A FLEXIBLE AND MORE RESTRICTIVE ACCESSIBILITY PROTOCOL FOR CELLULAR APPLICATIONS

Supriya[1], Girish B G[2]

[1](M.Tech,Student,Computer science engineering, S.J.C.Institute of Technology,Karnataka, India)
[2](Assistant Professor, Computer science engineering, S.J.C.Institute of Technology, Karnataka,India)

ABSTRACT

In Android OS, installing malicious or non-market applications may result in privacy breaches and sensitive data leakage. Also Android OS often have access to sensitive data and resources on user device. By Default, Android users do not have control over the application once the applications have been granted the requested privileges during installation. The threat arises when a device application acts maliciously and it may leaks the user's personal data without the user's consent.In many cases, however, whether an application may get a privilege depends on the specific user context and thus, we need a Context-Based Access Control mechanism by which privileges can be specified by the user. The efficiency of our access control mechanism and the accuracy of context detection device resources to spy on the user or leak the user's personal data without the user's consent.C-BAC (Context- Based Access Control) mechanism is modified in an Android Operating System. Using C-BAC, requested privileges can be dynamically granted or revoked to applications based on the specific context of user. C-BAC policies, differentiates between closely located sub-areas within the same location.

Keywords: *Context-based access control, smartphone devices, security and privacy, policies, mobile Applications*

1.INTRODUCTION

As smartphones are becoming more powerful in terms of computational and communication capabilities, application developers are taking advantage of these capabilities in order to provide new or enhanced services to their applications. For example, on March 2013 Samsung unveiled its Galaxy S4 device with 8 CPU cores and 9 sensors that enrich the device with powerful resources [1]. However, the majority of these resources can collect sensitive data and may expose users to high security and privacy risks if applications use them inappropriately and without the user's knowledge [2]. The threat arises when a device application acts maliciously and uses device resources to spy on the user or leak the user's personal data without the user's consent [3]–[5]. Moreover, users carrying their smartphones in public and private places may unknowingly expose their private information and threaten their personal security as they are not aware of the existence of such malicious activities on their devices.

To prevent such threats, users must be able to have a better control over their device capabilities by reducing certain application privileges while being in sensitive contexts e.g. confidential meetings. To achieve this, smartphone systems must provide device owners with configurable policies that enable users to control their device usage of system resources and application privileges according to context, mainly location and time. Since such a feature is still missing in popular smartphone systems, such as in Android systems, it is crucial to investigate approaches for providing such control to device users.

The need for configurable device policies based on context extends from high profile employees to regular smartphone users. For example, government employers, such as in national labs [6], restrict their employees from bringing any camera-enabled device to the workplace, including smartphones, even though employees might need to have their devices with them at all times as their devices may contain data and services they might need at any time. With context-based device policies, employees may be allowed to use smartphones as they can disable all applications from using the camera and any device resources and privileges that employers restrict while at work, while the user device can retain all its original privileges outside the work area. Context-based policies are also a necessity for politicians and law enforcement agents who would need to disable camera, microphone, and location services from their devices during confidential meetings while retaining these resources back in nonconfidential

locations. With context-based policies, users can specify when and where their applications can access their device data and resources, which reduces the hackers' chances of stealing such data.

The design of context based policy systems for smartphones is challenging as it should fulfill the following requirements:

1) Applications should not be able to fake the location or time of the device, as they should not be able to bypass the policy restrictions applied on the device in a specific context.
2) As users are assumed to be mobile, the policy restrictions should be applied automatically on the device as the device's location changes.
3) The accuracy of location needs to be higher than the location accuracy by GPS, as we need to apply different policies in different spots or nearby sub-areas located within the same GPS location.
4) The enforcement of context-based policies should not require the application developers to modify source code, or impose any additional requirement on their applications.
5) The applied policy should not cause significant delays in the device functionality that could negatively impact the system performance.

In this paper, we propose a context-based access control (CBAC) mechanism for Android systems that allows smartphone users to set configuration policies over their applications' usage of device resources and services at different contexts. Through the CBAC mechanism, users can, for example, set restricted privileges for device applications when using the device at work, and device applications may re-gain their original privileges when the device is used at home. This change in device privileges is automatically applied as soon as the user device matches a pre-defined context of a user-defined policy. The user can also specify a default set of policies to be applied when the user is located in a non-previously defined location. Configured policy restrictions are defined according to the accessible device resources, services, and permissions that are granted to applications at installation time. Such policies define which services are offered by the device and limit the device and user information accessibility. Policy restrictions are linked to context and are configured by the device user. We define context according to location and time. Location is determined basically through visible Wi-Fi access points and their respective signal strength values that allows us to differentiate between nearby sub-areas within the same work space, in addition to GPS and cellular triangulation coordinates whenever available. We implement our CBAC policies on the Android operating system and include a tool that allows users to define physical places such as home or work using the captured Wi-Fi parameters. Users can even be more precise by differentiating between sub-areas within the same location, such as living rooms and bedrooms at home or meeting rooms and offices at work. Once the user configures the device policies that define device and application privileges according to context, the policies will be automatically applied whenever the user is within a pre-defined physical location and time interval.

2.EXISTING SYSTEM

Security for mobile operating systems focuses on restricting applications from accessing sensitive data and resources, but mostly lacks efficient techniques for enforcing those restrictions according to fine-grained contexts that differentiate between closely located subareas[7]. Moreover, most of this work has focused on developing policy systems that do not restrict privileges per application and are only effective system-wide. So User disable all applications from using the camera and any device resources and privileges that employers restrict while at work, while the user device can retain all its original privileges outside the work area.

Disadvantages of existing system:

- Do not cover all the possible ways in which applications can access user data and device resources.
- The User leakage of their privacy.
- Existing location-based policy systems are not accurate enough to differentiate between near by locations without extra hardware or location devices[7],[8],[9].

3.PROPOSED SYSTEM

- In this paper, we propose a context-based access control (CBAC) mechanism for Android systems that allows smart phone users to set configuration policies over their applications' usage of device resources and services at different contexts.
- Through the CBAC mechanism, users can, for example, set restricted privileges for device applications when using the device at work, and device applications may regain their original privileges when the device is used at home. This change in device privileges is automatically applied as soon as the user device matches a pre-defined context of a user-defined policy.

- The user can also specify a default set of policies to be applied when the user is located in a non-previously defined location. Configured policy restrictions are defined according to the accessible device resources, services, and permissions that are granted to applications at installation time. Such policies define which services are offered by the device and limit the device and user information accessibility. Policy restrictions are linked to context and are configured by the device user. We define context according to location and time.
  Advantages of proposed system:
  - Applications should not be able to fake the location or time of the device.
  - Can develop securer and more acceptable applications for end users.

4.ARCHITECTURE DESIGN

In this section, we introduce the design of our architecture through describing the components of our access control framework with the corresponding role of its entities.Our framework consists of an access control mechanism that deals with access, collection,storage, processing, and usage of context information and device policies. To handle all the aforementioned functions, our framework design consists of four main components as shown fig.1

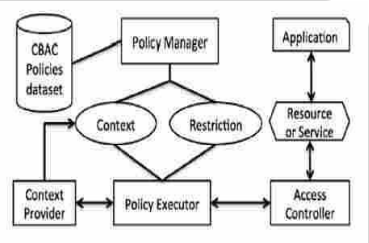

Fig 1: Access control framework.

The *Context Provider (CP)* collects the physical location parameters (GPS, Cell IDs, Wi-Fi parameters) through the device sensors and stores them in its own database, linking each physical location to a user-defined logical location. It also verifies and updates those parameters whenever the device is re-located.
The *Access Controller (AC)* controls the authorizations of applications and prevents unauthorized usage of device resources or services. Even though the Android OS has its own permission control system that checks if an application has privileges to request resources or services, the AC complements this system with more control methods and specific fine-grained control permissions that better reflect the application capabilities and narrow down its accessibility to resources. The AC enhances the security of the device system since the existing Android system has some permissions that, once granted to applications, may give applications more accessibility than they need, which malicious code can take advantage of. For example, the permission READ_PHONE_ STATE gives privileged applications a set of information such as the phone number, the IMEI/MEID identifier, subscriber identification, phone state (busy/available), SIM serial number, etc.
The *Policy Manager (PM)* represents the interface used to create policies, mainly assigning application restrictions to contexts. It mainly gives control to the user to configure which resources and services are accessible by applications at the given context provided by the CP. As an example, the user through the PM can create a policy to enable location services only when the user is at work during weekdays between 8 am and 5 pm.
The *Policy Executor (PE)* enforces device restrictions by comparing the device's context with the configured policies. Once an application requests access to a resource or service, the PE checks the user-configured restrictions set at the PM to either grant to deny access to the application request. The PE acts as a policy enforcement by sending the authorization information to the AC to handle application requests, and is also responsible to resolve policy conflicts and apply the most strict restrictions.

5. IMPLEMENTATION

In this section, we introduce the technical details of our implementation which includes our modifications to the Android OS and the components of the Policy Manager custom application that acts as an intermediary between the OS and the user's desired policy configurations.

5.1 Policy Manager Components

The Policy Manager custom application consists of the four main Android application components: Activities, Broadcast Receivers, a Content Provider, and a Service.

Activities: The user interacts with the Policy Manager via activities, and through these activities, a user is able to define physical locations and subsequently configure a set of policies for these locations. The main constituents of these activities include *Application Events, Permission Access, Resource Access, System Preferences, and Time Restriction*.

Broadcast Receiver: We extended the Android's Broadcast Receiver class and created two custom classes, the Start Location Service Receiver and the Boot Receiver classes.

The *StartLocationServiceReceiver* is responsible for triggering our customized Location Service for retrieving device location information. The Boot Receiver's main task is to schedule when the Start location Service.Receiver should request the location service. Once the Boot Receiver receives the *BOOT_COMPLETED* Intent from the system, it uses the Android's Alarm Manager service to let the receiver schedule a pending Intent to be sent periodically to our Start Location Service Receiver in order to update the device location.

Service. The LocationService service is derived from the IntentService class that facilitates offloading work from the main application's thread, allowing tasks to be performed in the background on a separate thread if desired. Location- Service determines if the device has moved to or still is in a previously registered area. Offloading the aggregation of location-based data in a separate thread reduces the performance impact of the execution of the LocationService on the Policy Manager. We use the AlarmManager to periodically activate the LocationService to ensure the device's location is always up-to-date. By default, the LocationService is activated once per minute, but we give the user the choice to configure how often the service is executed. The duration of the service depends on the number of snapshots of location parameters to be taken, which is currently configured to four per area.

Content Provider: The policies configured by the user are stored within the Policy Manager data directory. This data is private to our custom application and cannot be accessed by other applications or the system itself, as a result of Linux's kernel user ID access control mechanisms. PolicyCP is our custom content provider that acts as a secure intermediary between the policy database and all objects outside of the *Policy Manager's* running process. We chose to use the SQLite database to store user-configured policies due to the support and ease of programming provided by the Android API's associated with storing and managing databases on Android devices.

5.2 Permission Management

In the Android system, all resources that require explicit access rights in the form of permissions are protected by the Activity Manager Service class via permission verification. When an application attempts to use any of these resources, the *ActivityManagerService's* method called *checkComponent* Permission is invoked to verify if the calling application has the appropriate permission(s) to access the resource.

We apply our modifications to this particular method by simply intercepting the permission call before the system performs its standard permission verification process. Given the permission and the application name, the system subsequently calls our custom content provider's *revoke Resource Access* to determine the next course of action. Depending on the user's policy configuration, the next course of action could either be returning the constant Package Manager. *DENIED* in the *checkComponent Permission* if the user has configured to block that permission from the requesting application, or letting the normal verification process take its course. We also give the ability to revoke any or all permissions system-wide via the *Policy Manager's* interface.

5.3 Restrictions on User Data

Our implementation of data obfuscation complements much of the techniques used in [10] and[11], but instead under the domain of CBAC policy restrictions. We obfuscate user data from applications attempting to access it if the policy restriction applies to those applications. We modify the Android APIs that access the user data saved on the device. Relational database systems are the common data management systems used to create, store, and manage user data. Accessing these data usually require calling the *ContentResolver's query()* method, and thus we modify it for our purposes. Instead of returning the expected Cursor object needed to point to the required data, a Null Cursor object is substituted. A *NullCursor* object represents an empty dataset, such as an empty list of pictures as if pictures were not present or never stored on the device.

5.4 Managing System Peripheral State

We also give users the option to configure a policy to restrict access to peripherals (e.g., Bluetooth) when entering a particular location. Specifically, users can set up their devices to prevent applications from modifying a peripheral's current state (enabled/disabled). While it is possible to modify a peripheral's current state by using

permission management, we modify the specific methods that enable/disables these peripherals in order to prevent applications from crashing that do not have code for handling exceptions resulting from revocation of permissions. As an example, for Bluetooth we modified the BluetoothAdapter class and for Wi-Fi we modified the WifiManager class so to assure that these modifications do not result in application crashes and to prevent applications from modifying peripherals current state. Whenever an application tries to modify the state of a system peripheral, our content provider PolicyCP checks the validity of the request and would refuse the request if the request tries to override a user-configured restriction.

5.5 Intent Management

Intent messages are one of the common forms for inter- and intra-communication between application components, sent via three methods: startActivity(), sendBroadcast(), and start- Service(). Preventing an application from sending intents is simply a matter of intercepting the intents when the aforementioned methods are called by applications. Intent interception provides the user the ability to prevent an application component from starting another activity,

broadcasting any possible sensitive information, or executing a possibly suspicious background service. For example, without the need of declaring the Android permission ”RECORD_AUDIO”, an application can indirectly access the device’s microphone recorder application by requesting the Activity class to send a record audio intent. Therefore, we modified the Activity class which hosts startActivity() and startActivityForResult(), and the ContextWrapper class which contains sendBroadcast() and startService().

We modified these methods to intercept the Intents and control the actions performed based on those Intent objects. We classify these Intents based on the contents and description of the intent objects. The user is given, via the Policy Manager interface, the ability to prevent a specific set of Intents from being sent.

6.CONCLUSION

In this work, we proposed a modified version of the Android OS supporting context-based access control policies. These policies restrict applications from accessing specific data and/or resources based on the user context. The restrictions specified in a policy are automatically applied as soon as the user device matches the pre-defined context associated with the policy. Our experimental results show the effectiveness of these policies on the Android system and applications, and the accuracy in locating the device within a user-defined context.

Our approach requires users to configure their own set of policies; the difficulty of setting up these configurations require the same expertise needed to inspect application permissions listed at installation time. However we plan to extend our approach to give network administrators of organizations the same capabilities once a mobile device connects to their network.

REFERENCES

[1] Wikipedia, (May 2013). Samsung galaxy s4 specifications.[Online].Available:*http://en.wikipedia.org/wiki/Samsung_Galaxy_S4*

[2] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel,and A. N. Sheth, “Taintdroid: An information-flow tracking system for real time privacy monitoring on smartphones,” in Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation, 2010, pp. 1–6.

[3] J. Leyden, (Apr. 2013). Your phone may not be spying on you now but it soon will be. [Online].Available:*http://www.theregister*.co.uk/2013/04/24/kaspersky_mobile_malware _infosec

[4] R. Templeman, Z. Rahman, D. J. Crandall, and A. Kapadia, “Placeraider: Virtual theft in physical spaces with smartphones,” in Proc. 20th Annual Netw. Distrib. Syst. Security Symp. (NDSS), Feb. 2013.

[5] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, “Soundcomber: A stealthy and context-aware sound Trojan for smartphones,” in Proc. 18th Annu. Netw. Distrib. Syst. Security Symp., Feb.2011, pp.17–33.SHEBARO ET AL.: CONTEXT-BASED ACCESS CONTROL SYSTEMS FOR MOBILE DEVICES 161

[6] L. L. N. Laboratory, Controlled items that are prohibited on llnl proporty. (2013). [Online]. Available:https://www.llnl.gov/about /controlled items.html

[7] M. Conti, V. T. N. Nguyen, and B. Crispo, “Crepe: Context-related policy enforcement for Android,” in Proc. 13th Int. Conf. Inf. Security, 2011, pp. 331–345.

[8] A. Kushwaha and V. Kushwaha, “Location based services using android mobile operating system,” Int. J. Adv. Eng. Technol., vol. 1, no. 1, pp. 14–20, 2011.

[9] S. Kumar, M. A. Qadeer, and A. Gupta, "Location based services using android," in roc. 3rd IEEE Int. Conf. Internet Multimedi Serv. Archit. Appl., pp. 335–339.

[10] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall,"These aren't the droid you're looking for: Retrofitting android to protect data from imperious applications," in Proc. 18th ACM Conf. Comput. commun. Security, 2011, pp. 639–652.

[11] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino, "Identidroid:Android can finally wear its anonymous suit," Trans. Data Privacy, vol. 7, no. 1, pp. 27–50, Apr. 2014