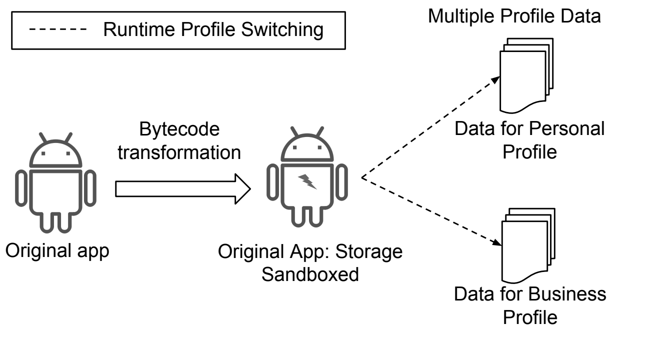

Runtime Profile Switching
Multiple Profile Data
Bytecode transformation
Data for Personal Profile
Original app
Original App: Storage Sandboxed
Data for Business Profile

Figure 1: Overview

In addition, mobile devices have become very personalized and users have developed strong preferences for them. This demand has led to companies allowing users to use a device of their choice even to access business apps and documents. This program is popularly called 'Bring Your Own Device' and allows users to use the same device they use their personal apps on for business apps as well. In order to keep the business data secure, the business apps are sandboxed within the personal phone so the business data is not accessible to personal phones. This is done through specialized software called 'Enterprise Mobile Management' (EMM) (e.g., VMware AirWatch [18] and Android Enterprise [10]) which sets up a separate configuration for individual apps for business as well as personal use. Using some mechanism (automated or manual), the EMM software then enables the correct configuration during use time. However, in order to use apps inside such sandboxed environments, apps need to be customized. Therefore, if a user wants to use her favorite note taking app to record business meetings notes and personal notes, it cannot be supported using EMM software. Rather, they have to manually switch between the two accounts depending on their need.

It is therefore desirable to design a general system that allows regular apps to access multiple profiles with separate storage and allow for easy transition between these profiles. Further, we envision a system wherein apps will be able to use the rich context information such as geolocation, network connectivity, user identity

EdgeSys'18, June 10–15, 2018, Munich, Germany
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5837-8/18/06…$15.00
https://doi.org/10.1145/3213344.3213350

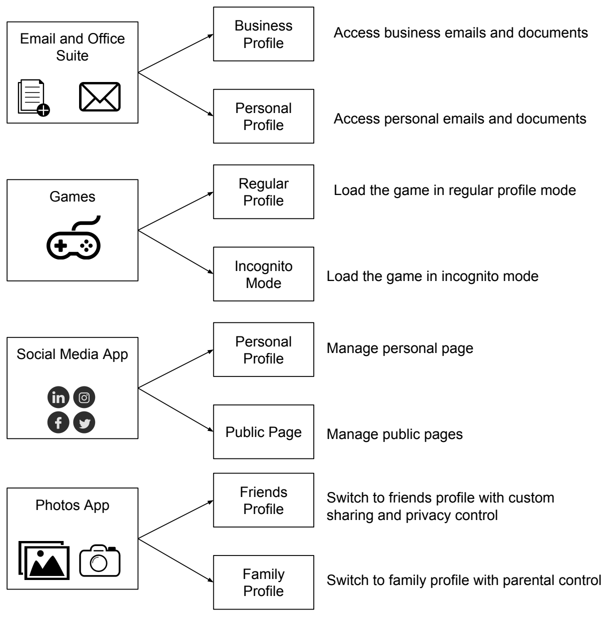

Email and Office Suite
Business Profile
Access business emails and documents
Personal Profile
Access personal emails and documents
Games
Regular Profile
Load the game in regular profile mode
Incognito Mode
Load the game in incognito mode
Social Media App
Personal Profile
Manage personal page
Public Page
Manage public pages
Photos App
Friends Profile
Switch to friends profile with custom sharing and privacy control
Family Profile
Switch to family profile with parental control

Figure 2: Use Cases of Multiple-Profiles

etc., and use this context to automatically switch to the appropriate sandboxed storage environment. For instance, when a user is connected to her office wireless network, the email client should operate with the user's work account. It is also desirable to contain such functionality completely in the app so platform modifications are not required for deployment. Finally, to support legacy apps, it would help to provide an automatic way of instrumenting previously developed apps while also providing a library to apps being developed currently to allow for easy use of such functionality in legacy as well as new apps.

In this paper, we present our prototype system *Duvel*, which achieves the aforementioned goals by creating a general, context-based profile management framework. It is built on top of the *BlueMountain* [6], a framework for flexible data management in mobile devices.

The remainder of this paper is organized as follows. In Section 2 we discuss more use cases and scenarios where using apps with multiple profiles will be beneficial, in Section 3 we will discuss the general design space for creating such a system, in Section 4 we provide the implementation details of our prototype, in Section 5 we discuss the related work, and end with discussing the future work and conclusion in Section 6.

2 MOTIVATION

As mentioned in section 1, our key insight is to enable multi-profiles on mobile apps through storage sandboxing. In this section, we will motivate its benefits through four specific use cases where storage sandboxing would greatly benefit the users. The summary of these use cases is shown in Figure 2.

Scenario 1, Personal/Business Apps: Alice is a business professional who uses her phone for both business and personal work. To keep her personal data separate from her business data, Alice uses two different accounts with many apps, such as email client and office suite. When Alice has signed-in with her business account, but wants to send a personal email, or access a document from her personal office suite, she has to sign-out of the app, and sign-in with her personal account. If Alice has to switch back-and-forth between the accounts often, it becomes an annoying process. This would perhaps not be a problem if the app she uses allows multiple accounts (e.g., GMail app), but most apps do not have such a feature. Alice can greatly benefit from a system like *Duvel*, where she can switch between profiles easily and instantly. To make the process even easier, *Duvel* can automatically switch between Alice's accounts using a certain policy. For instance, the system can switch to Alice's business account whenever she is connected to her office Wi-Fi, or switch to her personal account when she is connected to her home Wi-Fi.

Scenario 2, Incognito/Guest Mode: Bob is a mobile user who is very conscious of his privacy and tries to minimize his online footprint. On his desktop, Bob typically browses in incognito mode. But on his mobile device, Bob has to sign out of every app after he uses it and sign in again the next time he wants to use it. Given the number of apps on his mobile device, he sometimes forgets to sign out of apps and many apps continue to track him, which makes Bob unhappy. Bob can take advantage of a system like *Duvel*, where Bob can set a policy to always switch to guest-mode or incognito mode at a particular time of the day, or when on international travel. In incognito mode, the app uses a temporary sandboxed storage environment which will not be persisted. It can also be used for guest mode. E.g., when Bob's friend wants to borrow Bob's device to quickly check their social media account. This features works like Android's guest mode, but for individual apps rather the whole system.

Scenario 3, Social Media: Carol is a food blogger who extensively uses Instagram to publicly post pictures about her culinary creations. She also uses Instagram to share her personal photos with her close friends and family. To ensure her private photos are separate from her public postings, she has two accounts—one for the food blog and another for her personal photos. Whenever Carol wants to post a picture, she has to manually sign out of one account and sign in with the other. Every fresh login takes significantly long because Instagram has to cache the data from the servers, which are discarded when the user signs out. When Carols uses *Duvel*, data for both the accounts are present in their respective storage sandboxes, and switching is much faster. The app can even make an informed guess about which account Carol is likely to use. For instance, when Carol is away from home, she will be likely to use her personal account for her photos, while she is in her home, she may be likely to post to her food blog. Games are another category of apps where users tend to use multiple accounts. In particular, players of several war strategy games have multiple 'farm' accounts that are used to accumulate resources and supply to a 'main' account that is used for fighting.

Scenario 4, Multi-User: Dave is a typical mobile user and stores a lot of personal photos on his phone. Dave loves to show his friends and family the photos he captures. But when Dave's friends are viewing his photo gallery, he wants to hide all of his family pictures. *Duvel* enables this separation by allowing Dave to split his data into two profiles. Even though the photo app does not have multiple online accounts like the previous cases, *Duvel* would allow Dave to virtually partition his data into sandboxes. *Duvel* can later use context information, or a manual input from Dave to decide which of the many profiles in an app should be activated.

3 DESIGN SPACE

This section overviews the design space of multi-profile management systems. First, we will discuss some of the existing systems for managing multiple profiles, and then, we will discuss our envisioned system. Next, we enumerate the desired features in a multi-profile management system and compare our design with existing systems. And lastly, we will discuss some of the challenges involved in building our system.

3.1 System Design

As we discussed in Section 1, various EMM systems, allow users to access two different versions of a custom app. Google's 'Android Enterprise' brings support for multiple profiles for the Google suite of apps. We will compare the features of both these systems in Section 3.2.

One way to enable multi-profile for general apps is to allow a system-wide multiple accounts setting. This is the approach Android takes. A user can switch between their accounts, including the guest account, by signing off the system and signing back again with a different account. This would affect all the apps in the system e.g., when a user switches to a new account, they might want to do it only for the email app and use a common account for a to-do app. Another approach is to let the apps manage the profiles. Some apps, such as GMail App, allows a user to sign in with multiple accounts at the same time, but they work as a merged account rather than two separate accounts (e.g., when sending an email, the user has to choose which account to use). The third approach to enable multiple app profiles is to allow multiple versions of the app to run side by side in a virtualized environment. Popular apps like Parallel Space [16] take this approach. Because the entire app is running in a virtual environment, it results in higher overhead in terms of energy consumption, CPU usage, and memory footprint. Because of this, the number of instances of an app one can run inside the environment such as Parallel Space is limited to two.

While all of these approaches solves part of the problem of running multiple apps, each one of them has certain limitations. And none of them can offer a dynamic and policy-driven account switching, based on a context of usage (e.g., geolocation, network connectivity, user identification etc.), customizable by the user. In Section 3.1.1 we discuss the design of *Duvel* which handles these shortcomings.

3.1.1 *Duvel Design.*

To create a context-driven multi-profile manager, *Duvel* takes the approach of sandboxing only the storage layer of an Android app. Android apps, store all the user files, login information, access-keys, and everything required to identify a

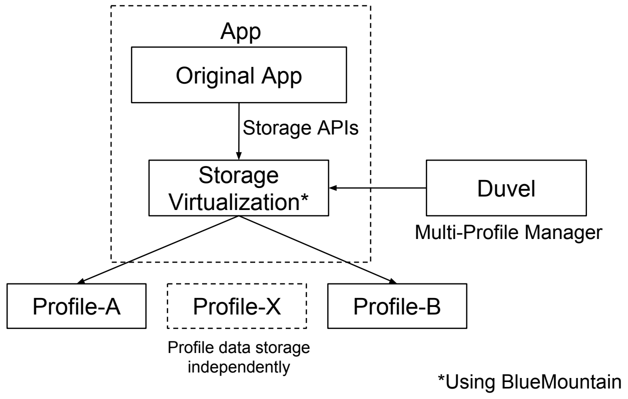


Figure 3: Architecture of *Duvel*

user in the private storage area as files or in a database. Our system exploits this design by letting apps to create multiple virtual storage spaces, within their private storage and to use them one at a time via storage redirection. *Duvel* intercepts all storage calls, and when the app tries to access say `file.txt`, *Duvel* redirects the read to `version-1/file.txt` or `version-2/file.txt`. This gives the app an illusion of having multiple versions of any given file.

To intercept the storage calls and create a storage sandbox within an app, *Duvel* uses BlueMountain [6], which in turn uses bytecode instrumentation to correctly intercept all the storage calls within the app[1]. Next, to handle the switching of profiles, *Duvel* has an app to work as BlueMountain's data-management app. This app has two components—(1) the storage class implementations (as required by BlueMountain) which get loaded into the instrumented app at use time, and (2) an interface through which a user can initiate switching profiles. The app can also implement automatic switch-over based on context information like such as geolocation, network connectivity, and others. To initiate a profile-switch, the data-management app sends an intent to the instrumented app, which changes storage sandbox which it is using. This architecture is shown in Figure 3. In the next section, we compare the features of our design with the existing systems.

3.2 Feature Comparison

Table 1 shows a comparison of BlueMountain with other systems that offer multiple profiles. The features are explained below:

1. **Profile isolation:** This feature isolates one profile from the other and is supported by all systems.
2. **Third party apps:** This feature allows a third party to develop apps that work with the given system. Three of the four systems allow this, but 'Android Enterprise' works only with Google suite and does not support third party apps.
3. **Simultaneous profiles:** This feature enables multiple profiles to be active at the same time. Currently, we are not

[1]When an app is instrumented with BlueMountain, the code to intercept all the storage calls (files, databases, and key-value store) is injected into the app. The injected code can also dynamically load the classes to handle storage calls from a special app, known as the data-management app. The user chooses which data-management app has to be loaded by a given app.

Solution	Profile Isolation	Third Party Apps	Simultaneous Profile	Unmodified Apps	Automatic Conversion	Dynamic Switching
Android Enterprise	✓	✗	✓	*N/A*	*N/A*	✗
AirWatch	✓	✓	✓	✗	✗	✗
Parallel Space	✓	✓	✓	✓	*N/A*	✗
Duvel	✓	✓	✗	✗	✓	✓

Table 1: Feature Comparison Table

able to support this feature with *Duvel*, but we are exploring ways of enabling this.

(4) **Unmodified apps:** This feature allows unmodified apps (in terms of source and binary) to directly work with the system. While *Duvel* requires apps to be modified, this is not a major concern (see next feature).
(5) **Automatic conversion:** Although unmodified apps cannot work directly on *Duvel*, we provide automated tools (based on BlueMountain) to modify the app (through byte-code instrumentation) to support *Duvel*.
(6) **Dynamic Switching:** This feature allows profiles to be switched automatically, based on a dynamic context. Only *Duvel* can support this feature.

3.3 Discussion

Context and Policy: To make the most of *Duvel*'s design, we should be able to use the context information to approximately predict which profile a user is likely to use. Second, the user should have enough flexibility to tweak the policies depending on their needs. We are studying some of the context-aware solutions that have been proposed in the past for our final design. Currently, we can enable simple context-switching such as ones based on the wireless AP the user is connected to.

Dynamic Switching: When the instrumented app is notified of the profile change over requests, it has to flush all the data to persistent storage, and restart using a different storage sandbox. When we try to automatically invoke the changeover, we will have to address data consistency issues that may arise if part of the data is not committed to storage at the time of switch-over.

Hardware based tracking: Google recommends developers to use software based GUID for tracking users instead of hardware based identifiers. When GUIDs are used, multiple sandboxed storages appear as different users. However, if an app uses hardware based identifier (IMEI, MAC, etc.), apps might be able to detect the use of virtualized storage and refuse to function. This would be a limitation of *Duvel*.

User Experience: One of the important challenges of adding functionality to apps through byte-code instrumentation is to maintain a consistent user experience. For instance, when we use automatic profile switch-over, users should be aware of the currently active profile. We are exploring different techniques like UI overlay, using notification bar, etc., to create a seamless experience.

Security: When *Duvel* instruments an app to enable multiple profiles, we need to ensure that the security of the app is not compromised. In particular, when using multiple profiles, we need to ensure that data is not leaked between profiles—as analyzed by AppFork [14], many apps leak data through the using shared storage. One way *Duvel* can deal with this problem is by keeping track of the data created in shared storage and making them accessible only when the profile that created the data is active. Another challenge is to provide enough flexibility for the apps and the users to create their own security policies (e.g., wipe data on a particular profile on multiple wrong pins). This would allow users to use apps that need high security like business apps.

4 DUVEL PROTOTYPE

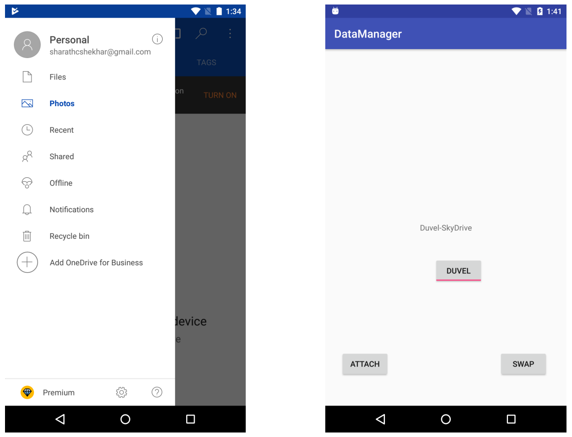
In this section, we will briefly discuss the implementation details of our prototype. We will then discuss the performance overhead of our system.

4.1 Implementation Considerations

The development of our system is divided into two parts: (1) We customized BlueMountain framework by adding functionality for profile switching. One of the implementation challenges at this step was during the app initialization—our system has to be fully initialized before any files are accessed by the app. While this was not a problem in simple apps, in complex apps having multiple threads, occasionally, we saw file access even before our system was initialized. We solve this by caching the I/O at start-up, until *Duvel* is ready to handle them. Another extension we wrote was to handle the files created by Webview based apps. These files are not created using standard storage APIs and BlueMountain could not handle them. (2) We developed the profile switchover functionality as a data-management app. Using the BlueMountain SDK, we could develop *Duvel*'s data-management app in less than 1000 lines of code.

4.2 Evaluation

To evaluate the working of our system, we developed a simple photo app, which can capture images, store them in the private app space, and display them in a gallery. We instrumented this app with BlueMountain and used it with *Duvel* data-management app. We could divide the photo app into multiple profiles and switch between them, both manually as well as based on the WiFi network to which the phone is connected. To demonstrate even complex



(a) Instrumented SkyDrive **(b) *Duvel* App to switch profiles**

Figure 4: SkyDrive working with *Duvel*

Photo App	Boot Time (ms)
Regular	149
With *Duvel*	278
After Switch	292

Table 2: Boot time in Milli-Seconds

Photo App	Write (ms)	Read (ms)
Regular	45	20
With *Duvel*	60	35

Table 3: I/O performance, 4MB Photo

apps can be handled by our system, we picked Microsoft OneDrive app and instrumented it using BlueMountain and tested it with *Duvel*.

4.2.1 Usage experience. With our current system, every time the profile is switched, we had to kill the app and restart it to reflect the changes. But while using the apps, both the photo app we developed and Microsoft OneDrive, we did not notice any lags within the app. The boot times of apps increased marginally when the app is instrumented with *Duvel* code. The boot time goes up even more when we restart after changing the profile. However, in our experience, this was not very perceivable to the user. The boot times are shown in Table 2.

4.2.2 Microbenchmarks. To measure the I/O overhead, we measure the read and write times for a 4MB photo from the custom photo app, both after instrumenting with *Duvel* and without instrumentation. These numbers are based on an average of 10 runs and are shown in Table 3. We can see that the performance overhead is modest. We intend to understand the source of the overhead, and reduce it in our future *Duvel* implementations.

5 RELATED WORK

Our current system is based on our previous work BlueMountain [6], a system that separates the data management logic from the app logic and uses BlueMountain transformer, a byte-code instrumentation tool based on Reptor [13].

Amongst the many existing works in the area of mobile data management, a system that shares many similarities and goals with *Duvel* is AppFork [14]. AppFork explores the area of BYOD both from a functional as well as a security standpoint and proposes many extensions to Android's framework to support the per-app user profiles. However, one of the goals of *Duvel* is to restrict all modifications to app-space in order to keep the system practical. Therefore while AppFork requires modifications to the underlying platform, *Duvel*'s approach enables easy deployment without disrupting the current ecosystem. However, *Duvel* can use some the ideas discussed in AppFork (e.g., preventing data leaks) and achieve this functionality in the app-space.

Additionally, *Duvel* also explores the idea of using context-awareness to create an adaptable storage, improving privacy, and adding flexibility on mobile systems. In this section, we discuss some of the prior work has explored these ideas.

Context-Aware systems: Many previous works have tackled context-awareness to provide a suitable service to the user depending on the context. On the mobile side, many context-aware storage systems have been build which can use context to cache data to location systems or use location based pre-fetching of data [11, 12, 17]. Other systems such as Encore [1] improve the privacy on mobile systems based on the context of the interacting devices. A general description of building a context-aware system has also been discussed here [4]. While all of these systems discuss novel ideas on gathering context, *Duvel*'s contribution is in using the gathered context to create an illusion of multiple versions of the same app running on a mobile phone.

App Sandboxing and Virtualization: Both commercial and research systems have explored creating virtualized environments on Android to increase security and add flexibility. Recent versions of Android allows multiple system-wide accounts on mobile phones. While this is equivalent to system wide sandboxing, *Duvel* allows multiple profiles at a finer granularity making the switching between app profiles much faster.

Various commercial Enterprise Mobility Management (EMM) systems, dedicated to managing user-owned devices in business environments have been developed in the recent past. Systems like Airwatch [18], XenMobile [8], Android Enterprise [10], etc., provide a suite which includes device management, access control, data management (e.g. Airwatch Content Locker), etc., enabling personal devices to securely access corporate data. These platforms typically require a custom enterprise app to take advantage of their solutions.

Systems such as Boxify [3], Condroid [7], and NJAS [5], Cells [2] create virtualization in app space where the entire app is working in a sandboxed environment. While, *Duvel* uses a similar idea to sandbox the storage system, unlike many of the virtualization environments, *Duvel* can dynamically switch between the different sandboxed environments. Also, by sandboxing only the storage layer, we suspect, *Duvel* might have a lower overhead compared to the full virtualized environments [14]. We intend to evaluate this and compare *Duvel*'s performance with other sandboxing systems in the future.

Byte-code instrumentation: Bytecode rewriting and instrumentation are popular techniques to modify an app's behavior without needing the source code. Many of the previous systems use these techniques [9, 15] for achieving a specific purpose, while Reptor [13] has built a general tool for byte-code instrumentation on Android. Although *Duvel*'s contribution is not towards the advancement of byte-code instrumentation techniques (we used previously developed systems BlueMountain and Reptor), like many systems based on byte-code instrumentation, *Duvel* had to solve many nuanced engineering challenges.

6 FUTURE WORK AND CONCLUSION

In this paper, we present a novel approach for sandboxing app data with practical use cases in enterprise mobility management, privacy, and multitasking. We have discussed the benefits of sandboxing data, the design space in general along with many use cases.

In our preliminary work, we have extended BlueMountain to achieve storage virtualization. While we have used a simple model based on network connectivity for deducing the context, our design opens up many possibilities of integrating intelligent context prediction systems to provide a rich user experience. Our future work is in the area of creating much more intelligent context prediction systems based on user identification, geolocation, and environment prediction.

7 ACKNOWLEDGMENTS

This work was supported in part by the generous funding from the National Science Foundation, CNS-1350883 (CAREER) and CNS-1618531. We would like to thank the anonymous reviewers for their valuable feedback. We also want to thank everyone working in our lab (Reliable Mobile Systems), for providing timely feedback.

REFERENCES

[1] Paarijaat Aditya, Viktor Erdélyi, Matthew Lentz, Elaine Shi, Bobby Bhattacharjee, and Peter Druschel. 2014. EnCore: Private, Context-based Communication for Mobile Social Apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14)*. ACM, New York, NY, USA, 135–148. https://doi.org/10.1145/2594368.2594374

[2] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. 2011. Cells: a virtual mobile smartphone architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 173–187.

[3] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp Von Styp-Rekowsky. 2015. Boxify: Full-fledged App Sandboxing for Stock Android. In *Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15)*. USENIX Association, Berkeley, CA, USA, 691–706. http://dl.acm.org/citation.cfm?id=2831143.2831187

[4] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 4 (2007), 263–277.

[5] Antonio Bianchi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. 2015. NJAS: Sandboxing Unmodified Applications in Non-rooted Devices Running Stock Android. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '15)*. ACM, New York, NY, USA, 27–38. https://doi.org/10.1145/2808117.2808122

[6] Sharath Chandrashekhara, Taeyeon Ki, Kyungho Jeon, Karthik Dantu, and Steven Y. Ko. 2017. BlueMountain: An Architecture for Customized Data Management on Mobile Systems. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom '17)*. ACM, New York, NY, USA, 396–408. https://doi.org/10.1145/3117811.3117822

[7] Wenzhi Chen, Lei Xu, Guoxi Li, and Yang Xiang. 2015. A lightweight virtualization solution for Android devices. *IEEE Trans. Comput.* 64, 10 (2015), 2741–2751.

[8] Citrix. 2017. XenMobile. (Jan 2017). Retrieved July 10, 2017 from https://www.citrix.com/products/xenmobile/

[9] Benjamin Davis, Ben Sanders, Armen Khodaverdian, and Hao Chen. 2012. I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications. In *Proceedings of the IEEE Mobile Security Technologies (MoST '12)*.

[10] Google. 2017. Android for Work. (Jan 2017). Retrieved July 10, 2017 from https://www.android.com/work

[11] Christopher K Hess and Roy H Campbell. 2003. A context-aware data management system for ubiquitous computing applications. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on.* IEEE, 294–301.

[12] Sharat Khungar and Jukka Riekki. 2004. A context based storage for ubiquitous computing applications. In *Proceedings of the 2nd European Union symposium on Ambient intelligence*. ACM, 55–58.

[13] Taeyeon Ki, Alexander Simeonov, Bhavika Pravin Jain, Chang Min Park, Keshav Sharma, Karthik Dantu, Steven Y. Ko, and Lukasz Ziarek. 2017. Reptor: Enabling API Virtualization on Android for Platform Openness. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, New York, NY, USA, 399–412. https://doi.org/10.1145/3081333.3081341

[14] Temitope Oluwafemi, Earlence Fernandes, Oriana Riva, Franziska Roesner, Suman Nath, and Tadayoshi Kohno. 2014. *Per-App Profiles with AppFork: The Security of Two Phones with the Convenience of One.* Technical Report. https://www.microsoft.com/en-us/research/publication/per-app-profiles-appfork-security-two-phones-convenience-one/

[15] Lenin Ravindranath, Sharad Agarwal, Jitendra Padhye, and Chris Riederer. 2014. Procrastinator: Pacing mobile apps' usage of the network. In *Proc. ACM MobiSys.*

[16] Parallel Space. 2017. Parallel App: Run Multiple Social and Game Accounts in Your Phone Simultaneously. (Jan 2017). Retrieved April 6, 2018 from http://parallel-app.com/

[17] Patrick Stuedi, Iqbal Mohomed, and Doug Terry. 2010. Wherestore: Location-based data storage for mobile devices interacting with the cloud. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 1.

[18] VmWare. 2017. Air-Watch Enterprise mobility platform. (Jan 2017). Retrieved July 10, 2017 from https://www.air-watch.com/