# Enforcing Fine-Grained Security and Privacy Policies in an Ecosystem within an Ecosystem

Waqar Ahmad Joshua Sunshine
Christian Kaestner
School of Computer Science
Carnegie Mellon University, USA

Adam Wynne
Bosch Research and Technology Center, USA

## Abstract

Smart home automation and IoT promise to bring many advantages but they also expose their users to certain security and privacy vulnerabilities. For example, leaking the information about the absence of a person from home or the medicine somebody is taking may have serious security and privacy consequences for home users and potential legal implications for providers of home automation and IoT platforms. We envision that a new ecosystem within an existing smartphone ecosystem will be a suitable platform for distribution of apps for smart home and IoT devices. Android is increasingly becoming a popular platform for smart home and IoT devices and applications. Built-in security mechanisms in ecosystems such as Android have limitations that can be exploited by malicious apps to leak users' sensitive data to unintended recipients. For instance, Android enforces that an app requires the Internet permission in order to access a web server but it does not control which servers the app talks to or what data it shares with other apps. Therefore, sub-ecosystems that enforce additional fine-grained custom policies on top of existing policies of the smartphone ecosystems are necessary for smart home or IoT platforms. To this end, we have built a tool that enforces additional policies on inter-app interactions and permissions of Android apps. We have done preliminary testing of our tool on three proprietary apps developed by a future provider of a home automation platform. Our initial evaluation demonstrates that it is possible to develop mechanisms that allow definition and enforcement of custom security policies appropriate for ecosystems of the like smart home automation and IoT.

*MobileDeLi'15*, October 26, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3906-3/15/10...$15.00
http://dx.doi.org/10.1145/2846661.2846664

***Categories and Subject Descriptors*** D.2.11 [*Software Architectures*]: Domain-specific architectures; D.2.4 [*Software/Program Verification*]: Model checking; D.4.6 [*Security and Protection*]: Access controls

***General Terms*** Security, Design

***Keywords*** Ecosystem, app store, app, security, privacy, fine-grained policies, home automation, Internet of things, Android

## 1. Introduction

Smart phone platforms, such as Android, are increasingly being used on a variety of smart devices in addition to mobile phones and tablets. For instance, Android devices include TVs, cameras, glasses, watches, headphones, DVD players, video and game consoles [10]. Increasingly, Android is considered as platform for smart home automation and IoT devices.

Smart home automation and IoT bring a lot advantages but they also expose their users to security and privacy vulnerabilities. For instance, an app on a smart home device may upload user's private data on a server that may use the data for unintended purposes. Such data leakage may result in serious security and privacy threats for users as well as serious legal implications for providers of the home automation and IoT platforms. Since smart devices may be running a variety of apps developed by a variety of developers, it will be necessary to sandbox the apps in order to ensure privacy and security.

We envision that an ecosystem within a smartphone ecosystem will be the platform for distributing apps for smart home and IoT devices. Considering that smart home and IoT apps will have access to user's sensitive private data, it is necessary to implement security and privacy policies in order to avoid data leakage to unintended recipients. App stores, such as Google Play, are primary source of apps for smart phones and tablets only. Google Play defines certain standard policies for apps that are uploaded on the store and the Android platform offers permission system [4] in order to restrict what an app can do. However, these mechanisms are coarse grained; for instance, if an Android app has Internet permission, it can share a user's data with anyone without any constraints. Moreover, apps can maliciously collaborate to circumvent security mechanisms in order to transmit data to unintended recipients [7–9].

In addition to the above mentioned security and privacy concerns, home automation and IoT platforms also present their own security challenges. Home automation and IoT apps are different from common smart phone apps because they interact with physical world and therefore, cybersecurity problems can easily become physical security problems (e.g., unlocked doors). Additionally, communication between apps on different devices is necessary to enable the full rich possibilities of home automation and IoT but we don't want this communication to be malicious. Therefore, we propose an ecosystem that enforces additional fine-grained policies on top of existing policies of the smartphone ecosystems for smart home or IoT platforms. This will enable home automation and IoT platforms to customize access of their resources to certain apps only with stricter permissions.

To this end, we have designed a tool that relies on static and dynamic analysis techniques to extract inter-app interactions and permissions from Android apps and filters the apps that do not comply with some pre-defined policies. We have tested our tool on three proprietary apps developed by a provider of an under development home automation ecosystem. Our initial evaluation demonstrates that it is possible to develop mechanisms that allow definition and enforcement of custom security policies appropriate for ecosystems of the like smart home automation. Moreover, such a tool can be integrated with the ecosystem for automatic enforcement of desired policies. Following are our specific contributions:

- We propose the idea of ”an ecosystem within an ecosystem”, wherein fine-grained security and privacy policies are enforced in the ecosystem within the main ecosystem.
- We build a tool that statically extracts inter-app interactions and permissions from Android apps and filters the apps that do not comply with pre-defined policies.
- We perform a preliminary evaluation of our concept on three real Android apps developed by a provider of a home automation ecosystem. The tool serves as proof of concept for definition and enforcement of fine-grained security and privacy policies appropriate for ecosystems of the like smart home automation.

In the future, we plan to develop mechanisms for definition of fine-grained security and privacy policies and build more sophisticated static and dynamic analysis techniques to enforce more security and privacy properties for apps of home automation and IoT ecosystems.

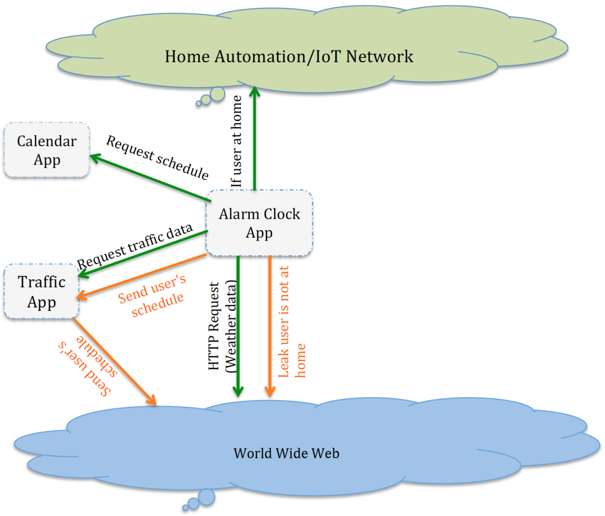

**Figure 1.** The Alarm Clock app needs Internet permission for downloading weather data and needs to talk to Traffic and Calendar apps for schedule and traffic information.

## 2. Problem Statement

Smart home devices often run on general purpose software platforms such as standard or custom versions of Android. These platforms provide built-in security mechanisms that are generally coarse-grained and do not enforce fine-grained security and privacy policies specific to platforms such as home automation and IoT.

Android provides inter-app interaction mechanisms (henceforth ”inter-app APIs”) that enable apps to interact with each other and exchange data during such interactions. Moreover, Android apps can communicate with outside world using a variety of APIs such as HTTP and SMS APIs. Android has a built-in security system that enforces certain restrictions on apps. For instance, Android's security system controls access to resources such as the Internet and user's private data (e.g., contacts) by requiring an app to declare permissions for the resources it needs. Moreover, apps may define their own permissions and thus restrict how they expose their own functionality. App users need to approve these permissions at the time of installation. However, this security system has limitations in the sense that if an app has certain permissions to perform some actions, there are no further constraints on how that app uses those permissions. As a result, Android apps having restricted permissions can maliciously collaborate through inter-app APIs in order to circumvent the Android security framework [7–9]. Therefore, two aspects, i.e., an *app's permissions* and its *inter-app interactions* are important characteristics of an app to analyze its security and privacy loopholes.

Security and privacy for home automation and IoT platforms is of critical importance. In order to explain their significance, we present here a scenario. Imagine an Alarm

Clock app, provided by an untrusted third party, installed on a smart phone device connected to a home automation system (see Figure 1). The app rings the alarm to wake up the user and switches on the lights but only when the user is at home. In order to make intelligent decisions about when to ring the alarm, the Alarm Clock app makes adjustments based on the user's schedule and the weather and the traffic conditions. Therefore, the Alarm Clock app communicates with a Calendar and a Traffic app on the smart phone device to access the user's schedule and the traffic information respectively, and connects with an online web service to download the weather data.

The above example presents a simple case in the context of broad and complex home automation ecosystems that have started to appear or being envisioned for the future. This system has obvious great benefits. However, there are some security and privacy pitfalls associated with it. The system has access to private data of the person at home, which can be used for malicious purposes. For instance, the app knows whether a user is at home; if this information is leaked, it may instigate an intruder to perform a burglary at a suitable time. The app has also access to the users' daily schedules, which is private information that users may want to share with only a limited number of people in their circles. Moreover, the Alarm Clock app connects to the Internet for accessing weather data, where it can easily leak this information to unintended recipients. Additionally, the Alarm Clock app may not be doing anything that appears to be malicious, but it may still be passing the user's private data to the Traffic app (some third party app) which then forwards the information to malicious recipients.

The current Android security framework does not address such security and privacy issues. For instance, the Android app in the example above that downloads weather data from a web service needs the Internet permission. The Android platform only enforces whether the app has the Internet permission and does not check how the app uses it [4]. Therefore, the very app that has got the Internet permission for legitimate reasons, may misuse it to leak user's private data, resulting in security and privacy threats to the user. There may also be other apps (e.g., the Traffic app) with whom the Alarm Clock app can share user's private data and Android does not perform any checks on what data is being exchanged across apps. Moreover, there could be a variety of smart Alarm Clock apps with different features available in an ecosystem for users to choose from and there may not be any straightforward mechanisms to know which of the those apps are malicious.

Therefore, additional mechanisms are required in order to address potential security and privacy threats arising specifically due to nature of home automation and IoT platforms. We envision that there will be a home automation ecosystem inside the Android ecosystem with stricter and more fine-grained policies. We propose to build mechanisms for defining fine-grained security and privacy policies written by the home automation ecosystem provider and develop static and dynamic analysis techniques that guarantee enforcement of those policies. This will empower the ecosystem owners to define and enforce policies instead of leaving security and privacy in the hands of app developers and deliver only certified apps to home automation and IoT devices from their app stores.

We have chosen Android for our further research and development because it is increasingly becoming popular platform for smart home devices, in addition to phones and tablets [10]. Moreover, instead of building a solution from scratch, we plan to reuse existing security framework of Android and build our solution on top of it, and this will also let users to install all kinds of Android apps in addition to those available from IoT and home automation ecosystems only.

## 3. Proposed Solution

We propose a design for an ecosystem for home automation and IoT platforms within the broad Android ecosystem. Figure 2 provides a high level view of the proposed solution. The left hand side in the Figure 2 gives the home automation and IoT app store view and the right hand side gives the runtime view of an app on a smart phone device.

All apps for the home automation or IoT ecosystem are installed from its own app store, which follows stricter security and privacy policies defined by the home automation or IoT ecosystem provider. The home automation or IoT ecosystem policies will typically disallow access to security and privacy sensitive Android APIs (e.g. HTTP APIs) by restricting an app's permissions. The policies are enforced partially at the home automation or IoT app store using static analysis techniques and partially at a smart home device using dynamic checking of runtime behavior of an app. Only the apps that comply with home automation or IoT ecosystem policies are published on the IoT app store. Moreover, only an app that has been downloaded from the home automation or IoT app store gets access to the home automation and IoT Framework APIs; this is enforced using the app signing mechanism available in the Android.

Protected versions of the sensitive APIs are made available through alternative home automation and IoT Framework APIs (e.g. IoT HTTP APIs) with dynamic enforcement of ecosystem policies. For instance, if an app needs to access the Internet, instead of using the Android HTTP APIs, it uses the HTTP APIs provided by the home automation and IoT Framework that restricts, at runtime, the websites that can be accessed or the data that can be exchanged with the web server. Any direct attempt to access Android's sensitive APIs by an app is rejected by the Android operating system because the app does not have required permissions. A coarse-grained policy such as the Android Internet permission does not define constraints such as the websites an app is allowed

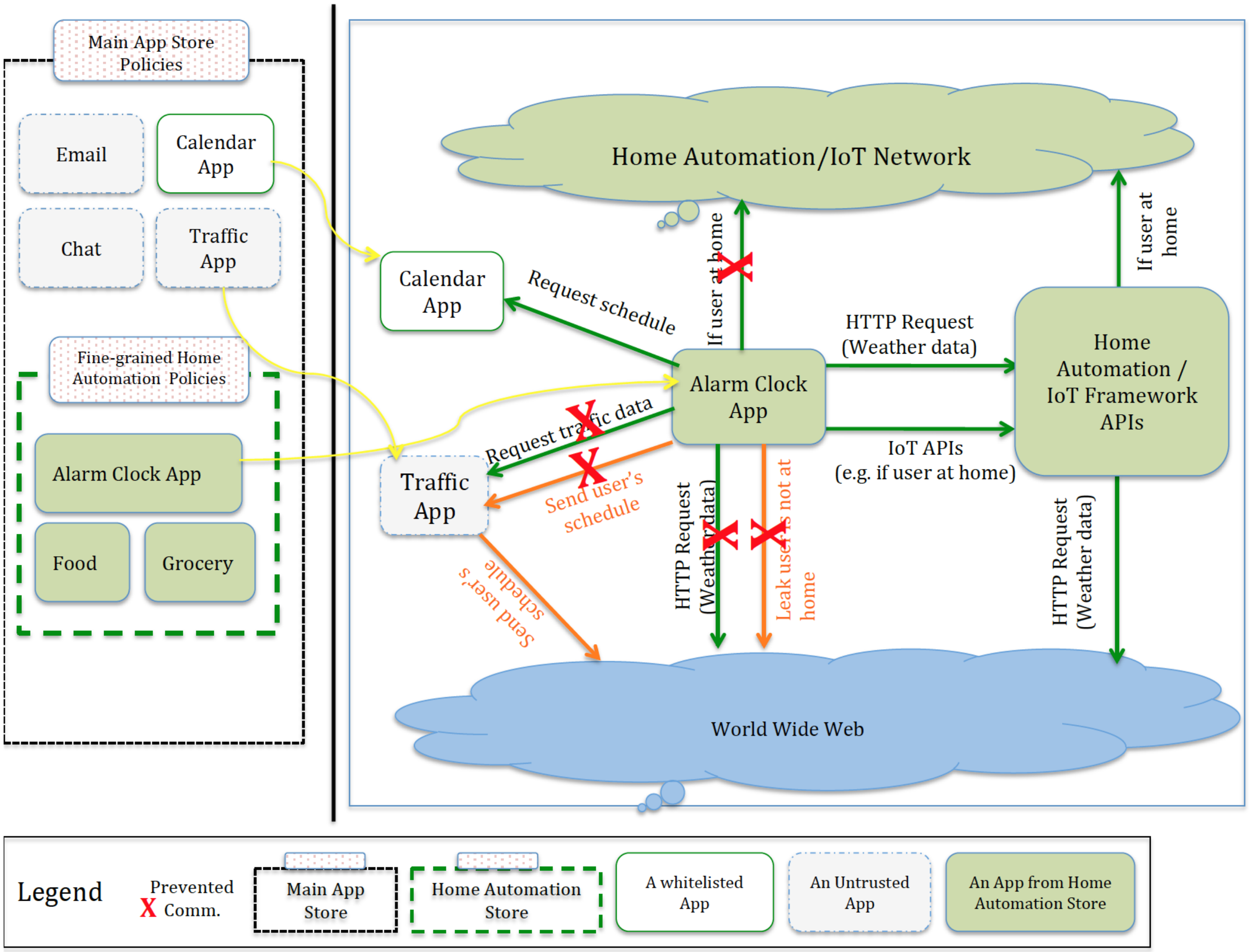

**Figure 2.** On the left hand side are the apps published on the main app store and the apps on the home automation or IoT app store On the right hand side is a smart phone device where the Alarm Clock app, downloaded from home automation app store, is installed. As an app on home automation platform, it is not allowed to access sensitive APIs but it can use alternative APIs provided by the home automation and IoT Framework.

to talk to or the data it is allowed to upload or download. A fine-grained policy, on the other hand, additionally specifies the websites and data constraints e.g., URLs that an app is allowed to access. These fine-grained constraints are dynamically enforced by the home automation and IoT framework.

The interaction of a home automation and IoT app with other apps is also restricted. An app is allowed to communicate either with other apps from home automation and IoT ecosystem or with a whitelist of trusted apps. Some apps (e.g., a Calendar app from a trustworthy developer) may not be in the home automation and IoT ecosystem but are whitelisted because they are known to be not malicious. This is enforced by requiring apps to use only the explicit intents for inter-app interactions and is checked statically at the app store level.

The left hand side in Figure 2 illustrates the idea of a sub-ecosystem within another ecosystem for our running example. A subset of the apps filtered through stricter security policies has been published on home automation app store. On the right hand side in the Figure 2 is a smart phone device (e.g., an Android device) where the Alarm Clock app (the example App discussed earlier) has been installed from the home automation app store. As an app on the home automation platform, it is not allowed to directly access sensitive Android APIs (e.g., HTTP APIs) and sensitive home information (e.g., if user is at home). In order to access such information, it is, however, allowed to communicate with the home automation and IoT framework APIs that dynamically check the data being exchanged. The Alarm Clock app can communicate with whitelisted apps (e.g. Calendar app) but is not allowed to talk to untrusted apps (e.g. the Traffic app).

In order to enable access of the traffic information to home automation and IoT apps, either a traffic app within home automation and IoT ecosystem is needed or an app from outside the ecosystem needs to be whitelisted.

The prototype tool that we have developed implements the proposed solution in the following way:

- Restrict an app to a limited whitelist of permissions or no permissions at all.
- Restrict an app to a limited whitelist of other apps with whom they can interact explicitly.
- Enable an app to access privileged resources, if required, only through APIs provided by home automation ecosystem framework.

## 4. Current Work

In order to realize our vision of fine-grained policy definitions and enforcement in home automation and IoT ecosystems, we have started building necessary tools and technologies. So far, we have built a tool that extracts inter-app interactions and permissions from Android apps and filters the apps that do not comply with some pre-defined policies. The tool serves as a prototype for privacy and security policy definition and enforcement in home automation and IoT platforms.

Our current tool has two modules. One module is repository of policies. The other module comprises of programs that statically analyze Android apps in order to check if they comply with predefined policies and report violations, if any. The current tool has support for the definition and enforcement of two types of policies, i.e., permissions whitelist and inter-app components whitelist.

*The Permissions whitelist policy* defines a list of permissions that are considered safe and therefore, apps are permitted to have them. Users of the tool can define a whitelist of permissions in a configuration file. Our static analysis tool identifies the apps that have permissions other than those defined in the whitelist of permissions. A provider of the home automation or IoT ecosystem can use this tool to statically enforce this policy at its app store. At runtime, this policy is enforced by the Android platform by disallowing an app to access any APIs for which it has not declared permissions in its manifest file.

*The Inter-app components whitelist policy* defines a list of Android components (i.e., components that exist in other apps) that are considered safe and therefore, apps are permitted to communicate with them using the Android inter-app APIs. Users of our tool can define a whitelist of third party app components. Our static analysis tool extracts inter-app interactions, i.e., which other apps an app may communicate with and what data may be exchanged. If extracted inter-app components do not exist within the current app or in the whitelist of components or cannot be statically determined, violations are reported. In order to control malicious data flows from one app to another, an ecosystem provider may want to restrict apps to communicate with only a predefined whitelist of apps. This feature can help ecosystem providers achieve this goal.

There are many ways inter-app interactions may occur in Android. For instance, inter-app APIs, content providers, shared preferences etc. In current work, we have focused on inter-app APIs. There are a variety of methods in inter-app APIs such as various overloaded versions of startActivity, startService, bindService, and sendBroadcast. Our static analysis tool targets these methods and extracts data being exchanged. Parameters that are exchanged across apps are generally passed as instances of the Android class *Intent*. We keep track of values of data members of interest (i.e., context and class) in *Intent* instances. Moreover, since *Intent* instances may be modified using a number of methods in the *Intent* class, we maintain side-effects of each relevant method of the *Intent* class in a file. Method side-effects refer to the data members affected by a method call and how those data member are affected (added, overwritten).

In order to extract inter-app interactions, we have implemented our static analysis tool as an intra-procedural data flow analysis program using the frameworks soot [2] and FlowDroid [1]. We use FlowDroid [1] to build an app's call graph that respects the Android lifecycle and then use the soot framework [2] to perform data flow analysis. In Android, inter-app interactions may be implicit or explicit. There is no straightforward mechanism to know which other apps an app can communicate using implicit intents. Therefore, we enforce that only explicit intents are used by an app. We extract following data of an *Intent* instance being passed to an inter-app API function: context and class. In an explicit *Intent* call, context and class uniquely identify the target component of an inter-app call. We perform conservative data flow analysis of Android apps. We keep track all the intent instances that may possibly be passed to an inter-app API function and if on any program flow, we are unable to extract the target app of inter-app call, we report the app as insecure. This might result in some false negatives but it does ensure that only the trusted apps pass through the policy enforcement and are published on home automation or IoT app store.

In order to ensure future extensibility of our tool, we have made following configurable:

- List of methods of interest in Android inter-app APIs (e.g. startActivity, startService, bindService etc.)
- Proprietary APIs. Apps may use some APIs that are trusted by a home automation/IoT ecosystem provider. An ecosystem provider may list such API methods here.

The current tool is in a prototype state and serves as a proof of concept for our broad vision. In the future, we plan to extend this tool with fine-grained policies that can be checked statically at app store and enforced dynamically at smart home devices. We may also use or extend existing tools such as Epicc[8], FlowDroid[1], IC3[9], and DroidSafe[5]. Though these tools have purposes different from ours, they can be used to extract useful information (e.g., inter-app parameters) about apps.

## 5. Related Work

Our work is related to existing work in static and dynamic information flow analysis. There are quite a few tools that perform static information flow analysis of an Android app from a variety of perspectives. FlowDroid [1] is a static taint-analysis tool that models the Android lifecycle methods and callbacks in order to extract information about data that flows from a source (e.g. location APIs) to a sink (e.g. SMS APIs) within an Android app. Epicc [8], IccTa [7], IC3 [9], and DroidSafe [5] perform information flow analysis of Android apps from inter-component or inter-app interaction perspectives. While aforementioned tools provide a variety of useful information such as what data may potentially be exchanged using Android APIs and in inter-component or inter-app interactions, they do not define a way to distinguish between malicious and non-malicious data exchanges. Our strategy, on the other hand, is to take a step further by distinguishing between trusted and untrusted apps. We plan to achieve this by defining security policies and enforcing them at app store of the ecosystem provider through static analysis techniques and at smart home devices through dynamic techniques.

There is also some related work in the domain of dynamic information flow analysis. Fragkaki et al. suggest a framework for formally analyzing permission system in Android and provide mechanisms to modify existing Android permissions system to support rich security policies [3]. Jia et al. proposes mechanisms for defining and enforcing information flow policies for Android apps [6]. Their perspective, however, is different from ours. While they suggest making changes in existing security system of the Android platform in order to enable developers define fine-grained permissions, we focus on detecting permission misuse by apps by letting ecosystem providers define and enforce security and privacy policies and restrict app developers to a whitelist of safe permissions and inter-app interactions without any modifications in the Android.

NaCl [11] combines static and dynamic techniques to ensure secure execution of untrusted x86 native code in a web browser. NaCl validator statically analyses code to make sure that it only uses code and data patterns that are safe and NaCl sandbox serves as a runtime check. Our approach, however, is to restrict apps to a whitelist of permissions and inter-app interactions and force them to go though a framework provided by ecosystem provider for accessing privileged data and APIs at runtime.

## 6. Conclusion and Future Work

Smart home automation and IoT promise to bring a lot advantages but they also expose their users to certain security and privacy vulnerabilities. We envision that an ecosystem within broad smartphone ecosystem will be the platform for distribution of apps for smart home and IoT devices. Such a sub-ecosystems can enforce additional fine-grained custom policies on top of existing policies of the smartphone ecosystems. In this paper, we have explained the importance of security and privacy in home automation and IoT platforms and proposed a solution to enforce custom security and privacy policies. We have also built a prototype tool to demonstrate that it is possible to develop mechanisms that allow definition and enforcement of custom security policies appropriate for ecosystems of the like smart home automation and IoT. In the future, we plan to develop additional mechanisms for definition of fine-grained security and privacy policies and build more sophisticated static and dynamic analysis techniques to enforce those policies. As part of the custom policy definition, we intend to work on creation of a policy language that makes definition of sophisticated fine-grained policies possible.

## Acknowledgments

We are thankful to Dr. Jonathan Aldrich for his continuous guidance throughout this research work.
This paper is based on the work supported by Stevens Grant No. 33649.1.1042344 and Bosch Research and Technology Center North America.

## References

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 14, New York, NY, USA*, pages 259–269, 2014.

[2] A. Einarsson and J. D. Nielsen. A survivor's guide to java program analysis with soot. page `http://www.brics.dk/SootGuide/sootsurvivorsguide.pdf`, 2008.

[3] E. Fragkaki, L. Bauer, L. Jia, and D. Swasey. Modeling and enhancing android's permission system. *Computer Security, ESORICS 2012: 17th European Symposium on Research in Computer Security*, 7459 of Lecture Notes in Computer Science:1–18, 2012.

[4] GoogleAndroid. System permissions. *Android Developer Guide*, page `http://developer.android.com/guide/topics/security/permissions.html`, 2015.

[5] M. Gordon, D. Kim, J. Perkinsa, L. Gilhamy, N. Nguyen, and M. Rinard. Information-flow analysis of android applications in droidsafe. *Proc. of the Network and Distributed System Security Symposium (NDSS), The Internet Society*, 2015.

[6] L. Jia, J. Aljuraidan, E. Fragkaki, L. Bauer, M. Stroucken, K. Fukushima, S. Kiyomoto, and Y. Miyake. Run-time enforcement of information-flow properties on android. *Computer Security, ESORICS 2013: 18th European Symposium on Research in Computer Security*, pages 775–792, 2013.

[7] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, , and P. Mcdaniel. Iccta: Detecting inter-component privacy leaks in android apps. *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, 2015.

[8] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. *Proceedings of the 22Nd USENIX Conference on Security, SEC'13, Berkeley, CA, USA*, pages 543–558, 2013.

[9] D. Octeau, D. Luchaup, M. Dering, S. Jha, , and P. McDaniel. Composite constant propagation: Application to android inter-component communication analysis. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015.

[10] Wikipedia. Android (operating system). page `https://en.wikipedia.org/wiki/Android_(operating_system)`, 2015.

[11] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. *IEEE Symposium on Security and Privacy*, 2009.