Building Efficient Integrity Measurement and Attestation for Mobile Phone Platforms

Xinwen Zhang[1], Onur Acıiçmez[1], and Jean-Pierre Seifert[2]

[1] Samsung Information Systems America, San Jose, CA, USA
{xinwen.z,o.aciicmez}@samsung.com
[2] Deutsche Telekom Laboratories and Technical University of Berlin
jean-pierre.seifert@telekom.de

Abstract. Integrity measurement and attestation mechanisms have already been developed for PC and server platforms, however, porting these technologies directly on mobile and resource-limited devices does not truly satisfy their performance constraints. Therefore, there are ongoing research efforts on mobile-efficient integrity measurement and attestation mechanisms. In this paper we propose a simple and efficient solution for this problem by considering the unique features of mobile phone devices. Our customized secure boot mechanism ensures that a platform can boot to a secure state. During runtime an information flow–based integrity model is leveraged to maintain high integrity status of the system. Our solution satisfies identified security goals of integrity measurement and attestation. We have implemented our solution on a LiMo compatible mobile phone platform.

1 Introduction

Mobile devices such as cellular phones and smartphones have been evolved to be more open and general-purpose so that security has become a significant issue for device manufactures, network and service providers. On one side, today's smartphones typically have increasing processing power, integrated functions, and network connectivity, and hence, there are more services deployed on these devices not only voice but also data, such as messaging, content sharing, and enterprise data processing. On the other side, mobile phone devices face the same kind of threats that have been surging in PC world. According to F-Secure [22], currently there are more than 350 mobile malware in circulation. Examples of the most notorious threats to cellphones include Skull [16], Cabir [2], and Mabir [10] targeting at Symbian operating systems. McAfee's 2008 mobile security report [11] indicates that nearly 14% of global mobile users have been directly infected or have known someone who was infected by a mobile virus, and more than 70% of users expect mobile operators or device manufacturers to preload mobile security functionality.

One major attack mechanism of mobile malware is to maliciously change system files or configurations thus disable some phone or platform functions. For example, mobile viruses like Dampig [4], Fontal [6], Locknut [9], and Skulls [16] maliciously modify system files and configurations thus disable application manager and other legal applications on a phone. Phones can even become unable to uninstall or disable

the malware once they are infected. Doomboot [5] installs corrupted system binaries into c: drive of a Symbian phone, and when the phone re-boots these corrupted binaries are loaded instead of the correct ones, and the phone crashes at boot. Similarly Skulls [16] can break phone services like messaging and camera. For another example, Cardblock [3] deletes system directories and destroys information about installed applications, MMS and SMS messages, phone numbers stored on the phone, and other critical system data. According to F-Secure [22], user installed applications are major threats to mobile platform security, including those downloaded from Internet or received from MMS and Bluetooth.

All these existing attacks compromise the integrity of a mobile phone device. Integrity measurement is a technique to enable a party to query the integrity status of software running on a platform, e.g., through attestation challenges. Several integrity measurement mechanisms have been proposed and developed. Trusted Computing Group (TCG) has published a set of specifications to measure, store, and report hardware and software integrity via transitive trust concept with hardware root-of-trust called Trusted Platform Module (TPM) and Core-Root-of-Trust-Measurement (CRTM). On a TPM-enabled platform, the CRTM measures the bootloader of the system before it is executed, and then stores the measured value into one of the platform configuration registers (PCRs) inside the TPM. The bootloader then loads OS image, measures it and stores via PCR extension, and then executes it [18]. In turn, the OS measures the loaded applications and stores their integrity values in PCRs before executing them. Upon an attestation challenge from a third party, the TPM signs a set of PCR values with an attestation identity key (AIK) and sends back the result. The challenger then can make decisions on the trust status of the platform by verifying the integrity of these values and comparing with the corresponding known-good values. For mobile phone platforms, TCG Mobile Phone Working Group has released specifications for mobile trusted module (MTM) [17].

Problems IBM Integrity Measurement Architecture (IMA) [26] is an implementation of TCG specification in Linux. In IMA, GRUB is augmented to include measurement functions for kernel, ramdisk images, and grub.conf. Also, a kernel module is implemented to measure OS components after the kernel is loaded (post-boot), including libraries, usual command and tool binaries, configuration files, and application modules. Unfortunately, there is no practical implementation of integrity measurement and attestation for mobile platforms so far. One major reason is that, mobile devices such as cellular phones are still limited in computing power, e.g., they typically employ processors running at 200-600 MHz and internal RAM memories of 32MB or 64MB in size. This mandates that any security solution must be very efficient and leave only a tiny footprint in the limited memory. However, IMA has 400-600 measurements on typical desktop and server platforms [7], out of which around 350 measurements are post-boot measurements. The measurements take long time during boot and introduces significant performance delay during runtime. PRIMA [25] extends IMA on mobile phone devices and simplifies the integrity measurements. However it still has more than 200 measurements on an Openmoko phone device [25]. Although many software components (e.g., libs and binary images) on a phone are smaller than corresponding components in PC

and servers, the performance overhead is still very significant and degrades the overall user experience.

Contributions In this paper, we first identify the security goals and requirements for mobile platform integrity measurement. We then propose an efficient solution towards these goals and requirements. Our major design objective is to reduce the number of software components to be measured on a platform, and thus reduces booting and runtime performance overhead and eases the integrity verification for attestation. Our solution consists of two major steps. First, we develop a secure boot mechanism to ensure that a secure kernel is booted, which in turn ensures a well-behaved measurement agent and enforces an authenticated security policy during runtime of the operating system (OS). Runtime integrity assurance is then achieved by leveraging a simple integrity model, which efficiently identifies the *trusted* and *untrusted domains* (including processes and resources) of a mobile platform, and restricts modifications from untrusted domain to trusted side. The essential idea of our integrity measurement is that we only measure integrity protection policy and enforcement mechanism based on our model, including OS kernel and user space components that accept inputs from untrusted domain on a platform (mainly framework daemons providing services to other applications on a platform). For verification, if our model is correctly enforced with carefully designed policy, we have the assurance that processes and resources in trusted domain are protected from untrusted applications, and the platform is always in good integrity status during runtime. The rational that makes our approach practical is that in mobile phone and many embedded devices, it's much easier to identify the board line between trusted and untrusted domains, as we discuss in later sections of this paper. While in PC and server environments, due to many network-faced services and installed software from many unverified resources, it's a complex task.

Outline In the next section we analyze the general security goals of integrity measurement and attestation for mobile platform. We then present our solution with secure boot and runtime integrity model to satisfy these goals in Section 3. We briefly present some implementation information of our approach on a LiMo compatible real smartphone device in Section 4. Section 5 presents some related work on platform integrity measurement. We conclude this paper in Section 6.

2 Security Goals

In this section we explain the security goals of integrity measurement and attestation on general computing environments. We then identify some further requirements for mobile platforms. We note that this is not a complete list of security goals, but the aspects considered in our solution.

Trusted Boot TCG specification requires trusted boot, which adopts the concept similar to AEGIS [19]. Specifically, upon booting, all software components including BIOS and OS bootloader are transitively measured by the CRTM of a platform and the results are extended in the PCRs of TPM. The bootloader then loads OS image, measures it and stores via PCR extension, and then executes it [18]. In turn, the OS can measure applications and stores their integrity values in PCRs before executing them.

Load-time Integrity Measurement There are two different types of integrity measurement for a software binary: load-time and runtime measurements. The TCG only specifies load-time integrity measurement–a piece of code or data is measured when it is mapped or loaded into main memory. In IMA [26], measurements are invoked in several system call functions such as `mmap` and `insmod` when code or kernel modules are loaded but before they are executed. After a code is mapped into memory and during runtime, it is very difficult to measure the integrity of the process considering very dynamic and undeterministic behaviors of typical applications, such as loading active code, receiving external inputs, and allocating dynamic memory.

In addition to trusted boot and load-time integrity measurement, integrity protection for mobile phones has the following extra requirements.

Secure Boot Due to the business model of mobile industry, a mobile device is a service convergence of many remote service stakeholders, including device manufacturer, network operator, and other service providers. There is a set of *mandatory engines* [17] reside on a single mobile platform and provide critical and indispensable services, which have to be running in known-good states, which mean that the integrity of these services must be verified to assure their trustworthiness. Therefore TCG Mobile Phone Reference Architecture [17] states that secure boot is mandatory for MTM. As mentioned, trusted boot requires any component during boot has to be measured and the result is securely stored. Further, secure boot requires that a measured value has to be verified before it is executed. If the measured result does not match a known-good value, the booting is aborted.

Low Booting and Runtime Overhead Most mobile devices such as cellular phones and smartphones are still limited in computing power. Also, user experience is a key business factor in consumer electronic (CE) market. This requires any security solution to be very efficient not to degrade overall user experience. For example, a typical mobile phone should boot in a reasonable time (e.g., less than 10s) and should not have a high overhead when starting a widget or opening the keyboard screen when making a phone call. Therefore low booting and runtime overhead is a pressing requirement for mobile platforms and integrity measurement during boot and in post-boot state should not degrade the performance and user experience too much.

Runtime Integrity Assurance Although runtime integrity measurement is not practical in both PC and mobile platforms, there should be some mechanism to preserve the integrity level of critical applications and resources during runtime, e.g., phone related services (telephony server) and platform management agents. Both TCG and IMA do not propose any mechanism for this purpose.

3 Our Approach

Overview We leverage two mechanisms to achieve these security goals. First of them is our security enhanced bootloader, which measures the kernel image, software TPM module, and our security policy file when the platform boots up. If all these are authenticated, we can assure that the base system (i.e., kernel) is trusted to (1) carry out the rest of the integrity measurements and (2) enforce the security policy after booting.

Our second mechanism relies on a simple integrity model (crafted specificly for mobile platforms) to identify the boundary between trusted and untrusted domains on a typical mobile platform and control the information flow. Security policy implementing our integrity model is enforced in both kernel and user space. The result is that only a few number of binaries and data objects need to be measured in post-boot phase, as they enforce our policy and handle service requests from both trusted and untrusted applications, e.g., most service daemons for platform management, telephony service, and other trusted third party services.

Security Assumptions We do not consider attacks in kernel, such as installing kernel rootkits [1] to bypass our security policy enforcement in kernel space. We further assume that an attacker cannot re-flash the bootloader on a mobile platform, e.g., which can be stored in ROM. That is, our integrity measurement and attestation ensures that a system is secure against software-based attacks, which is the major objective of trusted computing technologies [18].

3.1 Secure Boot

The challenge for secure boot is that there is no hardware implementation of TPM or MTM available for mobile phones so far. Our secure boot leverages a measurement agent in bootloader. For security considerations, we assume that the bootloader and especially this agent is tightly coupled with and protected by hardware, e.g., can be written into ROM. Moreover, a public RSA key of the device manufacturer is encoded in the measurement agent. We assume that the following known-good integrity values (SHA1 value) of the kernel image, software TPM module (swTPM), and security policy are signed by the device manufacturer and the corresponding certificates are stored along with these binaries in the platform. As an alternative, these values can be received from a trusted service provider over-the-air, e.g., from a server and signed by the mobile network operator of the device, in which case the credentials would include the public key certificate of the operator. This enhances the deployment flexibility as the kernel and security policy can be updated after the device is released to customer.

When the platform boots, the measurement agent measures the integrity of the kernel image, the swTPM module, and the binary policy and verifies these values by comparing with those in the certificates. The kernel is executed only after a success verification. The measured integrity values of the kernel image, swTPM, and the policy file are passed to the kernel upon start of its execution. The kernel loads the policy file and starts enforcing the access control rules following our integrity model. Recall that in our architecture, we rely on kernel level mandatory access control mechanism, i.e., the kernel has the ability to fully control the entire information flow in the platform. The kernel also initiates the swTPM and extends the PCRs using the the integrity parameters passed from the bootloader. Through these steps, we ensure that a secure kernel is loaded, an authenticated swTPM is working, and the kernel is trusted to enforce an authenticated security policy during runtime.

[1] According to F-Secure [22], rootkits have not been found on cellphone devices.

3.2 Secure Runtime

Due to very dynamic and undeterministic behaviors of typical applications, runtime integrity measurement is not practical in typical OS environment, if not impossible at all. We propose to leverage an integrity model for runtime integrity preservation. The fundamental idea is, if needed, a trusted application always receives information from other trusted applications or reads data from trusted domains, and untrusted processes cannot write to trusted resources. Through this, the integrity of trusted services and resources is preserved. That is, our model is a runtime information flow control mechanism.

Our model is built on the unique integrity requirements of mobile platforms. Typically, for desktop and server platforms, one of the major security objectives is to protect network-faced applications and services such as httpd, smtpd, ftpd, and samba, which accept unverified inputs from others [23]. As mentioned in Section 1, for mobile phone platforms, on the other side, the major security objective is to protect system integrity threaten by user installed applications, including those downloaded from Internet and received from MMS and Bluetooth.

The architecture of a mobile platform is different from conventional desktop systems. For example, LiMo platform includes a set of frameworks, which provide services to applications via function interfaces. Applications running on a mainstream Linux OS in (e.g.) a desktop communicate directly with kernel to access hardware resources through system call APIs. On the other hand, frameworks control the access and usage of hardware resources on mobile phone devices. For example, mobile phone applications can access SIM data only through the interface APIs of the telephony framework. This architecture is heavily dependent on the business model of mobile and wireless telecommunication industry, where different stakeholders or vendors (device manufactures, network carriers, and third-party service providers) provide variant frameworks and give interfaces to application developers.

A trusted process such as a service daemon may accept requests from both trusted and untrusted applications concurrently. Traditional integrity models are not efficient and flexible under these scenarios. For example, LOMAC [20], UMIP [23], and CW-lite [25] require a process dynamically downgrade its security level whenever it accesses low integrity objects or receives inputs from low integrity processes. However, the process needs to re-start whenever it needs to access high integrity objects later, which is not efficient for mobile devices. Therefore, we propose our integrity model as follows.

Integrity Model Like traditional security models, our model distinguishes subjects and objects in OS. Basically, subjects are *active entities* that can access to objects, which are *passive entities* in a system such as files and sockets. In our model, the set of subjects S are mainly active processes and daemons, the set of objects O include all possible entities that can be accessed by processes, and $S \subseteq O$. In an OS environment, there are many different types of access operations. In our model, for integrity purposes, we focus on three access operations: create, read, and write. From information flow perspective, all other operations can be mapped to these three operations [15].

Table 1 shows the basic integrity rules in our model, where $L(o)$ is the integrity level of object o. Note that by default we consider trusted entities as high integrity and untrusted entities as low integrity in this paper. The first two creating rules are *exclu-*

sively applied upon a single object creation. Typically, the first rule applies to objects that are *privately* created by a process. For example, an application's logs, intermediate and output files are private data of this process. The second rule applies to objects that are *precatively* created by a process upon the request of another process. In one case, $s1$ is a server running as a daemon process, the $s2$ can be any process that leverages the function of the daemon process to create objects, e.g., to create a GPRS session, or save its configuration data with GConf daemon (gconfd). In another case, $s1$ is a common tool or facility program that can be used by $s2$ to create object. In these cases, the integrity level of the created object is corresponding to that of $s2$.

Table 1. Integrity Rules

Policy Rule	Description
r1: $create(s, o)$: $L(o) = L(s)$	object o is created by a process s.
r2: $create(s1, s2, o)$: $L(o) = MIN((L(s1), L(s2))$	object o is created by process $s1$ with input from another process $s2$
r3: $can\_read(s, o)$: $L(s) \leq L(o)$	a process s only can read from an equal or higher integrity process or object o.
r4: $can\_write(s, o)$: $L(s) \geq L(o)$	a high integrity process s can write to an equal or lower integrity process or object o
r5: $can\_read(s1, s2, o)$: $L(s1) \geq L(s2) \wedge can\_write(s1, o) \wedge L(s2) \geq L(o)$	a high integrity process $s1$ can receive information from an equal or lower integrity subject $s2$, provided that the information will be written to lower integrity subject or object o by the high integrity process $s1$.

The r3 and r4 rules indicate that there is no restriction on information flow within trusted entities, and within untrusted entities, respectively, which is, fundamentally, a BIBA-like integrity policy. Rule r5 states that a trusted subject to behave as a communication or service channel between untrusted entities. Therefore not every subject can be trusted for this purpose. Typically, communications between applications and service daemons can be modelled with this rule. For example, on a mobile phone device, a telephony daemon can create a voice conversation between an application and wireless modem, upon the calling of telephony APIs. A low integrity process cannot modify any information of the connection created by a high integrity process, thus preventing stealthily forwarding the conversation to a malicious host, or making the conversation into a conference call.

Boundary of Trusted and Untrusted Domains Many embedded devices (including mobile phones) use read-only filesystems to store static binary images and data for system and application software, which is not a typical characteristics of desktop and server platforms. On a LiMo compatible smartphone [8] that we have evaluated, all Linux system binaries (e.g., `init`, `busybox`), shared libraries (`/lib`, `/usr/lib`), scripts (e.g., `inetd`, `network`, `portmap`), and non-mutable configuration files (`fstab.conf`, `inetd.conf`, `inittab.conf`, `mdev.conf`) are located in a read-only cramfs filesystem. Also, all phone related application binaries, configurations, and framework libraries are located in another cramfs filesystem. All mutable phone related files are located in an ext3 filesystem, including logs, tmp files, database files, GConf configuration resource files, and user-customizable configuration files (e.g., application settings and GUI themes). All user applications can only be installed in a dedicated directory of the ext3 filesystem and the `/mnt/mmc`, which is mounted when a flash memory card is inserted. Based on this layout, we decide that all

objects in user writable filesystem and directories are untrusted, and the others (including read-only and read-write filesystems and directories) are trusted. This is based on the integrity objective of our solution – to protect system and service components from user installed applications. We have observed very similar filesystem layout on many other Linux mobile phones, including Motorola A1200 [14] and Android [1].

We note that above approach to determine trusted and untrusted domains is just an example mechanism. The general idea of our model is to decide the boundary between platform system and service domains, and user installed application domains, via different filesystems and/or directories. Note that, filesystem layout is determined by the device manufacturer of a platform and such a separation can be enforced quite easily. This approach simplifies policy definitions and reduces runtime overhead compared to traditional approaches such as SELinux on PCs, which sets the trust boundaries based on individual files.

Integrity-aware IPC Our integrity-aware IPC aims to control information flow between subjects. Most IPC objects (including domain sockets, pipes, fifo, message queues, shared memory, and shared files) inherit their integrity levels from the processes that create them. Therefore, our model restricts any access where a high integrity process tries to receive data through an IPC object created by a low integrity process.

In many mobile Linux platforms such as LiMo, OpenMoko, GPE, Maemo, and Qtopia, D-Bus is the major IPC, which is a message-based communication mechanism between processes. A process builds a connection with a system- or user-wide D-Bus daemon (dbusd). When this process wants to communicate with another process, it sends messages to dbusd via this connection. The dbusd maintains all such connections (from many different processes), and routes messages between them. A D-Bus message is an object in our model, which inherits integrity level from its creating process. According to r5 of our model, trusted subject dbusd can receive any D-Bus message (low or high integrity level) and forward to corresponding destination process. Typically, a trusted process can only receive high integrity messages from dbusd. According to r5 of our model, if a process is a trusted daemon, like telephony service or GConf daemon, it can receive high and low integrity messages from dbusd, and handle them separately within the daemon.

Program Installation and Launching Application installation is usually performed by a dedicated installation program called installer. As the installer is trusted, it can read both high and low integrity application packages according to our model. Again according to our model, it can write (i.e., install) to trusted part of the filesystem if it reads the data from a high integrity software package, and can write to untrusted part of the filesystem if the data is from a low integrity software package. Similar to installation, during the runtime of a mobile system, a process is invoked by a trusted program called program launcher, and both high and low integrity processes can be invoked by the program launcher. All processes invoked from trusted program files are in high integrity level, and all processes invoked from untrusted program files are in low integrity level.

Mobile phones can be infected with malware via variant communication channels between phone devices and networks. For example, many malware in Symbian-based

phones send malicious codes via Bluetooth channel, or distribute with MMS messages (either in message contents or as attachments). Therefore in our model we treat any code received from these network-faced applications as untrusted by default. Thus, according to our model, any process invoked from arbitrary code by MMS agent or browser is in low integrity level and cannot write to trusted resources and services, such as corrupting system binaries or changing platform configurations. It is possible that some software received from Bluetooth/MMS/browser can be trusted. For instance, a user can download a trusted software from his PC via Bluetooth or from a trusted service provider's website via browser. Therefore it can be installed to the trusted side on read-write filesystem by the application manager on the phone, after the user explicitly indicates or confirms that this application is trusted, or with a source verification (e.g., via digital signature) of the software package.

3.3 Putting Together: Integrity Measurement and Protection

After introducing our secure boot mechanism and runtime integrity model, it is time explain how to measure and verify a mobile platform integrity. The overall platform architecture and measurements are show in Figure 1. As aforementioned, with secure boot, a mobile platform can boot to a state with an authenticated kernel, which in turn is trusted to enforce an authenticated policy based on our model. The integrity values of kernel image, swTPM and the policy are stored in PCRs of the swTPM, e.g., for attestation purposes after booting. After this, a measurement agent in the kernel and the swTPM can perform secure measurements and storage for other modules and trusted subjects during runtime. Specifically, when a measurement target is to be loaded, the measurement agent captures this event, measures its integrity, extends a corresponding PCR of the swTPM, and then loads the target into main memory and transfers its execution to it.

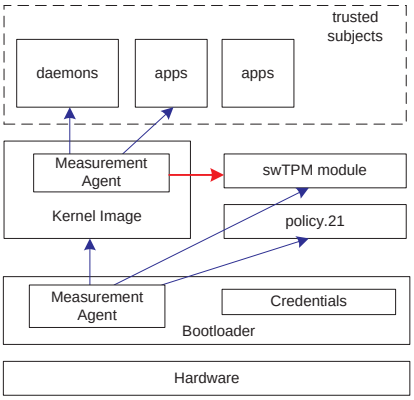

Fig. 1. Platform architecture: integrity measurement and verification with secure boot and runtime integrity enforcement.

During runtime, our objective is to minimize the number of components to be measured for performance reasons. Firstly, according to our integrity model, when the kernel is genuine, we have the assurance that all code and data in read-only filesystems cannot be altered by any means. Secondly, for any code that is located in a read-write filesystem and regarded as trusted, we check whether the code is an executable binary of a daemon, which is a trusted subject and enforces security policy in our model. If it is, we need to measure it. If the code is not a trusted daemon, we do not need to measure it, as its runtime integrity is preserved with the integrity policy rules based on our model, and the policy is honestly enforced by the kernel and daemons. We do not need to measure any other untrusted code and data, because they are already regarded as untrusted subjects and objects in our model and cannot alter any trusted entities due to our model and policy enforcement mechanisms explained above. Similar to IBM IMA [26], we do not measure dynamic data and configuration files.

The attestation of our platform is similar to traditional TCG-based approach. With an attestation service agent on the platform (not shown in Figure 1), when receiving an attestation challenge, the agent responses with PCR values signed by the swTPM. Due to space limitation, we ignore the details of these steps and also the procedure to obtain an attestation identity key for the platform.

4 Implementation

We have implemented our model on a real LiMo platform [8]. We augmented the bootloader (u-boot) on this platform with a simple integrity measurement agent. We ported the TPM emulator [27] on this platform. We are also porting the software MTM module [12] to this platform.

The integrity model is implemented with SELinux, which provides comprehensive security checks via Linux Security Module (LSM) in kernel. SELinux provides domain-type and role-based policy specifications, which can be used to define policy rules to implement high level security models. Actually, current SELinux does not have an integrity model built-in. On one side, our implementation simplifies SELinux policy for mobile phone devices by leveraging trusted and untrusted subject attributes. On the other side, our implementation augments SELinux policy with our integrity model.

D-Bus is the major IPC mechanism for most Linux-based mobile platforms. Following our integrity model, we extend D-Bus implementation in two aspects. First, each message is augmented with a header field to specify its integrity level based on the process which sends the message, and the value of this field is set by dbusd when it receives the message and before dispatches it. Secondly, according to our integrity model, if a destination bus name is a trusted daemon process, it can accept both high and low integrity messages; otherwise, it only accepts messages with the same integrity level as itself. We also have augmented other framework daemons of this LiMo platform to enforce similar integrity policy, including the GConf and telephony server, as they can access both trusted and untrusted messages via D-Bus.

We define two domain attributes to specify high and low integrity processes: *trusted* and *untrusted*, respectively. We use generalized filesystem labeling mechanism [13] in SELinux to label both cramfs and ext3 filesystems. Our policy size is less than 20KB

including `genfscon` rules for filesystem labelling. Comparing to that in typical desktop Linux distributions such as Fedora Core 6 (which has 1.2MB policy file), our policy footprint is tiny. We also studied the performance of our runtime security enforcement with micro benchmark. Our results show that for most operations, our security enforcement has less than 4% overhead, which is significantly less than the counterpart technology on PC [24].

5 Related Work

As aforementioned, IBM IMA [26, 7] is the integrity measurement solution for PC platforms. However, directly porting this to mobile devices is not practical due to its high computation overhead during booting and runtime. PRIMA [25] leverages the CW-lite information flow control to maintain a process's integrity, where particular interfaces of the process filter low integrity information when received by this process. However, identifying filtering interfaces in many service processes (daemons) on a mobile phone is not a easy task, especially many of them come from different software vendors, e.g., network carrier, device manufacturer, and third party service providers. Also, PRIMA still has more than 200 measurements on an Openmoko phone device [25].

Dynamic root-of-trust-for-measurement (DRTM) [21] is a mechanism on x86 platform to execute a piece of code in hardware protected trusted environment during runtime and without physically rebooting the platform. However this is not practical in mobile device. First of all, ARM processors have not built in this capability. Secondly, only 32KB size of code can be executed in the trusted environment in DRTM, which shifts the integrity measurement problem to other mechanism after that code is launched.

Runtime integrity measurement has been proposed recently with copy-on-write mechanism [28]. However, this is implemented on Xen-like virtualization environment, which is so far not practical for mobile platforms.

6 Conclusion

Towards the protection of mobile platform integrity threaten from untrusted user applications, we propose a simple and efficient solution for integrity measurement and attestation. Our solution uses a secure bootloader to measure the kernel and a software TPM, which ensures that the platform can boot to a secure state. After booting, we leverage a simple integrity model to preserve the runtime integrity of the system. Our model easily distinguishes trusted and untrusted domains on both filesytem and memory space, thus making the policy development very simple and verifiable. During runtime, we measure the enforcement mechanism of security policy based on our integrity model, i.e., trusted daemon processes, thus significantly reduce the number of components to be measured. We are developing a formal specification and security property verification of our proposed integrity model.

References

1. Android, http://code.google.com/android/.
2. Cabir, http://www.f-secure.com/v-descs/cabir.shtml.
3. Cardblock, http://www.f-secure.com/v-descs/cardblock_a.shtml.
4. Dampig, http://www.f-secure.com/v-descs/dampig_a.shtml.
5. Doomboot, http://www.f-secure.com/v-descs/doomboot_a.shtml.
6. Fontal, http://www.f-secure.com/v-descs/fontal_a.shtml.
7. IBM integrity measurement architecture, http://domino.research.ibm.com/comm/research_projects.nsf/pages/ssd_ima.index.html.
8. Limo Foundation, http://www.limofoundation.org/en/technical-documents.html.
9. Locknut, http://www.f-secure.com/v-descs/locknut_e.shtml.
10. Mabir, http://www.f-secure.com/v-descs/mabir.shtml.
11. Mcafee Mobile Security Report 2008, http://www.mcafee.com/us/research/mobile_security_report_2008.html.
12. MTM Emulator, http://hemviken.fi/mtm/.
13. NSA Security-Enhanced Linux Example Policy. http://www.nsa.gov/selinux/.
14. OpenEZX, http://wiki.openezx.org/main_page.
15. Setools–policy analysis tools for selinux, http://oss.tresys.com/projects/setools.
16. Skulls, http://www.f-secure.com/v-descs/skulls.shtml.
17. TCG Mobile Reference Architecture Specification Version 1.0, https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-reference-architecture-1.0.pdf.
18. TCG TPM Main Part 1 Design Principles Specification Version 1.2, https://www.trustedcomputinggroup.org.
19. Arbaugh, W. A., Farber, D. J., Smith, J. M.: A secure and reliable bootstrap architecture. In: *Proc. of IEEE Conference on Security and Privacy* (1997), pp. 65–71.
20. Fraser, T.: LOMAC: MAC you can live with. In: *Proc. of the 2001 Usenix Annual Technical Conference* (2001).
21. Grawrock, D.: *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*. Intel Press, 2006.
22. Hypponen, M.: State of cell phone malware in 2007, http://www.usenix.org/events/sec07/tech/hypponen.pdf.
23. Li, N., Mao, Z., Chen, H.: Usable mandatory integrity protections for operating systems. In: *Proc. of IEEE Symposium on Security and Privacy* (2007).
24. Loscocco, P., Smalley, S.: Integrating flexible support for security policies into the linux operating system. In: *Proc. of USENIX Annual Technical Conference* (June 25-30 2001), pp. 29 – 42.
25. Muthukumaran, D., Sawani, A., Schiffman, J., Jung, B. M., Jaeger, T.: Measuring integrity on mobile phone systems. In: *Proc. of the 13th ACM Symposium on Access Control Models and Technologies* (2008).
26. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: *USENIX Security Symposium* (2004).
27. Strasser, M.: Software-based TPM emulator for linux. Semester Thesis, Department of Computer Science, Swiss Federal Institute of Technology Zurich, 2004.
28. Thober, M., Pendergrass, J. A., McDonell, C. D.: Improving coherency of runtime integrity measurement. In: *Proc. of the 3rd ACM workshop on Scalable Trusted Computing* (2008).