Enforcing Security Policies in Mobile Devices Using Multiple Personas

Akhilesh Gupta[1,3], Anupam Joshi[1,2], and Gopal Pingali[1]

[1] IBM Research - India,
Plot 4, Block C, Vasant Kunj Institutional Area,
New Delhi 110070, India
akhilesh.iitdelhi@gmail.com, anupam.joshi@in.ibm.com, gpingali@us.ibm.com
[2] CSEE Department,
University of Maryland, Baltimore County
[3] Department of Computer Science,
Stanford University

Abstract. Cell phones are becoming increasingly more sophisticated, and such "Smart" phones are a growing front end to access the web and internet applications . They are often used in a multiple modes – for instance for both personal and business purposes. Enterprises that allow employees to use the phones in this dual mode need to protect the information and applications on such devices and control their behavior. This paper describes an approach that integrates declarative policies, context and OS level device control to enforce security by creating multiple personas for the device. We describe the approach, and present a proof of concept implementation on Android.

1 Introduction

Mid to high end smartphones have become commonplace in corporate settings. The same smartphone is used in a corporate as well as personal setting, has both types of applications, and stores both personal and corporate data. This can lead to significant security concerns. Sensitive data can be compromised by making the device behave inappropriately using one of the installed applications, or by hacking into the device using its connectivity to public networks. Data can even be compromised by the device being lost, or someone stealing the removable storage media or the device itself. While we illustrate this problem using a corporate / personal dual persona situation, more generally we might want the device to have multiple personas. For instance, one when it is connected to a trusted network, another when it is in a particular location (office vs home), and yet another when we are using a particular application. We address this problem by forcing the device to isolate its functionality and behave differently when being used in different contexts using a middleware layer.

We argue that a set of applications, data, and device capabilities are relevant to a context, which we call a *device persona*. Each such persona should have its own sandbox to run, and should be isolated from other personas. What is permitted within this sandbox needs to be dictated by the corresponding policy. We propose a security mechanism in which the device functions in distinct modes, and specifically show an implementation for separating personal use from enterprise use.

For example, the enterprise may want to specify in a policy that device capabilities like 802.11, Bluetooth, and GPRS are unrestricted while in the personal mode. However, if the user wishes to read their official email, it requires the device to switch to the enterprise mode. In this mode, the policy directs the enforcement mechanisms on the device to disable the Cellular radio and Bluetooth, and limit access via 802.11 interfaces to the enterprise VPN. The device platform is modified to actually enforce the issued policies. The enterprise VPN server is entrusted with the task to verify the login credentials of the user requesting to switch to the enterprise mode and to verify the authenticity of the device before granting network resources to the device. The device authentication is carried out by ensuring that the device is actually running the customized build of the platform distributed by the enterprise. We describe a proof of concept implementation carried out by customizing the Android Donut Mobile platform. Note that our approach builds on top of existing mechanisms such as link encryption and user authentication – it does not supplant them. Moreover, it also does not obviate the need for malware detection and remediation systems.

2 Related Work

Most of the current systems for device level security work on a per application basis, typically at install. For instance in Android[1], each application typically gets its own uid. All permissions it needs are predeclared (and signed) by the developer. At install time, these permissions are either granted to the uid or not based, amongst others, on interactions with the user. This is very coarse grained control. Ideally, the decision on whether or not to grant a request to access device features from an application should be based on the context of the device and the user. For instance, an enterprise may have a policy that cell phones with cameras are not allowed to take pictures inside the office, or that they must be in vibrate mode when in a meeting room. These are all instances of the context of the device, as captured in the policy of the space it happens to be in. Similar context dependant policies arise from the user's perspective. A user might want to make his Bluetooth device discoverable when at home or office, but not otherwise. We posit that a context dependent, policy driven security mechanism is best suited for device capability security.

Access control is a very well studied topic in literature, with a variety of models that have been developed over four decades of work. We focus here on two works of immediate relevance. The first, by Jansen et al[2] suggests that devices such as PDAs be provided enterprise security policies (in XML) on a smart card. These policies would describe access/noaccess decisions for specific device features. The PDA would run a trusted kernel which would enforce these policies.

[1] `http://www.android.com/`

The focus there was to provide policies, not base them on changing context. We extended this approach in our prior work[5] and showed that a pointer to such a policy could be sent over the 802.11 beacons, so that a "smart space" could point a device to its acceptable use policy. Our proposed approach builds upon some of these ideas. Susilo [6] identified the risks and threats of handheld devices connected to the internet. Since the mobile device is not subject to physical security as are the fixed computers in the wired networks, it is susceptible to attacks and hence can potentially host malicious code while it is in some untrusted network and try to propagate it, once back in the home network. Another scenario involves a temporary user granted access to the network injecting malicious code into the network.

Our system uses a declarative policy-based approach, where the rules of behaviour, or the boundaries of the sandbox, of entities in a variety of environments are described in a machine-understandable specification language. Policy driven systems can even be engineered so as to be extremely lightweight on resource constrained devices[5]. Semantic web languages such as RDF and OWL prove a natural choice for such policy languages.

Policy driven security has been looked at by several academic research groups over the last decade. Rei [3] is an example of a *declarative policy language* that uses Semantic Web technologies to describe policies as constraints over allowable and obligated actions on resources in the environment. Rei is, of course, just one of the recent efforts to develop declarative policy languages. Most are not, unlike Rei, motivated by the security and privacy issues in open systems such as ubiquitous computing. These include industry standards such as XACML [4], but also academic efforts such as Ponder [1].

3 Design Approach

Android is a free, open source, and fully customizable mobile platform by Google and the Open Handset Alliance. Before we discuss our implementation in the next section, there is a present a quick overview of three relevant components of the Android platform viz. *services, intent broadcasts and security.* A more general discussion of Android is beyond the scope of this paper. However, note that security is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. The permissions required by an application are declared statically in that application, so they can be known up-front at install time and will not change after that.

A *service* runs in the background for an indefinite period of time. Most core system functions are carried out by services. A *broadcast receiver* receives and reacts to broadcast announcements. Many broadcasts originate in system code to notify the rest of the system about a state change. Services and broadcast receivers are activated by asynchronous messages called *intents.* These *intents* name the action being announced making the concerned service handle the state as required.

There are several possible approaches to provide each persona with its own sandbox. The most obvious approach is to build in separation at the application level using the access control mechanisms provided by the OS. However, these are generally limited in most mobile platforms. More importantly, they can be easily overriden by the user. Another possibility is to introduce modifications at the kernel level. This approach is based on the idea of Jansen et al.[2]. However, this approach adds an extra layer to check to every access request thus raising the overhead significantly.

For Android, we consider an alternate approach that creates each sandbox as a "mode" similar to built in modes like the "Airplane mode". While such modes merely switch off certain functionality by default, they can be extended to perform the functions we need. We add such modes to the stock Android distrubution (1.6) and build a customized image from source to achieve our objectives. We utilized the Android system services to disable interfaces at the framework level. This also allows us to work with the Android security mechanisms and disable user controls in device settings preventing manipulation of the functioning in the modes, and not allowing functionality to be added to the sandbox by the user. Moreover, this approach makes it easier to work with Android networking for authentication procedures and the Volume daemon for working with accessibility of file-systems, both of which are useful for providing the sandbox. To keep the initial implementation simple, we start with the policy expressed in XML, and directly describe the device constraints.

4 Implementation

For our implementation on the Android platform, certain setup functions are performed during initialization of the device. An application has been created for the purpose of accepting the user's enterprise login credentials. The user settings interface presents a button to issue the command for switching back and forth between the two modes.

During the boot process, the device reads the enterprise policy from the policy file (included as part of the build) and configures enterprise mode accordingly. The volume daemon detects the external storage card and stores its configuration and mount point. The device boots up in the personal mode. On receiving the boot completed *intent*, the mount *service* on the device unmounts the external storage reserved for the enterprise mode.

The user opens the application for entering enterprise login credentials. When the user issues the command for switching the mode, the credentials are sent to the enterprise server for validation. In our prototype, we use IBM's internal single sign on system to validate the user.

An *intent* is generated to indicate the switch to the new mode and is broadcast to inform the rest of the device. The *services* corresponding to each of the radio interfaces execute the handler for this *intent*. The handler decides if the radio interface is sensitive to the enterprise mode and enables or disables the corresponding radio based on the policy. The status icons on the user interface are updated to present the current status of various wireless interfaces on the device. The user interface buttons corresponding to each of these radios also handle this *intent* and enable or disable their interface presence. In our example policy demonstrated in the screenshots, all network connections are disabled when in enterprise mode, and only the VPN connection to the enterprise gateway is permitted. This will prevent data leaks on any network path.

The mount *service* invokes the volume daemon to mount or unmount the external SD storage. We mount it for enterprise mode, and unmount it for regular usage. For added security, the filesystem on the SD card could be encrypted. The key, instead of being stored locally on the device, would be provided by the enterprise gateway via the encrypted VPN channel upon successful authentication. Note that we chose to keep the on device flash filesystem encryption free to comply with FCC mandates on smartphones that they be allowed to boot into a mode allowing calls to 911. An Android application can store data on the device memory itself which is private to that application ensured through linux file-system permissions. However, any data stored on the external storage is not secured. Thus, in our implementation, the external storage is reserved for enterprise data and can be accessed only in the enterprise mode. The external storage in Android is handled by the Volume daemon. The external storage is unmounted as soon as the user switches to the personal mode. We show in Figure 1, the disabling of the radios/network and mounting of the external SD card in response to a successful authentication into the enterprise mode.

The user can go back to the personal mode by unselecting the enterprise mode. In our present implementation, no authentication is done for this reverse switch, although it could be easily added if desired.

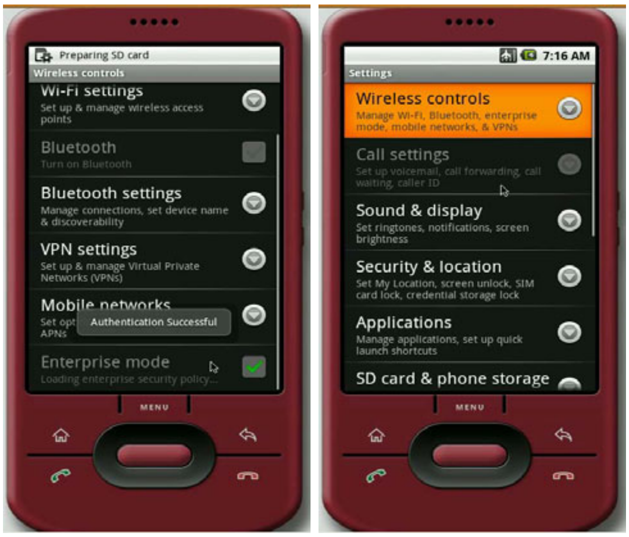
Preparing SD card
Wireless controls
Wi-Fi settings
Set up & manage wireless access points
Bluetooth
Turn on Bluetooth
Bluetooth settings
Manage connections, set device name & discoverability
VPN settings
Set up & manage Virtual Private Networks (VPNs)
Mobile networks
Set opt
APNs
Authentication Successful
Enterprise mode
Loading enterprise security policy...
MENU
7:16 AM
Settings
Wireless controls
Manage Wi-Fi, Bluetooth, enterprise mode, mobile networks, & VPNs
Call settings
Set up voicemail, call forwarding, call waiting, caller ID
Sound & display
Set ringtones, notifications, screen brightness
Security & location
Set My Location, screen unlock, SIM card lock, credential storage lock
Applications
Manage applications, set up quick launch shortcuts
SD card & phone storage
MENU

Fig. 1. Authentication with enterprise server successful. Radios mentioned in enterprise policy get disabled. SD card mounted.

An Android source patch has been created for ease of application of the modification made to the platform to the original source code. This modified source can be built for an actual android device to generate a system image. This system image can then be signed by the enterprise for installing onto real devices. The customized Android build can be flashed to devices and then distributed to employees of the organization. New policy files and build patches can be supplied by the enterprise network to the devices wirelessly using the Android OTA (over-the-air) update mechanism. We note that arbitrary personas can be created using our system that are specific to the particular needs of an enterprise or organization. Each persona can specify an arbitrary combination of functionality that is available.

5 Conclusion and Future Work

Devices such as smartphones are increasingly being used in both personal and professional/enterprise context. This creates serious security concerns about loss of data from the compromise or loss of the phone. Devices can also be attacked using their network connections. We have proposed a solution to this problem by having the device take on multiple personas, each corresponding to a sandbox where certain applications and device functionalities are allowed. What functionalities are allowed are based on the needs of the mode (e.g. personal, enterprise, travel hotspot, etc.) and are specified by a declarative policy.

References

1. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–37. Springer, Heidelberg (2001)
2. Jansen, W.A., Karygiannis, T., Gavrila, S., Korolev, V.: Assigning and Enforcing Security Policies on Handheld Devices. In: Proceedings of the Canadian Information Technology Security Symposium (May 2002)
3. Kagal, L., Finin, T., Joshi, A.: A Policy Language for A Pervasive Computing Environment. In: Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks (June 2003)
4. Moses, T., et al.: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, 200502 (2005)
5. Patwardhan, A., Korolev, V., Kagal, L., Joshi, A.: Enforcing Policies in Pervasive Environments. In: International Conference on Mobile and Ubiquitous Systems: Networking and Services. IEEE, Cambridge (2004)
6. Susilo, W.: Securing Handheld Devices. In: 10th IEEE International Conference on Networks (August 2002)