The thesis of Xuan Wu was reviewed and approved* by the following:

Sencun Zhu
Associate Professor of Department of Computer Science and Engineering &
College of Information Science and Technology
Thesis Advisor

Kyusun Choi
Associate Professor of Department of Computer Science and Engineering

Lee D. Coraor
Associate Professor of Department of Computer Science and Engineering
Graduate Program Chair of Department of Computer Science and Engineering

*Signatures are on file in the Graduate School
ABSTRACT

Prevalence of smartphones is growing among children. It presents threats to children. Children’s unlimited access to smartphones may become study distraction. Even worse, children are at the risk of privacy disclosure when they are using smartphones. Therefore, parental control is necessary on smartphones. However, as the most popular mobile operating system, Android provides little parental control features. Existing parental control applications on Android are vulnerable and it is difficult to develop a good parental control application on Android. In this thesis, we study parental control issues on smartphones and propose Simple Parental Control (SPC) to address them on Android. SPC is fine-grained and context-based. According to our experiments, SPC introduces little power consumption and restricts applications’ access to specific resources without compromising other functionality.
# TABLE OF CONTENTS

List of Figures ........................................................................................................ vi
List of Tables ........................................................................................................... vii
Acknowledgements ................................................................................................. viii
Chapter 1 Introduction ............................................................................................. 1
Chapter 2 Background ............................................................................................. 4
  2.1 Android architecture ....................................................................................... 4
  2.2 Android application components ................................................................. 5
  2.3 Communication between applications ......................................................... 5
  2.4 Android permissions ..................................................................................... 7
Chapter 3 Related work ............................................................................................ 8
  3.1 Android security improvement .................................................................... 8
  3.2 Parental control on Android ....................................................................... 10
Chapter 4 SPC design ............................................................................................ 12
  4.1 Design goals ................................................................................................. 12
  4.2 Usage scenarios ........................................................................................... 13
  4.3 Attack model ............................................................................................... 14
  4.4 Design options ............................................................................................. 15
  4.5 SPC model .................................................................................................... 17
  4.6 Architecture .................................................................................................. 21
Chapter 5 SPC implementation ............................................................................ 27
  5.1 SPC server .................................................................................................... 27
  5.2 Implementation of SPC on parents’ phones ................................................. 28
  5.3 Implementation of SPC on children’s phones ............................................. 31
Chapter 6 Evaluation ............................................................................................. 37
  6.1 Power consumption ...................................................................................... 37
6.2 Impact on applications .......................................................................................... 39

Chapter 7 Discussion .................................................................................................. 41

7.1 SPC attacks and defenses .................................................................................... 41
7.2 Automatically determine contexts sensitivity .................................................... 41
7.3 Parental control on iOS ....................................................................................... 42

Chapter 8 Conclusion ............................................................................................... 44

Reference ................................................................................................................... 45
LIST OF FIGURES

Figure 2-1 Android software stack ................................................................. 4
Figure 2-2 Android Inter-Process Communication ............................................. 6
Figure 3-1 Kirin architecture ............................................................................. 8
Figure 3-2 TaintDroid architecture ................................................................... 9
Figure 4-1 A usage scenario ............................................................................. 13
Figure 4-2 Resource check ............................................................................... 22
Figure 4-3 Launching check ............................................................................. 22
Figure 4-4 SPC Architecture .......................................................................... 23
Figure 5-1 SPC server ..................................................................................... 28
Figure 5-2 Mark unsafe regions for children ...................................................... 29
Figure 5-3 Select children's accounts ................................................................. 29
Figure 5-4 Select affected applications .............................................................. 30
Figure 5-5 Set time interval ............................................................................. 31
Figure 7-1 iOS restriction .................................................................................. 43
LIST OF TABLES

Table 4-1 SPC rules description ................................................................. 20

Table 6-1 SPC battery usage ...........................................................................38

Table 6-2 Applications in the experiment .........................................................39
ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Sencun Zhu for his patience, encouragement and guidance on my thesis.

Besides, I would like to thank Dr. Kyusun Choi for joining my thesis committee and insightful questions.

Last but not the least, I would like to thank my family: my parents Xiaokun Wu and Yongping Li, for supporting me spiritually throughout my life; my uncle and aunt, Zhiyong Guo and Yonghong Li for taking care of me in recent two years; my girlfriend, Jia Song, for always encouraging and supporting me.
Chapter 1

Introduction

Smartphones are one of the most popular consumer devices to date. As of 2013, 65 percent of U.S mobile consumers own smartphones [28]. The European mobile device market as of 2013 is 860 million [29]. In China, estimated smartphones shipments will exceed 450 million by 2014 [30]. Popularity of smartphones is also booming among children because children as well get their small hands on a smartphone at some point. *Lookout*, an information security company, and *The Online Mom*, a company providing technology and Internet advices for parents, together published a report [14]. It states that 56 percent of parents of kids 8-12 say their children own a smartphone. A survey commissioned by *PBS Kids* [15] found that 54% of parents plan to purchase or give a technology gift to their children in Christmas holiday season 2013. In addition, nearly 70% of parents are willing to give their kids applications and 90% of parents believe that educational applications will play an important role in children’s learning in the future. Researchers also found that educational applications could help children learn; one application they tested boosted vocabulary score as much as 27% [31]. With the growing prevalence of smartphones among children, children have more chance to learn and entertain themselves.

However, other than learning and entertaining opportunities, smartphones present threats to children. Children’s unlimited access to smartphones may become study distraction. Even worse, children’s privacy can be exposed online by applications. According to *NBC Action News*, smartphone picture uploads can reveal the locations of children’s home, school, and play areas [18]. Smartphones turn out to be much easier than any other digital devices to leak private information since the contents sharing nature. Children are also at risk of online sexual and violent materials on smartphones and it is much easier for online predators to locate children than through traditional ways [27]. In order to protect children from these threats, a straightforward way is to remove all the potential dangerous features on children’s smartphones such as location and network services. However, without these features, children may lose entertaining and learning opportunities too and smartphones will not be “smart” any more.
Parents are becoming more aware of smartphones’ threats with increasing worries [32]. On the one hand, parents are worried about their children’s unlimited access to smartphones since it may distract children from their normal study; on the other hand, they are worried that applications’ access to children’s sensitive data since it may cause privacy disclosure. In order to alleviate parents’ worries on threats from their children’s smartphones, parental control is necessary on smartphones.

One important observation for parental control on smartphones is that privacy issues largely depend on the context [25]. For example, in a mobile environment, disclosing children’s photos and locations to strangers is much more dangerous in outdoor environments than in classrooms. Therefore, context-based access control should be included in parental control. Another observation is that parents usually do not want to disallow an application to run. Instead, parents just want to restrict some of the application’s behaviors. For example, parents may worry about map applications will expose their children’s locations online while understanding their children may need these apps to check maps. Hence, instead of application-level coarse-grained access control, parental control should include fine-grained data access control.

As the most popular mobile operating system with 72.4% share of the total sold smart devices, Android provides little parental control features mentioned above. Some parental control applications in app markets offer to help. However, these parental control applications cannot manage applications’ access to resources and they are vulnerable because even children can easily bypass rules set by their parents.

In order to address the parental control issues on Android, in this thesis, we present a security module, *Simple Parental Control (SPC)*, which provides contexts awareness and fine-grained control over Android resources. With SPC, parents can easily generate security rules on their phones and push them to children’s phones over the air. More specifically, SPC consists of three parts.

(1) *SPC* on parents’ phones that helps parents generate parental control rules.
(2) *SPC server* that transfers and authenticates messages between parents’ phones and children’s phones.

(3) *SPC* on children’s phones that is implemented in the Android middleware layer. It monitors and collects contexts information and restricts children’s and applications’ access to their phones based on contexts.

With *SPC*, when a third-party application tries to access sensitive resources managed by *SPC*, before the resource is returned to the application, it has to pass *SPC resource check*. Besides, *SPC* model supports fine-grained data access control. When an application tries to access sensitive resources, instead of forcing the application to quit, *SPC* can allow the application to run but feed it with mocked data. *SPC* rules can be incrementally added into policy database so that parents do not have to generate all the rules in one shot. *SPC* can also automatically resolve conflicts among rules.

**Contributions.** In this thesis, we provide the following contributions:

1. We design and implement *SPC*. *SPC* enables parents to define and enforce fine-grained and context-based parental control rules over the air for their children.

2. With *SPC*, parents can push parental control rules to their children’s phones over the air.

3. In our implementation, *SPC* takes two types of contexts, location and time, into consideration. According to our experiments results, all the 100 applications in experiments can still run with potential threats eliminated by *SPC*. Additionally, *SPC* only slightly increases power consumption.
Chapter 2

Background

2.1 Android architecture

Android is built on Linux Kernel. Figure 2-1 shows Android software stack. Linux kernel is at the bottom of Android software stack. Above the kernel are native libraries and Android runtime. Built on top is Application Framework, which enables Android applications to interact with Core libraries and Linux kernel. Android includes a set of Core Libraries that provides most of the functionality in application programming interfaces (APIs) [26]. Android applications are written in Java. These applications then are compiled into Java class files. However, Android does not run these Java class files as they are. Java class files then are compiled into dex files, which are running in Dalvik Virtual Machine.

![Android software stack diagram](image)

Figure 2-1 Android software stack

Android applications must run in their own Dalvik VMs. Developers must sign their applications before releasing them to Android applications markets. Different Dalvik VMs cannot run in the same Linux process unless these applications have the same signature.
2.2 Android application components

Android applications consist of Android applications components. There are four types of components in an Android application, *Activity, Service, Content Provider, BroadcastReceiver* [11].

*Activity* is the user interface of an application. Basically, every ‘screen’ on Android is an *Activity*. An Android application usually has multiple *Activities*. An *Activity* can be launched from the Android launcher (Android home screen) or any other *Activities*.

*Service* is running in the background and it is invisible to phone users. For example, on Android, we can use the browser application to surf the Internet and at the same time, we can listen to music. The browser user interface we are looking at is an *Activity*, and the component playing music in the background is a *Service*.

*ContentProvider* is an interface to the database on Android. It provides a way to persist data on Android. Developers use it to communicate with the database on Android. *ContentProvider* is usually transparent to Android phone users.

*BroadcastReceiver* is an asynchronous event mailbox for *Intent* messages. *Intent* is the data structure through which Android components communicates with each other. We can regard *Intents* as messages stating which operations or actions need to be performed. Android will route *Intents* to their corresponding *BroadcastReceiver*. A *BroadcastReceiver* listens for specific *Intent* messages. When an *Intent* message is routed to a *BroadcastReceiver*, code in the *BroadcastReceiver* will be executed.

2.3 Communication between applications

Not only Android components but also Android applications can communicate with each other by *Intents*. For example, *Contacts* and *Phone* are usually two different applications on Android.
Contacts provides contacts management functionality while Phone provides voice calling functionality. In Contacts, we can select a specific contact, and call him or her. After we press the “Call” button in Contacts, an Intent containing the phone number is passed to Phone so that Phone knows what number to dial.

Communication between applications is implemented as Inter-Process Communication (IPC). As mentioned above, developers must sign their applications before releasing them to applications markets and applications with different signatures run in different Linux processes. IPC on Android is implemented as Figure 2-2 [1]. As Figure 2-2 depicts, system applications and user applications are running in different processes because their signatures are different. The application process has limited access to resources on Android. It can only access its own directory. If an application requests more resources, such as locations, it must communicate with system processes. Application processes communicate with system processes by Binder. Binder communication is implemented as a Remote Procedure Call (RPC). In system processes, different system services threads are always waiting for RPCs from applications processes. Before system services return any resources to user applications by RPCs, they will check if applications have permission to access it. Android permission check occurs in system process, it is almost impossible to bypass it by techniques such as Java Reflection [24].

![Figure 2-2 Android Inter-Process Communication](image)
2.4 Android permissions

Android permission system is based on Linux access control. Android regards different applications as different Linux users. Each application is assigned to a UID (user id). Each Android resource is attached to a GID (group id). An application has permission to access some resource if and only if the UID of the application is a member of the GID of the resource.

Android uses install-time control to manage permissions. While Linux access control is DAC (Discretionary Access Control), on Android, the install-time permission control is MAC (Mandatory Access Control). Applications declare all the permissions they want in a manifest file (AndroidManifest.xml). Users cannot install an application until they approve all the requested permissions at install-time. Once these permissions are approved at install-time, they will never be revoked unless the application is removed. Applications cannot request any more permissions after install-time. Developers can add new permissions to an application. But on users’ phones, the application cannot be updated until users approve new added permissions. At runtime, if an application tries to access some resource it does not have permission to, it will be shut down immediately.
Chapter 3

Related work

Researchers have proposed many frameworks for improving Android security since Android went to market. Besides, there have been already many parental control applications in Android app markets. In this chapter, we will walk through some of these frameworks and investigate current parental control status on Android.

3.1 Android security improvement

*Kirin* [4] [5] is a framework protecting Android phones from malicious applications. Current Android permissions system helps protect resources, but it does not define what a secure application is. With *Kirin*, at install-time, it would analyze all the permissions requested by the application and try to identify potential threats. *Kirin* architecture is shown in Figure 3-1. An application cannot be installed unless *Kirin Security Service* approves it.

Figure 3-1 Kirin architecture
Saint is a framework protecting one application from other applications’ attacks [3]. In addition to install-time enforcement, it provides runtime enforcement. At runtime, Saint monitors communication among components (*Activity*, *Service*, *BroadcastReceiver*, *ContentProvider*). Instead of performing application-level access control, Saint achieves component-level access control.

*CRePE* [9] allows end-users to define fine-grained context aware policies. *CRePE* not only enforces permissions check at runtime but also allows trusted third parties to define and activate policies at runtime. Decisions making in *CRePE* depends on the contexts of the phone. Features of *CRePE* are very close to those of parental control.

*TaintDroid* tracks and identifies smartphone privacy risks created by downloaded applications [2]. *TaintDroid architecture* is shown in Figure 3-2. *TaintDroid* dynamically tracks and analyzes privacy sensitive information from their sources such as GPS hardware, microphone and phone identifier storage to the point at which it leaves the phone through a wireless network interface.

Figure 3-2 TaintDroid architecture
3.2 Parental control on Android

(1) Profiles control

Before Android 4.2, there was no any user profiles control mechanism on Android. Children have the same permissions as their parents have on Android devices. Parents usually sign in the phone with their accounts. When children have a chance to use the phone, children can do anything with their parents’ accounts. For example, children may purchase applications and games. Even worse, children may be at risk of online predators and adult contents.

With Android 4.2, users are able to set up individual accounts just like in a PC, but it is only for tablets. With different accounts, parents and children can maintain two different worlds on the same Android tablet.

With Android 4.3, the restricted profiles feature is enhanced. On Android 4.3, parents’ accounts are just like an admin account on Windows. Parents can create a restricted profile for their children. Parents can define applications allowed to use by the restricted profile. Parents can also restrict applications’ access to resources such as locations for restricted profiles. If the user signs in the Android device with a restricted profile, he or she will not be able to access any disallowed applications. Again, it is only for tablets.

Profiles control helps parents maintain a safer digital world for their children, because children’s access to applications can be restricted by their parents. One possible reason why the restricted profiles feature is only available on tablets is that we often share tablets with family members but we seldom share phones with them.
There are many parental control applications in Android application markets. If we use “parental control” as keywords to search in Google Play Store [21], we can find many parental control related applications. After investigating some of them, we notice that functionality and design principles of these parental control applications are very similar.

Let us take Kids Place as an example [12]. Kids Place is the first application popped up when we search “parental control” in Google Play Store and it has been downloaded more than 500,000 times. In customer reviews, it gets four and a half stars, which is a very high score. Generally speaking, what Kids Place and other parental control applications do is to create a new home screen to replace the original one. When we press the home button, it returns to the new home screen, on which we can only do limited things. The new home screen only displays applications that parents allow children to use. Parents can add or remove applications on the new home screen or even change it back to the original home screen by entering the correct pin while children can only access applications allowed by their parents unless they figure out the pin. Even though these parental control applications limit children’s access to specific applications on Android phones, they do not restrict applications’ access to resources. Children are still at risk of privacy disclosure. Moreover, it is easy to attack. Normal phone users, even children can attack it and bypass parental control rules. We will talk about attacks to parental control applications in detail in Section 4.3.
Chapter 4

SPC design

In this chapter we present SPC design. In Section 4.1, we first introduce design goals of SPC. In Section 4.2, we present usage scenarios for SPC. In Section 4.3, we present our attack model. In Section 4.4 we walk through possible design options for SPC and explain why we cannot implement a good parental control application in the application layer. In Section 4.5 and Section 4.6, we present SPC model and design.

4.1 Design goals

From parents’ perspective, there are three major concerns about smartphones in their children’s pockets:

1) Children’s privacy can be exposed to malicious applications or people.
2) Children’s unlimited access to smartphones. It distracts children.
3) Contents on smartphones are inappropriate to children.

A good parental control system should help parents get rid of these concerns. However, it is not wise to totally disable all the potentially dangerous features on children’s phones, such as network and location services, because without these features, children may lose educational and entertaining opportunities too. Smartphones are not “smart” any more. Therefore, we design SPC for helping parents eliminate first two concerns meanwhile keeping the maximum functionality of children’s smartphones. Contents control on smartphones is orthogonal to the first two concerns. Even though SPC could provide it in some sense, it is not a major goal for designing SPC.
4.2 Usage scenarios

*SPC* is designed for scenarios that children and parents own different smartphones. Children always carry their smartphones and parents cannot be with their children all the time. Therefore, in order to provide protection for children, *SPC* is not only deployed on children’s phones but also parents’ phones. With *SPC*, parents can generate parental control rules on their own phones, and push them to their children’s phones over the air.

One possible usage scenario is shown in Figure 4-1. As mentioned, privacy issues depend on contexts. In Figure 4-1, school and home are safe environments, in which it is less risky for third-party applications to access children’s locations. However, if children are on their way to school, it is not as safe as at home or school. In these potential dangerous zones, if children’s locations are disclosed to any malicious applications, online predators may take the chance and locate children. With *SPC*, parents can generate parental control rules for their children’s phones so that children’s phones can adjust access control policy according to current contexts. Hence, with *SPC*
rules enforced on children’s phones, parents can let their children carry smartphones without much concern about privacy disclosure.

Another possible usage scenario is about limiting children’s access to smartphones. For example, parents usually disallow their children to use smartphones after 11:00 pm since it is time for children go to bed. However, children can violate this rule without parents’ awareness and sleep much later than parents’ expectation. With SPC, parents can generate a rule which disallows all the entertaining applications to run after 11:00 pm and push it to children’s phones. Before 11:00 pm, children can use the phone normally but after 11:00 pm, children cannot access those entertaining applications. With SPC, smartphones are much less likely for causing children to fall asleep late.

4.3 Attack model

In this section, we talk about our attack model. There are two major attacks: privacy disclosure and rules bypassing.

A) Privacy disclosure

In dangerous environments, a usually safe operation may become a threat to children. For example, when children are at home, it is safe for them to take a photo and immediately share it on social network, such as Facebook. However, if children are on the way home, after children post photos on Facebook, online predators may figure out children’s locations by these photos. Such privacy disclosure can happen even without any malicious applications installed on children’s phones. Hence, no matter how careful children and parents are, children’s smartphones may still accidently expose their privacy online just like the Facebook photo uploading example.
B) Bypassing parental control rules

Parents usually want to generate parental control rules to restrict children’s access to smartphones. Children should have no chance to bypass these rules. However, since behaviors of children on smartphones are unpredictable and complicated, sometimes some of these rules may be bypassed by children intentionally or unintentionally. Common attacks are rebooting the phone and forcing parental control related services to shut down.

4.4 Design options

In this section, we walk through and evaluate design options for SPC on Android and explain why we cannot implement a good parental control application in current Android application layer. Then we chose the one in the best interest of children’s safety to implement SPC.

(1) SPC as a parental control application

Android is known for its openness, that is, open source and open APIs. But can we really design a good parental control application in application layer on this open system? In this section, we will answer this question.

A parental control application is like a resource manager and a system monitor on Android. Third-party applications cannot access some disallowed resources unless the parental control application approves it. The parental control application should also keep monitoring the current system. Once it detects any anomaly events e.g., a disallowed application is running, it will take actions. In order to realize these features, the parental control application itself needs a lot of permissions, some of which are dangerous, such as preventing other applications from launching. However, these permissions are not provided in Android because they are too risky. Moreover, third-party applications with such features usually are considered as malicious applications. Instead of preventing other applications from launching, existing parental control applications,
such as *Kid Place*, use an alternative. These parental control applications create a launcher (home screen) to replace the original one. In the background, a *Service* is running to monitor the foreground application. If the foreground application is not allowed to run according to the policy of the parental control application, the *Service* just replaces it with the home screen. Therefore, parental control applications such as *Kids Place* cannot prevent you from launching any applications indeed. It just brings the launcher to the top of the screen every time its *Service* in the background detects current foreground application (*Activity*) is not allowed to run. This mechanism appears to work. However before the foreground application being replaced, the disallowed application still can run for a small time window, during which privacy disclosure may happen. Hence, it suffers from attacks. Applications can take advantage of the small time gap to attack parental control applications. They can immediately expose children’s privacy online right after their launching. In addition, the parental control *Service* only checks what the foreground application is. It usually ignores other *Services* in the background, which may also cause parental control issues. For example, the disallowed application can immediately start a music playing *Service* before the parental control application replaces it. After the foreground activity is swapped to the background by the parental control application, the music playing *Service* can continue to play music in the background. Because of it, children can also bypass parental control rules. For example, parents may not allow children to listen to music on the phone after 11:00pm. However, if children start to listen to music at 10:59pm, the music playing application will launch and run normally. After 11:00pm, while parental control applications can interrupt the foreground application (*activity*), *Services* such as the playing music *Service* can keep running in the background. Children can listen to music as late as they want so that they may sleep much later than their parents’ expectation. Another defect with existing parental control applications is that they cannot manage applications’ access to resources even though they can manage children’s access to applications in some sense. Therefore, with existing parental control applications installed, children are still at the risk of privacy disclosure and distraction even without malicious applications on their phones.

According to our discussion above, the answer to the question whether we can design a good parental application on current Android system is no, because it’s too risky for Android to provide support for some features of parental control applications.
(2) Restricted profiles on Android phones

Restricted profiles feature is now only available on Android tablets. The idea of restricted profiles can be applied to Android phones. Restricted profiles can solve many problems in parental control. Children’s access to applications can be restricted. Children can only access to applications allowed by their parents. Restricted profiles feature also provides protection for resources on the phones. For example, applications cannot access to children’s locations unless parents specifically enable it when setting up their children’s profiles. However, restricted profiles lacks of context awareness. Besides, parents do not have control remotely on their children’s phones.

(3) Integrating a parental control framework into current Android

The last design option we will talk about is to extend the Android source code so that it can support all the parental control features we need. In order to manage applications’ access to resources, we can put an additional check on resources parents want to manage. In order to manage children’s access to applications, we can another check where an Android component is launching. Code changes are made in Android middleware layer. It does not require any change to existing applications. The disadvantage of this option is that existing Android phones cannot use this feature unless users replace its phone’s ROM with ours. Since this option does not require any code change to existing applications and with the new added framework, both users’ and applications’ access can be managed, we take this option and SPC will be integrated into Android middleware layer.

4.5 SPC model

In this section, we present SPC model and definitions. SPC model supports fine-grained and context-based control over resources on Android.
4.4.1 Definitions

We extend definitions in GRePE [9].

Definition 1. Rule. A rule \( R = (r, \text{<level>}, \text{<scope>}, \text{<contexts>}, \text{<access>}, \text{<generation date>}, \text{<expiration date>}) \) describes control policy on resource \( r \), where \( \text{<access>} ::= \text{allowed} \mid \text{denied} \); \( \text{<level>} ::= \text{app} \mid \text{data. app} \) means if this rule is active, applications controlled by this rule cannot run, while \text{data} means applications controlled by this rule can still run, but cannot get valid resources; \( \text{<scope>} \) specifies applications controlled by this rule; \( r \) can be any resources on the phone, such as camera and locations. If \( r \) is \text{N/A}, it means applications controlled by this rule cannot launch once this rule is active; \( \text{<generation date>} \) is the time when \( R \) is generated. For example, if parents do not allow children to play games on the phone after 10:00pm, the rule is should be similar to this \( R = (\text{<N/A>}, \text{<app>}, \text{<candy rush, temple runs>}, \text{<after 10:00pm>}, \text{<denied>}, \text{<20140120.102030>}, \text{<forever>} \). Table 4-1 shows all the fields and their descriptions of SPC rules.

Definition 2. Context. A context can be defined by the status of some variables such as locations, temperature, the presence of other devices, a particular interaction between the user and the smartphone, or a combination of these. One rule is associated with one context.

Definition 3. Active Context. A context \( c \) is said to be active, at a given time \( t \), if the circumstances that the context specifies are verified. More than one contexts can be active at the same time.

Definition 4. Active Rule. Given a rule \( r \) with context \( c \), \( r \) is an active rule only if \( c \) is an active context.

Definition 5. Conflict. Two rules are in conflict if they are defined on the same resource \( r \) and their contexts have intersections and they have share at least one application in \text{scope} field, but with opposite access.
\[ R_1 = (\langle \text{resource\_set1}, \langle \text{level\_1}, \langle \text{appset\_1}, \langle \text{context\_set1}, \langle \text{access1}, \langle \text{generation\_date1}, \langle \text{expiration\_date1}\rangle \rangle \rangle \rangle \rangle \rangle \]\n
\[ R_2 = (\langle \text{resource\_set2}, \langle \text{level\_2}, \langle \text{appset\_2}, \langle \text{context\_set2}, \langle \text{access2}, \langle \text{generation\_date2}, \langle \text{expiration\_date2}\rangle \rangle \rangle \rangle \rangle \rangle \]\n
\[ \text{is\_conflict} = (\text{access1} \oplus \text{access2}) \land \neg ((\text{appset\_1} \cap \text{appset\_2}) = \emptyset) \land \neg ((\text{resource\_set1} \cap \text{resource\_set2}) = \emptyset) \land \neg ((\text{context\_set1} \cap \text{context\_set2}) = \emptyset) \]

\( R_1 \) and \( R_2 \) are in conflict if and only if \( \text{is\_conflict} \) is true.

E.g.,

\[ R_1 = (\langle \text{camera}, \langle \text{app}, \langle \text{FunnyFaces, GoodPhoto}, \langle \text{not at home}, \langle \text{denied}, \langle 20140102,112200\rangle, \langle \text{forever}\rangle \rangle \rangle \rangle \rangle \rangle \]

\[ R_2 = (\langle \text{camera}, \langle \text{app}, \langle \text{FunnyFaces}, \langle \text{not at home}, \langle \text{allowed}, \langle 20140102,112300\rangle, \langle \text{forever}\rangle \rangle \rangle \rangle \rangle \]

\( R_1 \) and \( R_2 \) both define access control rules for \text{camera} but with opposite access. Besides, \( R_1 \) and \( R_2 \) share the same contexts value “not at home” and “FunnyFaces” is controlled by both rules, so there is a conflict between \( R_1 \) and \( R_2 \).

**Definition 6. Policy.** A policy \( P \) is a set of rules.
### 4.4.2 Conflicts resolving strategy

We need to resolve conflicts among rules if conflicts happen. Suppose R1 and R2 are two rules and they are conflicted. R2 is the more recently generated rule (\textit{generation\_date2} is more recent than \textit{generation\_date1}).

\[
R1 = (<resource\_set1>, <level\_1>, <appset\_1>, <context\_set1>, <access1>, <generation\_date1>, <expiration\_date1>)
\]

\[
R2 = (<resource\_set2>, <level\_2>, <appset\_2>, <context\_set2>, <access2>, <generation\_date2>, <expiration\_date2>)
\]
When conflicts happen, we resolve conflicts by using the following steps:

(1) Get applications controlled by both R1 and R2.

\[
\text{overlapped_applications} = \text{appset}_1 \cap \text{appset}_2
\]

(2) Update R1’s scope as follow:

\[
\text{appset}_1 = \text{appset}_1 - \text{overlapped_applications}
\]

(3) If appset_1 is not empty, write R1 back with updated scope. If appset_1 is empty, eliminate R1 from the policy set.

(4) Add R2 to the policy set.

4.4.3 SPC v.s CRePE

SPC extends most of the definitions from CRePE but they differ in some aspects. In CRePE, every policy is associated with a context; while in SPC, we associate a rule with a context. The context of a policy in SPC is the union of contexts of rules in the policy. It is difficult for parents to generate all the rules at a time for their children’s phones, especially when there are many context status variables to consider. With SPC, parents can generate one rule at a time and the policy can be incrementally refined. In addition, in SPC, we can specify level option in a rule so that it can provide a fine-grained and flexible control over resources on children’s phones.

4.6 Architecture

SPC is designed as a modification to Android middleware. For each of resources SPC manages, before the resource returned to the application, SPC should check if applications can access it according to current policy and contexts (Figure 4-2). We define such check as resource check. In addition, SPC should enforce a check where an activity or service is launching. (Figure 4-3). We define such check as launching check. For each resource managed by SPC, resource check is put
before the resource returned to applications. *Resource check* is after Android permission check. If a request to some resource is denied by Android permission check, it will not reach the *SPC* permission check. *Launching check* is put where an Android component is launching. One of the defects of existing parental control applications have is that they are not able to totally prevent *Activities* and *Services* of other applications from launching. *Launching check* can make up for it. Activities and Services cannot launch unless it can pass *Launching check*. Another important feature of *SPC* is contexts awareness. *SPC* makes parental control decision based on current contexts. In order to get current contexts, *SPC* should be able to monitor and collect contexts information.

![Figure 4-2 Resource check](image)

**Figure 4-2 Resource check**

![Figure 4-3 Launching check](image)

**Figure 4-3 Launching check**

It is desirable that we could develop all the *SPC* features without any modification into Android middleware layer. However, as we discussed above, current Android do not support some of parental control features we need. In order to achieve context-based and fine-grained control over resources, modification to Android middleware layer is necessary.
Figure 4-4 shows the architecture of SPC. As mentioned, SPC is not only on children’s phones, but also on parents’ phones. They communicate with each other by network. Parents can generate parental control rules on their own phones and push them over the air to their children’s phones. We will introduce each component in Figure 4-4 in the following sections.

(1) **SPC on parents’ phones**

SPC on parents’ phones is an ordinary Android application running in the application layer. It does not require any special support from Android. Parents can use it to generate parental control rules and push them to their children’s phones.

(2) **SPC on children’s phones**

SPC on children’s phones is more like a security framework while SPC on parents’ phones is more like a normal application. On children’s phones, in order to achieve what the SPC model
describes, modification to Android middleware is necessary. Before we dig into details of SPC architecture, let us first look at general functionality of SPC on children’s phones and what change needs to be made.

Figure 4-4 presents the architecture of SPC on children’s phones. We will explain each module in detail and use an example to illustrate how SPC on children’s phone works.

A. SPC Interface

SPC Interface is responsible for communicating with the outside of the phone. SPC Interface is running as a Service in Android application layer. Children have no access to SPC Interface. SPC Interface transfers messages between parents’ phones and Android middleware of children’s phones. It is important to ensure messages transferred into Android Middleware are not malicious. Therefore, SPC Interface signs all the messages it sends and receivers of these messages should verify the signature. Once SPC Interface receives verified and valid parental control rules from parents’ phones, after simple preprocessed, these rules will be pushed to Android middleware, in which Policy Provider will receive these rules.

B. Policy Provider

Policy Provider manages SPC rules. Rules in Policy Provider can only be updated by SPC Interface. Policy Provider only accepts messages signed by SPC Interface. Policy Provider maintains the SPC policy. All the rules are stored in a ContentProvider (one of the Android components). The ContentProvider can only be accessed by Policy Provider.
C. Conflicts Resolver

Conflicts Resolver is a part of Policy Provider. The functionality of Conflicts Resolver is self-evident. When Policy Provider adds a new rule to the policy, conflicts may happen. Conflicts Resolver is responsible for detecting conflicts and resolving it by the strategy we provide in Section 4.4.2.

D. SPC Decider

SPC Decider makes parental control decisions. When requests reach resource check or launch check, SPC Decider will pull policy set from Policy Provider and current contexts from SPC Monitor Service. SPC Decider can generate active rules based on the policy set and current contexts. With current active rules, SPC Decider can make decisions on whether or not to grant access.

E. SPC Monitor Service

SPC Monitor Service is designed for monitoring and collecting contexts. SPC Monitor Service is an Android Service running in the background. Before SPC Decider makes a decision, it will ask this service for current contexts. According to Android Service life cycle, when the memory is critically low, it may kill background Services. So there is a chance that SPC Monitor Service gets killed. Even though SPC Monitor Service can restart shortly after being killed, we should handle situations without SPC Monitor Service. SPC Monitor Service also monitors current system status. Once it detects any anomaly events, such as a disallowed application is running, it will take an action and stop it.
F. An example

We now use an example to illustrate how SPC on children’s phones works. Suppose when children are not at home, their parents do not allow any applications to access the camera except the original camera application on children’s phones. Parents can generate a SPC rule like this:

\[ R = (\text{<camera>}, \text{<app>}, \text{<all the application except the original camera app>}, \text{<not at home>}, \text{<denied>}, \text{<2014-01-05-100500>}, \text{<forever>}) \]

Parents can push R to their children’s phones over the air. On children’s phones, SPC Interface will receive R. SPC Interface then sends it to Android middleware and Policy Provider will handle it. Policy Provider adds R into the policy. Conflicts Resolver finds out currently there are no rules controlling camera and there are no conflicts. Then Policy Provider commits this new rule and the SPC policy is updated. When children are on their way to school, if children try to use third-party applications to take a photo, the request to use the camera will reach resource check before the camera is returned to the application. When resource check is triggered, SPC Decider will pull rules from Policy Provider and contexts from SPC Monitor Service, according to which active rules can be generated. In this case, R is active since children are not at home, and the access to camera must be denied. Since the level is app, SPC Decider denies the access and force the application to quit.
Chapter 5

SPC implementation

In this chapter, we introduce our SPC implementation on Android. We implement SPC as a modification to Android 4.3. In our implementation, we take two major contexts, location and time, into consideration. Parents can generate time-based or location-based rules on their phones and push them to their children’s phones over the air. SPC implementation consists of three parts. 1) On parents’ phones, there is an application for parents to generate rules. 2) On children’s phones, in order to support fine-grained and context-based access control policies, we modify the source code of Android and integrate SPC module to it. 3) SPC server. SPC server maintains the relationship between parents and children. All the messages transferred between parents and children are authenticated by SPC server.

5.1 SPC server

We implement SPC server on GAE (Google App Engine). Google App Engine is a Platform as a Service (PaaS) offering to let one build and run applications on Google’s infrastructure. In order to make SPC easy for parents to use, we take advantage of the Google account on Android. Android users usually sign in their devices with their Google accounts so that they can enjoy popular Google services, such as Gmail, Google Hangout etc. SPC server is built on GAE. As a part of Google, GAE can use Google account credentials and Google cookies to authenticate requests from Android phones. SPC server maintains relationship between parents and children. As long as the phone can be authenticated by Google, it can be authenticated by SPC server too. All the messages transferred between parents and children must go through the server and be authenticated (Figure 5-1).
5.2 Implementation of SPC on parents’ phones

SPC on parents’ phones is an ordinary application providing user interface for parents to generate SPC rules. Generated rules can be pushed to their children’s phones over the air. SPC can manage access to locations and capability to launch an Activity or Service.

(1) Location-based rules

Location is one of the most important sensitive data on smartphones, especially for children. Besides, location is also an important context variable for smartphones to make access decisions. Parents are usually concerned more about dangerous regions than safe ones for their children. In SPC, parents can mark potential unsafe regions on Google Map. Once their children’s phones are in these regions (Figure 5-1), SPC Monitor Service can detect it and activate related rules defined by parents. After that, parents can select their children’s accounts (Figure 5-2). Parent-children relationship is maintained by SPC server. Then parents will receive applications list on their children’s phone (Figure 5-3). Parents can specify what applications will be affected by the newly generated rules. A sample generated location based rule is,

\[ R = (\text{<GPS location>}, \text{<data>}, \text{<Google Maps>}, \text{<Anywhere between home and school>}, \text{<denied>}, \text{<2014-01-05-100500>}, \text{<forever}> ) \]
This rule means if children are on anywhere between home and school, *Google Maps* can still run but cannot access GPS locations.

![Figure 5-2 Mark unsafe regions for children](image1)

![Figure 5-3 Select children's accounts](image2)
(2) Time-based rules

Time is another important context for children. On the one hand, children’s safety depends on time. It is usually more dangerous to disclose children’s privacy in the evening than in the daytime. Therefore, to protect children, applications’ access to resources should depend on time. On the other hand, children’s access to the phone also depends on time. Parents usually do not allow their children play games on smartphones at midnight. With SPC, parents can select a specific time interval, and set affected applications when generating location based rules (Figure 5-4).
A sample time based rule is like this,

\[ R = (\langle all \rangle, \langle app \rangle, \langle all \rangle, \langle between 10:30pm and 8:00am \rangle, \langle denied \rangle, \langle 2014-01-05-110500 \rangle, \langle forever \rangle) \]

Rule \( R \) means children cannot open any applications between 10:30pm to 8:00am.

**5.3 Implementation of SPC on children’s phones**

As mentioned, *SPC* on Children’s phones is implemented as a modification to Android middleware. In this section, we introduce some implementation details, tricks and issues.
5.3.1 App level and data level access control

It is easy to implement application-level access control in Android middleware layer. We can force the application to quit if it tries to access some sensitive resource. When applications call Android API, usually the Context (the parent’s Class of Activity and Service) object of the application will pass into Android middleware layer. If SPC permission check fails, we can force applications to quit by calling Context.finish(). SPC also supports fine-grained data access control. In SPC rules, we can specify <level> option where level ::= < data | app >. If level is data, when an application tries to access resources it is not allowed to, instead of preventing the application from running, SPC allows the application to run but feed it with mocked data. In our implementation, if an application without SPC location permission tries to access locations on the phone, SPC will generate a random location and return it to the application. It is easy to mock data such as locations but it is not easy to mock resources like camera. The simplest way is to return Null when SPC resource check fails. However, many applications cannot handle Null where an object is expected and they may crash by receiving Null. When crash happens, fine-grained data access control degrades to application-level access control. In order to avoid the degradation, our suggestion is that when SPC resource check fails, SPC should return a mocked object rather than Null. We can generate the mocked object by extending the original class. In the mocked object, we can disable potential dangerous features by overriding.

We use an example to illustrate this mock technique. Suppose we want to manage access to camera. Applications can access camera and take a picture by using the following code.

```java
Camera camera = Camera.open()
camera.takePicture()
```

In order to manage access to camera, we can put a SPC resource check where the camera object is returned. In /frameworks/base/core/java/android/hardware/Camera.java, there are two methods to return a camera object. One is public static Camera open(int camerald); the other is public static Camera open( ). The original open(int camerald) method is like this,
public static Camera open( int cameraId){
    return new Camera(cameraId)
}

The modified open(int cameraId) method is like this,

public static Camera open( int cameraId){
    if (!SPCDecider.resource_check('camera'))
        return new MockCamera();
    return new Camera(cameraId)
}

MockedCamera can be implemented like this,

public class MockedCamera extends Camera{
    @Override
    public void takePicture(ShutterCallback shutter, PictureCallback raw,
        PictureCallback postview, PictureCallback jpeg){
        }
}

If an application cannot pass the resource check, it will get a MockedCamera object instead of a camera object. When the application calls takePicture, nothing will happen. We can apply the same technique to public static Camera open( ) method. In addition, applications can call Android built-in Activity to access camera and take photos. SPC launching check will be triggered before the built-in camera Activity launches. If the application is not allowed to access camera, SPC launching check can deny the access and open a mocked Activity instead of the camera Activity.
Fine-grained data access control is important for parental control because it eliminates potential threats meanwhile maintaining the basic functionality. For example, a map application may share users’ locations online. When children are in an unsafe environment, this location sharing feature becomes a threat. If we disallow the application to launch, children lose the chance to check the map as well. With SPC, if rules are set correctly, children can use the application, but when the application tries to read locations, it will get mocked location data so that children’s locations cannot be disclosed.

5.3.2 Apps list on children’s phones

When parents are generating SPC rules, parents’ phones should be able to get the most updated applications list on their children’s phones. On children’s phones, it is easy to read applications list and upload it to SPC server so that parents’ phones can also have access to it. A naive implementation is that children’s phones periodically read the applications list and send it to the server. The problem here is the frequency to update SPC server. If the frequency is low, children may install some malicious applications without parents’ awareness. For example, if children purchase an adult application in the Google Play Store, parents should be notified as soon as possible. Even though the feature of notifying parents with new installed applications is not implemented in current SPC, it is definitely necessary for SPC to support it. If the frequency is high, system performance will be reduced, since reading applications list from API involves I/O operations and it also introduces unnecessary network traffic, which is battery-consuming.

In our implementation, except for the very first time to send applications’ list to SPC server, SPC on children’s phones updates it every time it receives Application Install Broadcast Event or Application Uninstall Broadcast Event. These events are broadcasted by Android when installing or uninstalling an application.
5.3.3 Interrupt current running application

*SPC check* can be performed before launching an Android application or Android *component*. However, let us consider this situation. Parents define a rule that children cannot open any game applications after 10:30 pm because it is time for children to go to bed. If children open a game application at 10:29 pm, it would pass the *SPC* check. As long as children do not quit the game, he can use the app as late as he wants. In order to prevent this from happening, other than performing permissions check before launching applications, *SPC* should also be able to monitor the foreground application and background services. In our implementation, *SPC Monitor Service* does the job for us. *SPC Monitor Service* is running in the background. In addition to monitor current contexts and collect contexts information, it monitors current running *Activities* and *Services*. Once *SPC Monitor Service* notices that a rule can be activated by current contexts, it will check whether the foreground application and background services are allowed by current policy. If not, *SPC Monitor Service* forces it to quit.

5.3.4 *SPC Services can be killed*

When Android is running out of memory, it will kill *Services* in the background to release resource. Phone users can also kill *Services* by using applications like task managers. There are two important *Services* in *SPC*. One is *SPC Interface*; the other is *SPC Monitor Service*. Even if the chance of *SPC’s Services* to be killed is low, it still can happen. After *SPC Monitor Service* is killed, it will restart when there is enough resource to support it to run. If *SPC Monitor Service* is killed for some reason and it has not restarted yet, *SPC check* cannot pull contexts information from *SPC Monitor Service*. We adopt a conservative way to handle situations where *SPC Monitor Service* is killed. All the rules are considered to be active if *SPC Monitor Service* is not running so that children’s access to applications and applications’ access to resource will be limited in this situation but it is in the best interest of children’s safety. If *SPC Interface* is killed, children’s phones cannot communicate with *SPC server* and parents’ phones. Children’s phones cannot receive any new rules pushed by their parents without *SPC Interfaces*. As *SPC Monitor Service*, *SPC Interface* will be restarted later if it is killed. *SPC server* stores rules generated by parents’ phones. When *SPC Interface* is restarted, it can pull rules generated during its shutdown.
5.3.5 SPC extension

There are different kinds of resources on Android and there are a huge number of potential contexts to consider such as speed and temperature. In our implementation, SPC manages children’s access to applications and applications’ access to locations based on current time and location. In the future, we may want to consider more context variables. For instance, parents want to generate a rule that disables all the applications’ access to camera if the current moving speed is more than 30 miles per hour. In order to achieve this, on parents’ phones, we should create an interface for parents to generate the rule. After the change to code of SPC on parents’ phones is done, parents can update it from Google Play Store or other applications markets so that they can generate the new rule. On children’s phones, things are more complicated. Currently SPC Monitor Service does not collect speed information, so we must add it. Besides, we must put a SPC permission check on camera to manage applications’ access to it. All the changes are made in Android middleware layer. SPC on children’s phones cannot be extended without updating Android system. Hence, if we want to take more contexts variables into consideration, SPC Monitor Service has to be modified to collect contexts information; if we want to manage more resources on Android, a SPC permission check has to be put on the new resource.
Chapter 6
Evaluation

In this chapter, we will evaluate SPC performance. On parents' phones, SPC helps parents generate context-based rules. SPC is an ordinary application rather than a security module on parents' phones. Hence, in this section, we only evaluate the performance of SPC on children's phones. We evaluate it from two aspects, power consumption and impact on applications.

6.1 Power consumption

In this section, we investigate the effect of SPC Monitor Service on battery usage. In order to monitor and collect current contexts, SPC Monitor Service is always running in the background as an Android Service. Network communication with parents’ phones and SPC server also increases battery usage on children's phones. However, battery usage introduced by network on children's phones largely depends on parents' behaviors. If parents frequently push parental control rules to their children’s phones, power consumption on children's phones will increase since more network operations are involved. In our evaluation, we rule out the effect of parents' behaviors and only consider power consumption introduced by SPC Monitor Service.

In our experiments, the phone we use is a Nexus 4 phone installed with modified Android 4.3, which is integrated with SPC. The test phone has no SIM cards. In our implementation, SPC considers two types of contexts, time and location. In order to activate SPC rules automatically, SPC Monitor Service check current time and location every minute. Before each experiment, we fully charge the phone and set the time to turn off the screen to 30 minutes. In each experiment, we turn on the screen and set up all the experiment parameters. When the screen turns off automatically, we record current battery percentage. For each set of parameters, we repeat the experiment for 10 times.
Table 6-1 SPC battery usage

<table>
<thead>
<tr>
<th>SPC STATUS</th>
<th>RUNNING APPLICATION</th>
<th>LAST TIME</th>
<th>AVG BATTERY USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>None</td>
<td>30 mins</td>
<td>7.9%</td>
</tr>
<tr>
<td>On</td>
<td>None</td>
<td>30 mins</td>
<td>8.2%</td>
</tr>
<tr>
<td>Off</td>
<td>A test application that keeps reading GPS location</td>
<td>30 mins</td>
<td>10.2%</td>
</tr>
<tr>
<td>On</td>
<td>A test application that keeps reading GPS location</td>
<td>30 mins</td>
<td>10.2%</td>
</tr>
</tbody>
</table>

Table 6-1 shows experimental parameters and results. In the first two experiments, we do not run any applications. The phone stays on the home screen. According to the result, the difference of power consumptions between phones with and without SPC is very small. SPC Monitor Service increases 3.8 percent $((8.2 - 7.9)/7.9)$ battery usage. Therefore, in our implementation, even though SPC Monitor Service is always running in the background, it does not consume too much power. However, we have to point out that in our current SPC Monitor Service implementation, it is only responsible for two contexts, location and time. As more and more contexts are involved, such as speed and temperature (normally there is no thermometer on smartphones, but we can get it by calling network services), battery usage of SPC would be higher than the value we got in the experiment. In the last two experiments, we run a test application, which keeps reading GPS locations. In this case, no matter SPC Monitor Service is on or off, every time when the test application is trying to access GPS locations, it has to pass SPC Permission Check. When SPC Monitor Service is turned off, SPC Permission Check just supposes all the rules are active. According to our experiment results, the difference of power consumption between phones with
and without SPC enabled is little. Hence, *SPC Permission Check* makes little effect on battery usage.

6.2 Impact on applications

In this section, we investigate impact of *SPC* on functionality of applications. *SPC* provides fine-grained data access control at runtime. When an application tries to access resources it is not allowed to, instead of forcing the application to quit, *SPC* can allow it to run, but feed it with mocked resources. It is not clear how much impact such mocked resource returned by *SPC* can make on basic functionality of applications. If the mocked resource severely impacts applications, the fine-grained data access control would not be helpful and degrade to application-level access control. In this section, we will evaluate *SPC*’s impact on applications.

Table 6-2 Applications in the experiment

<table>
<thead>
<tr>
<th>APPLICATIONS IN THE EXPERIMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Weather Channel</td>
</tr>
<tr>
<td>Facebook</td>
</tr>
<tr>
<td>FruitNinja</td>
</tr>
<tr>
<td>AccuWeather</td>
</tr>
<tr>
<td>JetpackJoyride</td>
</tr>
<tr>
<td>My Current Location</td>
</tr>
<tr>
<td>Zillow</td>
</tr>
</tbody>
</table>

In our experiments, we randomly chose 100 free popular applications requesting locations permission in *Google Play Store*. Some of these applications are listed in Table 6-2. Then we disallow these applications to access locations by pushing *SPC* rules to the phone. According to
our experiments, none of these applications crashes with the mocked location data. As discussed in Section 5.3.1, resources like locations are easy to mock, while resources like camera are not. The impact of the data level access control on applications depends on how we mock the resource.
Chapter 7

Discussion

In this chapter, we introduce SPC attacks and defenses, future work about SPC and parental control on another prevalent mobile operating system iOS.

7.1 SPC attacks and defenses

As mentioned in Section 4.3, there are two major attacks, privacy disclosure and bypassing rules. With SPC, if parents are aware of threats from their children’s smartphones and push correct rules to their phones, children’s privacy can be protected. Another attack is conducted by children. Children may bypass rules set by their parents. SPC is in Android middleware layer, it’s not easy to phone users to bypass its rules. Android system would kill SPC Monitor Service in order to get more resources. There is a small time gap between SPC Monitor Service killed and restarted. During this time gap, SPC adopts a conservative way, which considers all rules are active, so that children’s privacy can also be protected, and children cannot bypass any rules. Another possible attack is that children may logout with the Google account. If it happens, SPC server cannot authenticate children’s phones any more. In order to prevent it from happening, on children’s phones, when we login SPC server with the Google account on children’s phones the first time, SPC server will assign a random key to children’s phones. If children log out later, SPC server can authenticate children’s phones by using the random key.

7.2 Automatically determine contexts sensitivity

Currently, SPC depends on parents to decide contexts sensitivity. It is relatively easy for parents to generate access control policies for contexts such as home, school or school bus route whose locations or trajectory are known in advance. However, in other scenarios, often contexts sensitivity is not so clear and cannot be determined beforehand. For example, if children go to a
summer camp, it’s difficult for parents to generate rules in this situation since parents do not know what will happen in the summer camp. In the future, we may use machine learning methods to get a better idea of contexts sensitivity, and generate rules automatically without parents’ involvement.

7.3 Parental control on iOS

iOS is another prevalent mobile phones OS. Permissions control on iOS is stricter than that on Android. On iOS, it is almost impossible to create a parental control application like on Android. Features of parental control applications usually violate iOS security principles, such as an application can not interrupt another.

However, iOS itself does provide parental control [13], called restriction. Users can easily set up restrictions in iOS settings as shown in Figure 7-1. iOS restriction provides both coarse-grained and fine-grained permissions control. For example, we can disallow a map application to run or we can allow the map application to run, but it is not able to access locations. Users’ access to applications and applications’ access to privacy information such as locations and contacts can be controlled by iOS. Besides, it provides features for content control. Parents can determine what contents on iOS devices children are allowed to watch. For example, we can disallow iOS movie player to play any movies rating lower than PG-13.
Even though iOS is not an open source project and many innovative permissions control ideas are not integrated into it, restriction can still provide a good parental control mechanism for parents and children.
Chapter 8

Conclusion

The technology trend among parents and children presents entertaining and educational opportunities, but it also presents challenges. Smartphones bring threats to children and parental control is necessary on smartphones. In this thesis we present $SPC$ framework. $SPC$ addresses the current limitations of Android on parental control support. $SPC$ is fine-grained and context-based. $SPC$ protects children’s privacy from disclosure. It also restricts children’s access to applications so that smartphones do not distract them.

We are just at the beginning of this work. A most pressing need now is to extend $SPC$ so that it can manage more resources and contexts variables. According to our evaluation, power consumption of $SPC$ is very little. However, as more and more contexts and resources involved, power consumption may increase. We hope in the future, with improvement, $SPC$ can cover more resource and contexts without considerably increasing power consumption.
References


[27] “Target 11 investigates smartphone app predators use”


[32] “Parents worry about what their children are doing with devices they bought them”