Remote Policy Enforcement for Trusted Application Execution in Mobile Environments*

Fabio Martinelli[2], Ilaria Matteucci[2],
Andrea Saracino[1,2], and Daniele Sgandurra[2]

[1] Dipartimento di Ingegneria dell'Informazione,
Università di Pisa, Pisa, Italy
name.surname@iet.unipi.it
[2] Istituto di Informatica e Telematica,
Consiglio Nazionale delle Ricerche, Pisa, Italy
name.surname@iit.cnr.it

Abstract. Both in the cloud and mobile environments, a large number of online services is daily accessed through smartphones and tablets. Since several security, safety and trust concerns may arise when using these services, providers may require a usage policy to be enforced on the devices while accessing these services. This kind of policy enforcements enables service providers to have assurance that remote devices are in an acceptable state when using the provided service, according to their terms and conditions.

In this paper, we propose a framework which allows service providers to have assurance about the enforcement of some functional policies directly on the device. The proposed framework inserts an *enforcer* into the client's device, which is responsible for enforcing the provider's policy to abide by the terms and conditions of the service. To assure the integrity of the enforcer and of the policy, the framework exploits Trusted Computing techniques to remotely attest the enforcer's measurements. Preliminary experiments and a first prototype implementation for Android-based smartphones suggest that the approach is both viable and effective.

1 Introduction

Mobile devices are used to access a large number of online services, such as e-banking, multimedia streaming, location services, videogames and social networking. Since some security and safety concerns may be raised when using these online services, providers may be willing to have some assurance that, when using them, devices are in a well-know, and acceptable, state. As an example, it is safe if no key-logging software is active on the device during an e-bank transaction, due to the sensitive data exchanged. Hence, a safe usage of online services may require the definition of specific behaviors

* The research leading to these results has received funding from the EU Seventh Framework Programme (FP7/2007-2013) under grant n. 256980 (NESSoS), n. 257930 (Aniketos), from PRIN Security Horizons funded by MIUR with D.D. 23.10.2012 n. 719, and EIT ICT Labs activity 13077.

for the remote clients. These behaviors forbid actions that are incompatible with the service itself.

The set of the correct expected behaviors defined by a provider constitutes a *provider policy*. The definition and enforcement of a policy should provide a safe service usage both from the user and provider point of view and may also be part of an agreement on the service conditions between the user and the provider. In fact, in this scenario, the provider may grant a determined Quality of Service (QoS) only if the user behavior is compliant with a certain specification. For example, a video-streaming provider may grant the streaming of high definition videos with a negligible latency, assuming that the available bandwidth is higher than a predefined threshold. However, supposing that the user is filling the bandwidth using other network-exhausting programs (e.g., through torrent, download manager, etc.), the QoS of the agreement cannot be provided. The monitoring of the effective behaviors, to discover if there are policy violations, can be hard to implement in real-scenarios since (i) provider policies may concern behaviors of third-party applications, OS, users or other components that are not controlled by the provider itself; (ii) it may be deceived by malicious applications installed on a device.

To overcome these issues, we propose a provider policy enforcement framework based upon remote attestation. The proposed framework allows providers to have assurance that service policies are enforced by remote devices when using their services. To this end, policy compliance is applied by a global enforcer running on the device, which monitors all the security-relevant events on a mobile device. The enforcer periodically submits policy reports to the service provider. To ensure that both the enforcer and the policies have not been compromised, or faked, the framework exploits a Trusted Platform Module (TPM) on the device to build a chain-of-trust. In this way, the provider can have assurance that, when enforcing the policy, the remote device is in a trusted state.

Contributions. The contributions of the paper are multifold and include the following:

- we propose a remote measurement framework to provide assurance to service providers about the enforcement of policies on remote devices when accessing their services;
- we discuss the format of the policy to be globally enforced on remote devices: the policy is sealed with the application, and it specifies acceptable behaviors that are compliant with the terms of service;
- we provide a protocol for both attesting the initial state of the device and to alert the provider in case of policy violations;
- we discuss the first prototype implementation and the first results.

Outline of the Paper. The paper is organized as follows. Section 2 recalls some background notions related to the Trusted Computing Platform and to the Android Framework. In Sect. 3 we describe in detail the architecture of the proposed framework. Section 4 presents the first prototype of the framework along with some examples of the policies instantiated to some real use cases. Section 5 discusses some related works while Sect. 6 concludes by proposing further extensions.

2 Background

In this section we recall some notions about Trusted Computing, Android system, and the main components requested to build a remote attestation service.

2.1 Trusted Computing

One prominent framework of integrity measurement is promoted by the Trusted Computing Group (TCG), which is an industry consortium that defines specifications for hardware and software components [1]. The standard TCG measurement basically requires the computation of SHA-1 cryptographic hash of critical software components as soon as they are loaded into the system. The TCG guidance for measurement includes the Trusted Platform Module (TPM) as a hardware device to securely store and report measurement values through SHA-1 hash. This architecture provides a good framework for determining the integrity during software initialization. Since the TCG framework is pretty complex, we detail here some of its main components:

- RTM (Root-of-Trust for Measurement) is a on-device chip capable of performing reliable integrity measurements. This is the root of the chain of transitive trust.
- CRTM (Core Root-of-Trust for Measurement) is the small set of instructions that are executed by the platform when it acts as the RTM.
- RTS (Root-of-Trust for Storage) is the part of the framework responsible for maintaining a not-modifiable summary of values of integrity digests and the sequence of digests.
- RTR (Root-of-Trust for Reporting) is the part of the framework that reliably reports information held by the RTS.
- PCR (Platform Configuration Register) are the set of physical, and tamper-proof, registers containing a digest of integrity digests.
- TPM (Trusted Platform Module) is used to implement all the functions defined in the TCG specification and includes the set of Root-of-Trust with shielded locations and protected capabilities.
- TSS (TPM Software Stack) is a library used by higher-level services that facilitates the use of the TPM.

In the following, we describe in more detail the TPM functionalities.

TPM and MTM. The TPM acts as a root-of-trust in the process that builds and configures the software environments and it ensures that a system has loaded its software properly. Moreover, it protects secrets such as asymmetric and symmetric keys. The TPM has a set of registers that are protected from the system software. The TPM implements two operations on each register content: `extend` and `quote`. The `extend` operation takes a value `V` as input and computes the SHA-1 hash of the current register content and `V`. This function is used to compute and store the hash of new values. Instead, in a `quote` operation, the TPM generates a message with the register contents and signs it with a private key protected by the TPM.

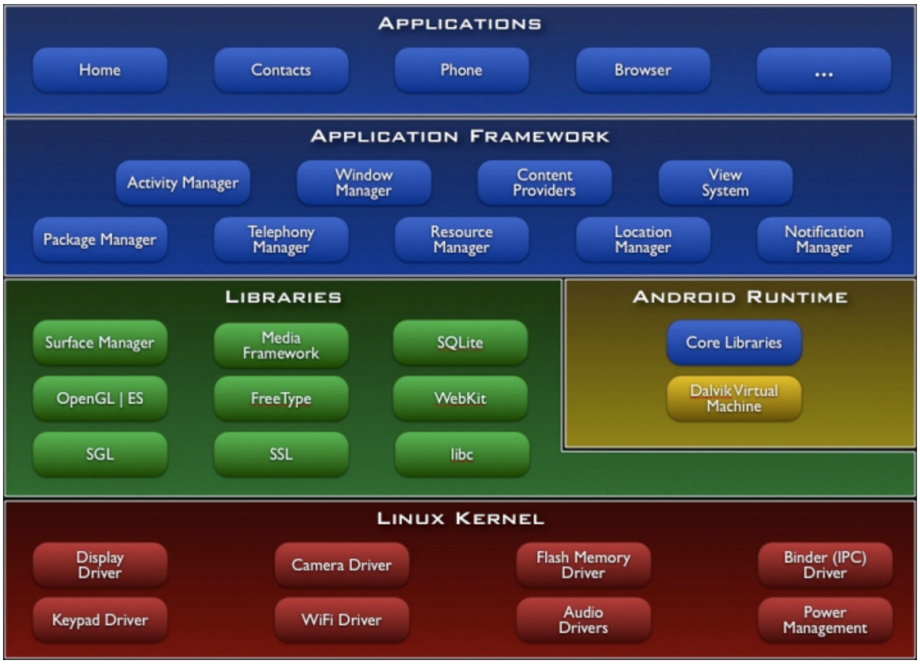

Fig. 1. Android Architecture (From [2])

The Mobile Trusted Module (MTM) is both a security device and an approved TCG specification [3] [4] for mobile devices. The specification for mobile devices differs from TPM specifications by introducing the concept of secure boot and by specifying the implementation of the MTM as a functionality rather than as a physical implementation in hardware and, finally, by taking into account the support of several coexisting MTMs in the same device. As an example, some of these may enforce discretionary policies (e.g.., MTMs exposed to user applications) whereas others may enforce mandatory security policies(e.g., the device manufacturer MTM).

2.2 Android

Android is an open source Operating System (OS) designed for mobile devices, such as smartphones and tablets, which currently has the largest share of the mobile device market. Android is a complex framework divided in several functional blocks and levels (Figure 1). The lowest level is a Linux kernel, cross-compiled through a toolchain, in order to run on mobile device architectures, i.e. ARM processors. Some binaries that can be commonly found on desktop distributions have been removed, to produce a lighter kernel more suitable to mobile devices. Applications (*apps*) run at higher level (Application Level) in a sandboxed environment. Each application runs in a virtual machine called *Dalvik Virtual Machine (DVM)*, which is a Java Virtual Machine optimized to run on mobile devices. Every instance of the DVM is handled as a different Linux user, with its own storage and virtual memory space. This ensures application isolation,

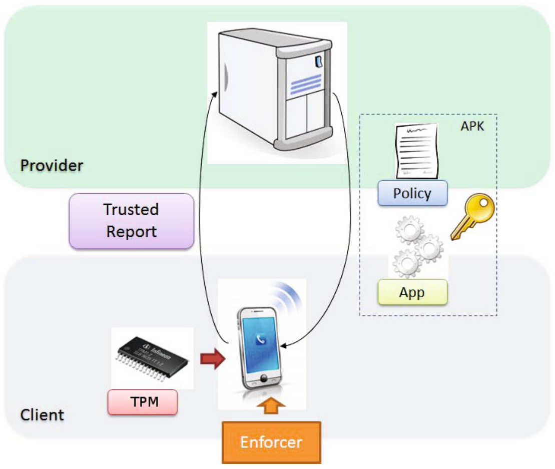

Fig. 2. Client-Provider Architecture

i.e. each application runs in its sandbox and cannot interfere with other running or idle applications.

Both the application framework and the libraries level offer a large number of APIs to allow applications to interact with device components and kernel functions. The number of applications available for Android systems is continuously growing. The official online market for applications, i.e., Google Play, distributes more than 700K apps. Several applications require Internet access to provide a service and are based on a classical client-server paradigm, where the server provides a service to all the clients (mobile devices) requesting it.

3 Architecture

This section describes the architecture of the proposed framework, and it specifies the components that have to be ported on Android systems. The architecture is composed of two main parts: the client-side, i.e. the on-device enforcement mechanism, and the provider-side part, i.e. the provider policy specification. A high-level view of the system is shown in Fig. 2.

We first start by describing some real-world use cases to show the viability and the benefits of the proposed architecture.

Secure Driving. Consider a street navigation software, e.g. `Google Maps`, which allows users, e.g. in cars, to continuously receive route directions while driving. The provider of such a software may require that while driving texting and app-browsing are forbidden to avoid possible distractions while driving that may pose serious risks to the safety of the driver[1]. In fact, if the application is used, this means that the user may be

[1] This could be based upon the speed of the user/device, e.g. to avoid forbidding messaging apps while walking.

in a potential dangerous situation. Hence, an example of a remote policy enforcement is to block all the operations that require a user manual interaction to avoid possible distractions while driving.

Flight Mode. Usually, while on board of a plane passengers are asked to keep their mobile phones in Flight Mode (radio interface down) for the entire duration of the flight, as the radio interface may interfere with the airplane system. In such a situation, the airline company may enable users to download an application that constantly updates their phones through Wi-Fi on the current plane position[2], weather condition of the final destination, delay if any. In such a scenario, a policy to be enforced on the device is to disable the telephony radio interface and only allow Wi-Fi to access on-board services while flying.

Real Location. The provider of a location-based service may require that no services of location obfuscation are active on the device. Location-based services provide a reliable service only if the given location is correct. In such a scenario, a policy to be locally enforced on the device may be based on a black-list of applications, known to obfuscate the location, that cannot run on the device while the location-based service is active

Game Fairness. Online games are known to be populated by unfair players that try to cheat to gain more popularity / points or simply to provide denial of service both to the service provider and other players. As an example, users may exploit some applications to simulate a heavy network latency, slowing the reaction of their adversaries. In this scenario, a game-provider would like to forbid the usage of cheating applications when the user is playing its online video-game.

3.1 Provider-Side Architecture

Service providers may require to have some assurance that, when using their online services, remote devices are in a well-known, and acceptable, state. Hence, in the proposed framework, users access online services through apps developed and distributed by the provider itself that are shipped together with a policy to be enforced. A *policy* is a formal complete specification of the acceptable security-relevant behavior allowed to applications executed on the platform [5]. Policies may also concern user or OS behaviors and may be expressed using (i) several formalisms, such as formal specification languages [6], (ii) high-level or natural language, or (iii) execution graphs. To enforce a policy, the behaviors that are of interest should be constantly monitored, by verifying that they match the ones described in the policy, otherwise they should be forbidden.

Policy Format. Each provider policy is written in eXtensible Markup Language (XML) using a nested tag structure to specify allowed and disallowed actions. In the current implementation the full list of XML tags enables control on the following elements:

– Blacklist and whitelist of installed or running applications.
– Usage of network interfaces, such as data connection, Wi-Fi, Bluetooth and NFC.

[2] E.g., on Windows Phone devices, during flight mode it is possible to enable Wi-Fi.

Table 1. Policy Specification

```
<policy default_reaction={deny, report, lower_trust}>
<networking>
      <wifi reaction={deny, report, lower_trust}> {enabled,disabled} </wifi>
      <radio reaction={deny, report, lower_trust}> {enabled,disabled} </radio>
      <bluetooth reaction={deny, report, lower_trust}> {enabled,disabled} </
          bluetooth>
      <nfc reaction={deny, report, lower_trust}> {enabled,disabled} </nfc>
</networking>
<telephony>
      <call reaction={deny, report, lower_trust} time_per_day=[1,1440]
          number_per_day=[1,inf] direction={outgoing, incoming}> {enabled,
          disabled} </call>
      <sms reaction={deny, report, lower_trust} number_per_day=[1,inf] direction={
          outgoing, incoming}> {enabled,disabled} </sms>
      <mms reaction={deny, report, lower_trust} number_per_day=[1,inf] direction={
          outgoing, incoming}> {enabled,disabled} </mms>
</telephony>
<location>
      <gps reaction={deny, report, lower_trust} time_per_day=[1,1440]> {enabled,
          disabled} </gps>
      <cell_location reaction={deny, report, lower_trust}> {enabled,disabled} </
          cell_location>
      <ip_location reaction={deny, report, lower_trust}> {enabled,disabled} </
          ip_location>
</location>
<input>
      <touchscreen reaction={deny, report, lower_trust}> {enabled,disabled} </
          touchscreen>
      <keyboard reaction={deny, report, lower_trust}> {enabled,disabled} </
          keyboard>
      <side_button reaction={deny, report, lower_trust}> {enabled,disabled} </
          side_button>
      <voice_input reaction={deny, report, lower_trust}> {enabled,disabled} </
          voice_input>
</input>
<output>
      <ringtone reaction={deny, report, lower_trust}> {enabled,disabled} </
          ringtone>
      <speakerphone reaction={deny, report, lower_trust}> {enabled,disabled} </
          speakerphone>
</output>
<app_running>
      <app1 package_name={package_name1}> {enabled,disabled} </app1>
      <app2 package_name={package_name2}> {enabled,disabled} </app2>
      ...
      <appn package_name={package_name1}> {enabled,disabled} </appn>
</app_running>
<app_installed>
      <app1 package_name={package_name1}> {enabled,disabled} </app1>
      <app2 package_name={package_name2}> {enabled,disabled} </app2>
      ...
      <appn package_name={package_name1}> {enabled,disabled} </appn>
</app_installed>
</policy>
```

- Enabled input mechanisms, e.g. touch-screen, physical keyboard, voice commands, etc.
- Outgoing voice traffic, SMS/MMS messages, raw data through network interfaces.
- Black list and white list of contacts.
- Usage of location providers.

The full policy specification language is described in Table 1. The policy can also be extended by adding new XML tags to control more smartphone elements. Each XML tag specifies the policy for a critical device component using a nested structure. The possible value for each tag is either `enabled` or `disabled`, where `disabled` means that the component cannot be used. Through attributes it is possible to define finer grained policies. By default each component is enabled. The attribute `default_reaction` of the policy tag specifies the default reaction for policy violations. This reaction can be customized for each component using the `reaction` attribute of each policy subtag. The criticality order is, from the less critical to the most critical: `lower_trust`, `report`, `deny`.

To implement these specifications on Android devices, the framework needs to update the `AndroidManifest.xml` file. In Android, every application comes in the form of an APK file. Android application package file (APK) is the file format used to distribute and install application software. Every APK must have an `AndroidManifest.xml` file in its root directory. The manifest presents essential information about the application components and security relevant authorization that the application requires to work correctly. Since the policy is part of the application (APK), we have decided to express the policy in XML so to be included in the `AndroidManifest.xml` file. This file is bound to the application by means of digital signature, to ensure the integrity of both application and policy. When installing the application, the user accepts the provider policy, which will be enforced on the device.

3.2 Client-Side Architecture

In this section we detail the on-device components required to support the integrity measurement/reporting and policy enforcement.

Enforcer. The enforcer is a multi-layer component, which monitors the action performed on the device, and enforces the provider policy. In the current implementation, the monitoring is performed both at application-level and kernel-level, but this component can be extended to enforce policies and monitor behaviors at any level. When the enforcer detects a behavior that is non-compliant with a policy, the enforcer executes a *reaction*, which is an action specified in a policy to react to a performed misbehavior. Examples of reactions are:

- *Deny*: the misbehavior is blocked before it takes place;
- *Report*: the provider is notified about the non-compliance of the device/application with the policy;
- *Trust Lowering*: the trust level of the client is lowered. Different strategies may be applied towards clients with low trust levels.

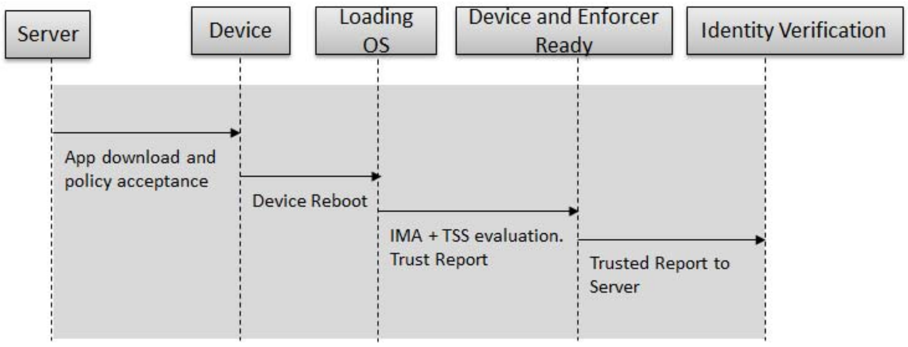
Server
Device
Loading OS
Device and Enforcer Ready
Identity Verification
App download and policy acceptance
Device Reboot
IMA + TSS evaluation. Trust Report
Trusted Report to Server

Fig. 3. Remote Attestation Protocol

In the current prototype, the enforcer has been implemented by modifying the system at the application level and inserting a Linux module at kernel-level. The part of the enforcement at application level monitors the device interfaces activities, like GPS, Wi-Fi, Bluetooth and NFC. This component also controls and handles the events of outgoing and incoming phone calls and SMS/MMS messages. The kernel module tracks all the running processes and is responsible of stopping (or reporting, or lowering the trust level of, according to the policy) the applications that violate a policy.

Trusted Policy Enforcement. Integrity of the client-side architecture is assured through usage of Trusted Computing. The Trusted Computing building blocks that have to be included on the Android platform have been described in [16], and are the following: a Root-of-Trust for Measurement (RTM), a Root-of-Trust for Storage and Reporting (RTS/RTR) and a Static Chain of Trust (SCoT). To build the Chain-of-Trust, the Linux kernel is enhanced by including the Integrity Measurement Architecture (IMA). The enforcer exploits a minimal TSS implementation for PCR `extend` operations, to allow its measurement functions to communicate the results to the TPM, and for PCR `quote` to enable the trustworthy reporting of the stored value registers of the PCR to the remote provider.

The framework requires a verification of the initial device integrity through a set of measurements on its configuration, which includes a set of hash computations of the code of its kernel and of the running applications. The root-of-trust is rooted to the physical platform TPM. Hence, to measure the initial integrity of the device, starting with the TPM, the following steps are required. Firstly, the TPM applies a set of measurements on the boot-loader, so that from now on, all the steps can be measured from boot to kernel loading and its modules. Attestation requires that the measurements of the device are certified by the keys stored in the TPM and that the provider can establish trust in the device's integrity based upon these measurements. By computing the hash of the running software, signed by the private key of the TPM, the provider can be assured of the trustworthiness of the data received. Then, further integrity measurements are performed by IMA, ported for Android, which communicates with the TPM to safely measure, and store the results of, all the executables loaded on the device as well as the Dalvik VM, run-time libraries and the enforcer.

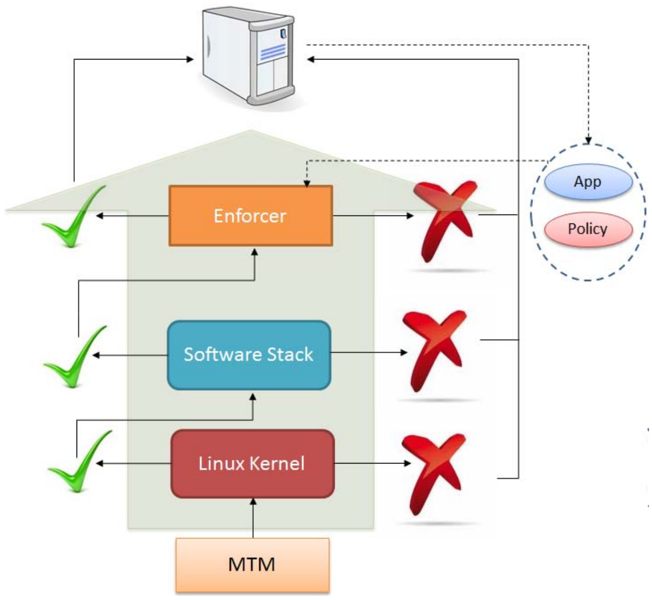

Fig. 4. Trusted Chain

These steps establish the first chain-of-trust up to the Dalvik VM and the enforcer. Then, the enforcer is responsible for ensuring that the chain-of-trust reaches the considered application. To this end, the enforcer continuously monitors the device status according to the received policy, i.e. by forbidding any unacceptable behavior and/or by reporting the misbehavior to the service provider. Once the attestation tokens (PCR quote and measurements logs) are received by the service provider, which acts as a challenger, the service provider needs to verify the trustworthiness, and policy compliance, of the remote device. This basically means to validate the digital signature on the quotes. This step is necessary to verify that a genuine TPM vouches for the measurements logs that, hence, are not fake and unmodified. To verify this, the service provider requires the public portion of Attestation Identity Key (AIK), which is issued using by a certification authority. The AIK is used for platform authentication, platform attestation and certification of keys. The whole chain-of-trust of the proposed framework is depicted in Fig. 4.

4 Preliminary Tests

This section reports a brief description of the current implementation on Android devices and presents some experiments with some example policies.

4.1 Current Prototype

In the current prototype, since no Android devices include a MTM module, some MTM emulators exist that implement their functionalities in software. Since we only need few

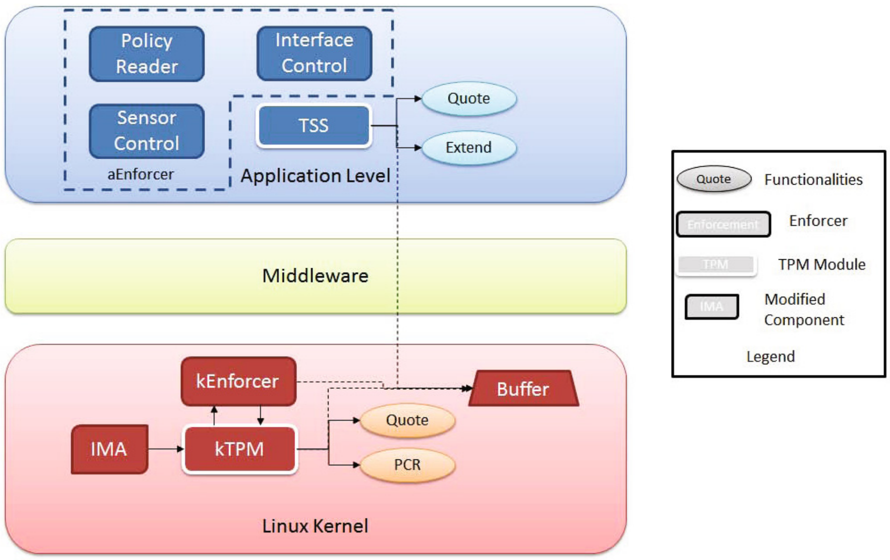

Fig. 5. On-Device Framework Architecture

of these functionalities, we have implemented a simple MTM emulator that provides TPM quote and protected storage only as a kernel module, called *kTPM*. The functions exported by kTPM can be called either by the kernel itself, through the IMA (modified to call these functions, i.e. the communication with the TPM is emulated through these two functions) and the enforcer, which includes the TSS (Trusted Software Stack) that communicates directly with kTPM. This communication is implemented by a protocol on a shared buffer in the `/proc/` directory, where basically the TSS writes the requested action and the parameters, then it loops until results are written by the kTPM.

As shown in Fig. 5, the enforcer is composed of several components, divided in two parts. The first part is the enforcer at the application-level (*aEnforcer*), and the second one at the kernel-level (*kEnforcer*). The *aEnforcer* includes the *policyReader*, which takes as input the policy in XML format. The *aEnforcer* also includes the *sensorController* and *interfaceController*, which are used to monitor and control the activities of the various sensors and network interface. As an example, it can shutdown interfaces on request if the policy requests so. Finally, the *aEnforcer* also includes the minimal TSS implementation to directly communicate with kTPM.

The kernel-level part of the enforcer, i.e. the *kEnforcer*, communicates bidirectionally with the application-part through the `/proc/kenf` file. As an example of policy enforcement, the *kEnforcer* may kill all the processes related to an application that violates a policy.

4.2 Experimentations

We have tested the prototype in all the use-cases described in Sect. 3 by simulating the environment. To this end, we first have coded four simple applications that emulate the functionalities requested by each scenario, e.g. a simple game for the Game Fairness scenario and the corresponding policy. The provider is emulated through a simple PHP server that sends the APK file of the tested applications with the manifest file including the policy and communicates with the emulator and then listens for communications through an exported interface of a Web Service.

In all of the four tests, the policy has been correctly enforced: as an example, as soon as an action violates the policy, the application has been killed by the the *kEnforcer* or the *aEnforcer* has reported the misbehavior to the server/provider. In the following, we report all the XML policies used in the tests.

Table 2. Policy Specification of Secure Driving Use-Case

```
<policy default_reaction="deny">
  <telephony>
    <call direction="outgoing">disabled</call>
    <sms direction="outgoing">disabled</sms>
    <mms direction="outgoing">disabled</mms>
  </telephony>
  <input>
    <touchscreen>disabled</touchscreen>
    <keyboard>disabled</keyboard>
    <side_button>disabled</side_button>
  </input>
</policy>
```

Secure Driving. This policy is active only when the navigation mode of the software is active, which is after the user has chosen the destination and is driving. To avoid that the user get distracted by the smartphone, all interaction that require an active user interaction have been disabled. The user can only interact with the device using voice controls. The full policy is described in Table 2.

Table 3. Policy Specification of Flight Mode Use-Case

```
<policy default_reaction="report">
  <networking>
    <radio>disabled</radio>
  </networking>
  <app_installed>
    <app_1 package_name="com.myflightcompany.surfandfly">enabled</app_1>
  </app_installed>
</policy>
```

Flight Mode. This policy mandates that the user keeps the radio interface of her phone disabled during flight. This is a safety requirement that should be enforced when the user is on-board. The policy is violated if the user enables the radio interface or uninstall the application, e.g. trying to avoid the policy enforcement. In both cases, the enforcement strategy is *report* so that the server, together with the cabin crew of the airplane, will know if the policy has been violated and who is violating the policy. The full policy is described in Table 3.

Table 4. Policy Specification of Game Fairness and Real Location Use-Cases

```
<policy default_reaction="deny">
  <app_running>
    <app_1 package_name="com.myfakelocator.fakelocator">disabled</app_1>
  </app_running>
</policy>
```

Real Location and Game Fairness. This policy forbids programs able to forge the user location to run while the provider location service is running. Similarly, a list of known cheating programs for online videogames may be blacklisted using the same technique. The full policy is described in Table 4.

5 Related Work

The design and implementation of Integrity Measurement Architecture (IMA), a secure integrity measurement system for Linux, is discussed in [8]. IMA enables automatic measurement of all software, such as program, libraries, and kernel modules, and it can also be used to measure static data files when specified by the software. [9] discusses an access control architecture that enables corporations to verify the integrity of a remote client and establish trust into its ability to enforce a security policy before allowing the client to access corporate Intranet services. *Property based attestation* [10] [11] is a framework to describe the behavior of the platform to be attested with respect to security-related requirements. As an example, a property may state that a platform has built-in mechanisms to conform to the privacy laws, or that it strictly separates processes from each other. With property attestation, a verifier is securely assured of security properties of the execution environment of the verified platform without receiving detailed configuration data. *Semantic integrity* [12] is a measurement approach targeting the dynamic state of the software during execution and, therefore, providing fresh measurement results. This approach can provide increased flexibility for the challenger, because the integrity monitor can examine the current state of a system to detect semantic integrity violations. Prima [13] is an extension of the Linux IMA system, that measures information flow integrity and can be verified by remote parties. [14] presents a framework to protect a mobile application at run-time through the use of TCG technologies. Application developers define an application policy that is enforced locally on the device. Examples of such policies are controlling which users can run the applications or what kind of results they can observe. A framework for remote attestation, implemented on the Android platform is presented in [15]. However, the presented framework is only used to verify the device integrity, ensuring that no unknown software is running on the device. [16] proposes an attestation approach for Android smartphones that integrates TCG and the Android's permission system, in particular by attesting the permissions used by the installed applications to a remote party at run-time. The authors of [17] propose a malware prevention architecture for smartphones that exploits applications signatures, process authentication and verification. The proposed framework allows a smartphone to run only trusted applications, e.g., signed applications, and those that are not modified. The trust of an application is a function of the application signature and MTM and is propagated through the processes through process authentication. Differently from the previous approach, the framework proposed in this paper is the first one that exploits Trusted Computing to verify the remote enforcement of provider policies. Finally, Mobile device management (MDM) [18] is a way to monitor and manage mobile devices deployed across mobile operators, enterprises by distributing applications, data and configuration settings over-the-air.

6 Conclusion and Future Work

In this paper we have presented a policy enforcement system for Android applications that access online services. Policies are given by service providers, to enforce both safety and security for the device and for the user. Policy enforcement is assured through an enforcement module included in the Android system and a Trusted Computing Platform, which ensures the integrity of reports sent to the server. We have presented a policy specification formalism that exploits XML language and the modification requested to include our system on Android devices. The proposed approach enables the definition and enforcement of provider policies, which users have to accept to access the service. The current implementation exploits an emulated MTM to implement the functions of Trusted Computing, which are requested to assure authenticity and integrity of the reported device status.

A first future extension of this work consists in including our framework on mobile devices with a real MTM. To the best of our knowledge, currently there are no devices that include both the MTM and the Android OS. Afterward we are planning to test our framework in an enterprise environment that uses the Bring Your Own Device paradigm with employers, by considering finer grained policies that takes in account context information. Finally, we plan to reduce the number of modifications requested to a standard device by requiring the provider to include into a single bundle both the application and the enforcer, along with the policy.

References

1. Pearson, S.: Trusted Computing Platforms, the Next Security Solution. Trusted Computing Group Administration, Beaverton (2002)
2. Wikipedia: Android operating system (2013), `http://en.wikipedia.org/wiki/Android_(operating_system)`
3. Trusted Computing Group: Mobile phone work group mobile trusted module specification, version 1.0, revision 7.02 (2013)
4. Trusted Computing Group: Mobile phone work group mobile reference architecture (2013)
5. Greci, P., Martinelli, F., Matteucci, I.: A framework for contract-policy matching based on symbolic simulations for securing mobile device application. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 221–236. Springer, Heidelberg (2008)
6. Aktug, I., Naliuka, K.: Conspec – a formal language for policy specification. Electron. Notes Theor. Comput. Sci. 197(1), 45–58 (2008)

7. Bente, I., Dreo, G., Hellmann, B., Heuser, S., Vieweg, J., Von Helden, J., Westhuis, J.: Towards permission-based attestation for the android platform. In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.-R., Sasse, A., Beres, Y. (eds.) TRUST 2011. LNCS, vol. 6740, pp. 108–115. Springer, Heidelberg (2011)
8. Sailer, R., Zhang, X., Jaeger, T.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium, vol. 13, p. 16 (2004)
9. Sailer, R., Jaeger, T., Zhang, X., van Doorn, L.: Attestation-based policy enforcement for remote access. In: CCS 2004: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 308–317. ACM, New York (2004)
10. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: NSPW 2004: Proceedings of the 2004 Workshop on New Security Paradigms, pp. 67–77. ACM, New York (2004)
11. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A., Stüble, C.: A protocol for property-based attestation. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing, pp. 7–16. ACM, New York (2006)
12. Petroni Jr., N., Fraser, T., Walters, A., Arbaugh, W.: An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data. In: Proc. of the 15th USENIX Security Symposium (2006)
13. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: policy-reduced integrity measurement architecture. In: Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies, pp. 19–28. ACM, New York (2006)
14. Zhang, X., Parisi-Presicce, F., Sandhu, R.: Towards remote policy enforcement for runtime protection of mobile code using trusted computing. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 179–195. Springer, Heidelberg (2006)
15. Nauman, M., Khan, S., Zhang, X., Seifert, J.-P.: Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 1–15. Springer, Heidelberg (2010)
16. Bente, I., Dreo, G., Hellmann, B., Heuser, S., Vieweg, J., Von Helden, J., Westhuis, J.: Towards permission-based attestation for the android platform. In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.-R., Sasse, A., Beres, Y. (eds.) TRUST 2011. LNCS, vol. 6740, pp. 108–115. Springer, Heidelberg (2011)
17. Ugus, O., Westhoff, D.: An mtm based watchdog for malware famishment in smartphones. In: Eichler, G., Küpper, A., Schau, V., Fouchal, H., Unger, H. (eds.) IICS. LNI, vol. P-186, pp. 251–262. GI (2011)
18. Joseph, A.: Mobile device management-brave new horizon or basic plumbing? (2013), `http://www.devicemanagement.org/content/view/20754/152/`