Volume 4, Issue 8, August 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Improving Integrity Protection for Open Mobile Platforms

T. Lakshmi Sravanthi
Department of Computer Science and Engg.
Swetha Institute of Technology & Sciences,
Tirupati, India

Mr. P. Nageswara Rao
Department of Computer Science and Engg.
Swetha Institute of Technology & Sciences,
Tirupati, India

Abstract-Now a days, the cellular phones and smart phones are increasing usage in people's daily life. So with this advantage the security of the mobile devices is a big challenge and gained more importance. The problem is very challenging as the working environment of these Mobile devices has become more general purpose and Open working environment at the same time we have to concentrate on performance, Scalability and efficiency. Apart from this We propose and implement SEIP, a simple and efficient effective solution of integrity protection of open cellular phone platforms, which is notified by the disadvantage of applying traditional integrity protection of the real world cellular phone platform models. With the increase use of the SEIP, significantly simplifies policy specifications while still achieves a good assurance of platform integrity. SEIP is executing within a commercially available Linux-based smartphone and illustrates that it can efficiently prevent certain Bugs and Malwares. The security policy of our implementation is less than 40 kB, and we will do a analysis, so that enhanced performance study shows that its weight is less and called it as Lightweight.

Keywords - Integrity protection, Efficiency, SEIP, open mobile platforms, Smartphone security.

I. INTRODUCTION

Generally with the increasing computing throughput, Scalability and connectivity of network across various mobile devices such as cellular phones and smart phones, more services are executed on these Mobile open platforms. Thus, their computing environments become more open than ever before. The security issue in these environments has gained more attention in these days. According to McAfee's 2006 Mobile Security Report, approximately 20 percent of global mobile users have been infected directly by a mobile virus. About 86 percent of consumers worry about receiving unauthorized and useless content so that information loss and theft will happen, and about more than 70 percent of users expect mobile operators or manufacturers to preload mobile security software. The number of infected mobile devices are increasing more and more daily according to McAfee's 2009 report. Third, security functionality should require minimum or zero number of interactions from a particular mobile user, for example, the end user should not be required to configure a user's security policy. This tell's then that the solution must very simple but more general enough so that almost all users can depend on default softwares even after the development new application which is installed. According to F-secure survey, by the end of year 2008, about 400 different Viruses has been found on various Mobie phones, including Worms, Trojans, and Malwares. Most of the existing Problems are due to user downloaded applications, such as images, games and Videos. Other major Problem mechanisms include Infrared, Bluetooth message transfer and multimedia message service (MMS). Many attackers attack on the Security of a mobile platform by unnecessarily modifying data or code on the particular mobile device. We will Consider the numerous number of attacks which will occur through infrared, Bluetooth and MMS interfaces. We design an effective integrity protection mechanism which should define the interactions between any code or data received from mobile communication devices and remaining parts.

We Propose an Simple, effective and efficient solution, called as SEIP, a mandatory access control based integrity and security protection mechanism for Open mobile phone Platforms. Our Proposed System is based upon acess and flow control Systems between trusted and untrusted domains. By confining untrusted applications' write operations to trusted domains. However, there is no Security model behind this mechanism and it is difficult to have the guarantee and verify if a system is running in a finite good integrity state. Second, many Assured and Trusted process on a mobile device provides Services to both trusted and untrusted applications, mainly the unified services such as Mobile telephony Client-server, effective messaging services, interprocess communications, and application configuration service.

- We analyse integrity threats on mobile platforms based on infection and compromising mechanisms. We then identify salient requirements for security solutions on mobile terminals and propose our strategies toward these requirements.
- Based on different functional behaviours of subjects in mobile systems, Here, we define and Implement a set of integrity protection rules to maintain their interactions with other Devices. Our solution focuses on the protection of phone and platform management services from untrusted applications such as those downloaded by users or received from Bluetooth/MMS.

- We Propose the design and implementation of our solution in a Linux-based Open mobile phone device, and we use Tiny Core Linux to define a respective policy. Our policy size is less than 40 kB in binary form and requires less than 15 domains and its types.

II. VULNERABILITY MODEL

We mainly study on the security and protection of integrity in open mobile platforms. we also study various advanced model's of mobile threats from two aspects: integrity assets and attack mechanisms. Note that, in this paper, we consider attacks from application level.

2.1 Assets for Platform Integrity

Resources of network service provider. A mobile device usually consists of sensitive data from network service provider, such as those stored in SIM card for network and service profiles. Unauthorized access to these data can compromise the running behaviour of the device and communications between the device and wireless network.

Device data and status settings. Modern mobile devices are employed with many sensors, such as timer, GPS, touch screen, and webcam. Manipulating these sensors without authorization from the user can cause unexpected behaviour of a device. Also, a device provides many status setting functions such as those for 3G, WiFi, Bluetooth, screen brightness level, and battery.

Resources of mobile user. A user stores many personal data on device, such as messages, address book, and online credentials. Many online service providers store user data on device side, such as online bank and entertainment services, which are targets for malware. By sending SMS/ MMS messages and making hidden phone calls to premium phone numbers, a malware can generate monetary cost to a mobile user.

We give the idea of some example malware and their infection mechanisms and target integrity assets.

The Mobile Malwares and their Behaviours

- DAmpig, Fontal, Locknut these are infected by Bluetooth, MMS, internet and modify system files and configurations, disable application manager and phone services.
- Cabir,CommWarrior,Mabir malware's propagation through Bluetooth, MMS which scan new devices with Bluetooth,sends user data and malicious code to new targets without authorization.
- Doomboot is affected with Bluetooth, MMS and blocks access to memory card, delete system files and installed application files and data.
- Skulls is by Bluetooth and blocks access to memory card, delete system files and installed application files and data.
- Redbrowser and Mquito malwares is by downloading java applications which sends SMS messages to premium rate number at a rate of 30 and 40 rupees per message.

2.2 Attach Mechanisms

Various Infection mechanisms exist in Open Mobile Platforms. With various constraints of computing mechanisms, bandwidth of the network and Input /Output the major application area of mobile devices is for usage of mobile data and application services from different Mobile service providers, instead of providing useful data and multiple services to others. For typical client-server platforms, one of the major integrity objectives is to protect multiple network-faced applications and services such as HTTP, SMTP and FTP which mainly accepts multiple unverified inputs from other devices . For all Open mobile phone platforms, the major objective of this security architecture are to protect system integrity that is threatened by user downloaded and installed applications. According to mobile security reports from AVG and NORTON, most existing malware attacks on cell phones are unintentionally downloaded and installed by user and so far all the worms does not require any user interaction for spreading of viruses across various mobile platforms .

Although almost all phones does not have Internet-related applications and services, many phones have standard and low bandwidth communication services such as file-sharing via infrared and Bluetooth. Although, There is increasing demand towards multimedia messaging service (MMS). Major malwares and Trojans have been found in Symbian phones which spread through Bluetooth and/or MMS such as Cabir, CommWarrior, and Mabir. Hence any information or unnecessary code received through this services should be called as untrusted and we cannot trust the application. when a user explicitly prompts to trust it then only we can trust it otherwise we cannot trust the application. We will follow the same consideration for any data which is received via browsers on mobile devices.

Various Integrity compromising mechanisms exist in open mobile applications. Many mobile malware's consists of platform security by disabling multiple platform functions. For example, once installed mobile viruses like Dampig, Fontal, Locknut, and Skulls maliciously modify multiple system files and their configuration thus disable application manager and other legal applications.

III. SEIP OVERVIEW

3.1 Requirements

Simplicity and effectiveness. It is very expensive to change any system function on a mobile device once it is massively produced and sold to final customers. Also, a mobile user typically does not have skills and knowledge to configure any

security functions except some simple application-level settings. On the other side, due to the complexity of mobile business model, i.e., hardware and software components can be integrated from different vendors, complex security policies and mechanisms can increase the cost of system integration and deployment.

Efficiency. One critical performance requirement lies on typical usage behaviours with mobile devices. A mobile user is not so "sticky" as that in desktop environments, where she spends a lot of time using a particular application. While a mobile user is "bouncy," and flies in and back out between different applications. Therefore, a trusted subject (e.g., a service daemon) may accept requests from both trusted and untrusted application concurrently (although may not be absolutely concurrent). Traditional integrity models are not efficient and flexible under these scenarios.

3.2 Design Overview

To achieve effective integrity protection mechanism and its goals with respect to the constraints of mobile computing environments, we propose the following techniques in our design methodology.

The boundaries between the trusted and untrusted domains are very clearly defined in SEIP. We will not consider fine-grained privileges for individual applications, we mainly focus on security of trusted Systems. With this principle, we identify system and its boundary's along with its related simple file's and its layout in many Linux-based mobile phones and in its server environments. Specifically, based on our Research, most Mobile phone-related services from multiple device manufacture's and network service providers are deployed on dedicated file system. but user may download many applications which can only be installed on another system directory, or memory card. Thus, for example, one mechanism can specify that by default all the applications belonging to the particular manufacturer or service provider are trusted for integrity purpose, while user installed applications are untrusted. An un-trusted application can be changed to trust one only through additional authentication mechanisms or explicit authorizations from multiple user's.

Toward this issue, we propose that some particular trusted processes can accept untrusted data while maintaining their integrity and security level's. The important requirement here is that accepted unauthorized information does not affect the behaviour of such trusted process. We distinguish different integrity levels via separating the information received from multiple users. For example, the telephony server accepts requests from both trusted and untrusted applications, while enforces constraints to the types of phone calls that an untrusted application can make. For another example, during the installation and launching of applications on mobile platforms, untrusted applications can be installed and launched through a trusted package management tool and application launcher (which of course operate on both trusted and untrusted domains), while the installer and launcher still maintain their respective integ-rity.

IV. DESIGN OF SEIP

This section presents design details and integrity rules of SEIP for mobile platform based on discussed security threats and our strategies. Although we describe within the context of Linux-based mobile systems (specifically, LiMo platform), our approach can be applied to other phone systems such as Symbian, as they have similar internal software architecture. One assumption is that we do not consider attacks in kernel and hardware, such as installing kernel rootkits or reflashing unauthentic kernel and filesystem images to devices. That is, our goal is to prevent software-based attacks from application level.

4.1. Trusted and Untrusted Domains

To preserve the integrity of a mobile device, we need to identify the integrity level of applications and resources. In mobile platforms, typically trusted applications such as those from device manufacture and wireless network provider are more carefully designed and tested as they provide system and network services to other applications. Thus, in our model, we regard them as high-integrity applications or subjects. Note that completely verifying the trustworthiness of a high-integrity subject, for example, via static code analysis, is out of the scope of SEIP. As aforementioned, our major objective is to prevent platform integrity compromising from user installed applications. Therefore, by default all user installed applications later on the platform are regarded as low integrity. In some cases, a user installed application should be regarded as high integrity, example, if it is provided by the network carrier or trusted service provider and requires sensitive operations such as accessing SIM or user data, for example, for mobile bank and payment applications. For an applica-tion belonging to third-party service provider, its integrity level may be based on the trust agreement between the service provider and user or manufacturer/network provi-der. For example, an antivirus agent on a smartphone from a trusted service provider needs to access many files and data of the user and network provider and should be protected from modification of low-integrity software; therefore, it is regarded as high integrity. Other high-integrity applications can be trusted platform management agents such as device lock, certificate management, and embedded firewall.

4.2. Subjects and Objects:

The design distinguishes both the objects and subjects in the Operating System. Basically, objects are Passive entities that can access by subjects, which are active entities in a system such as sockets and files. Subjects are mainly active processes and daemons, and objects include all possible entities that can be accessed by processes, such as files, directories, file systems, network objects, program and data files. Note that a subject can also be an object as it can be accessed by another process, for example, being launched or killed. In an OS environment, there are many different types of access operations. For example, SELinux predefines a set of object classes and their operations. For integrity purposes,

we focus on three major operations: create, open ,write and Read. From information flow perspective, all the access operations between two existing entities can be mapped to read-like and write-like operations.

The network manager framework in LiMo creates and maintains all network connections and profiles for different applications, such as packet data protocol (PDP) sessions for GPRS and access point associations for WiFi connections. Another example, Gconf daemon (gconfd) stores configuration data for individual phone applications, which can only access their data via GConf APIs, and gconfd is the only subject that can physically (in OS point of view) read and write the objects. Not only for those objects in regular OS such as files and sockets, our design protects objects internally maintained by these service daemons which affect the integrity of a platform.

Rules for Information Flow Control:

Rule 1: Create (s,o)⟵L(o)=L(s): when object O is created by a process S ,O inherits S's integrity level.
Rule 2: Create (s_1,s_2,o) ⟵L(o)=MIN((L(s_1),L(s_2)): When o is an created by the process s_1 with input from another process s_2, o inherits the lower bound of integrity levels of s_1 and s_2.
Rule 3: can_read(s,o) ⟵ L(s)<=L(o): a low integrity process s can read from a low or high integrity process or object o. likewise all the six rules are done until a high integrity process s reads low integrity object o, it's integrity level is changed to L(o).

4.3. Trusted Subjects:

In this project it is distinguished three types of trusted subjects[2] on mobile platforms, according to their functionalities and behaviors. Different integrity rules are applied to them for integrity protection purpose.

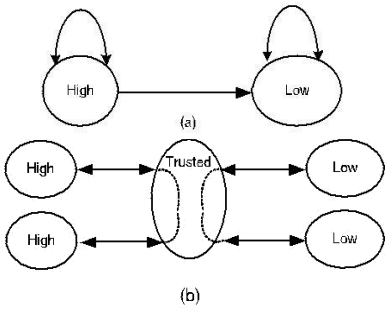

Fig. Information flows are allowed between high- and low-integrity entities, directly or indirectly via trusted subjects

Type I trusted subjects. This type includes high-integrity system processes and services such as init and busybox, which are basically the trusted computing base (TCB) of the system. Only high-integrity subjects can have information flow to those subjects, while un-trusted subjects can only read from them. Type I trusted subjects also include preinstalled applications from device manufacture or service provider, such as dialer, calendar, clock, calculator, contact manager, and so on. As they usually only interact with other high-integrity subjects and objects, their integrity level is constant during runtime.

Type II trusted subjects. These are applications provided by trusted resources, but usually read low-integrity data only, such as browser, MMS agent, and media player. They are usually predeployed in many smartphones by default, However, they mostly read untrusted Internet content or play downloaded media files in flash memory card. These subjects usually do not communicate with other high-integrity subjects in most current smartphone systems, and they do not write to objects which should be read by other trusted subjects. Therefore, in our design, we downgrade their integrity level during runtime without affecting their functions and system performance.

Type III trusted subjects. These are mainly service daemons such as telephony, message, network manager, interprocess communication (IPC), device status manager (reading and setting hardware status), and application and platform configuration services. Usually these subjects need to interact with both low- and high-integrity subjects.

Dealing with IPC:

A low-integrity process creates an IPC object and writes to it, a high-integrity process cannot read from it, according to our integrity rules. In many Open Linux mobile platforms such as LIMO, OpenMoko, GPE, Maemo, and Qtopia, D-Bus is the major IPC, which is a message-based communication mechanism between Various processes. A process builds a connection with another system or user vie D-Bus daemon .When a particular process wants to communicate to anther process, it sends messages to D-bus via its connection. The D-bus maintains connections between many processes and routes between them. A D-Bus message is an object in our design, which inherits integrity and security levels from its process created.

4.4. Program Installation and Launching*:*

An application to be installed is packaged according to particular format, i.e., .SIS file for Symbian and .ipk for many Linux-based phone systems, and application installer reads the program package and metadata and copies the program

files into different locations in local filesystem. As the application installer is a Type III trusted subject specified by policy, it can read application packages which are both high-level and low-level integrity. Also, according to our integrity Rule 2 and 5, it writes (when installing) to trusted part of the file system when reads high-integrity software package, and writes to untrusted part of the file system when reads low-integrity package.

On one aspect, this enhances the security as a malicious application cannot be launched to a privileged process, which is a major vulnerability in traditional OS; on the other aspect, this simplifies policy specification in a real system, which can be seen in next section.

4.5. Dealing with Bluetooth/MMS/Browser and Their Received Code/Data:

With the wide increasing malware infect mobile phones via variant communication channels between Mobile devices and their Service networks. For example, many Viruses in Symbian-based mobile phones which sends malicious codes via Infrared, Bluetooth channel, and distribute with MMS messages. While in the mean time more smartphones have been deployed with Internet browsers, which is another interface to access and receive untrusted code and data. SEIP regards MMS agents and mobile browsers as Type II trusted subjects via security policy. So their integrity level is changed to low whenever they receive data from outside, for example, reading message or browsing web content. Any code or data received from Bluetooth, MMS, and browser is un-trusted.

V. IMPLEMENTATION

We have implemented SEIP on a real LiMo platform. Our implementation is built on GNULinux, which is done thru sophisticated security checks via GNU Comprehensive Linux security module (LSM) in kernel. Also GNU and SELinux provides role-based and domain-type specification policies, which can be used to define policy rules to implement Maximum high-level security policies and its models. However, existing deployments of SELinux on desktop and servers have very complex security policies and usually involve heavy administrative task. Furthermore, current SELinux does not have an already built-in integrity model.Our implementation simplifies SELinux policy for mobile phone devices which totally depends on SEIP, which we have implemented. SEIP has built-in integrity considerations which auguments the implementation.

5.1 Trusted and Untrusted Domains

All Linux system binaries like init has shared libraries , scripts, and non mutable configuration files like (fstab. conf, init tab.conf, mdev. conf) are located in a read-only files which are stored in file system. Also, all the Mobile phone applications its configurations, and framework libraries are located in another file system. All mutable phone related files are located in an extension3 file system, including log files, temporary files, database files, application files, and user-customizable configuration files.

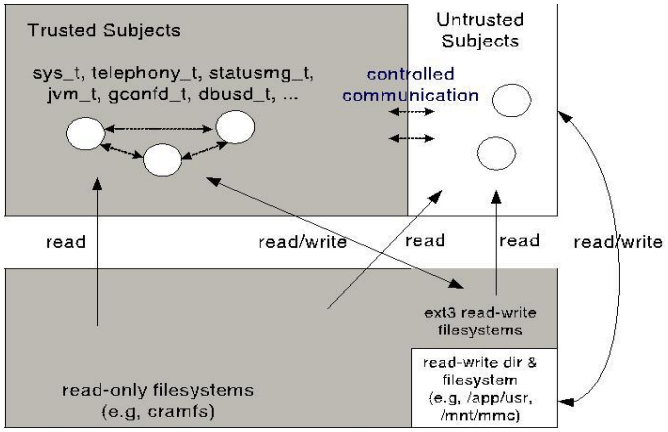

Fig. 2. Trusted and untrusted domains and allowed information flow between them on evaluation platform

all read-only filesystems and part of ext3 filesystem where phone related files are located are regarded as trusted, and user writable filesystems are regarded as untrusted. By default, processes launched from trusted filesystems are trusted subjects, and processes launched from untrusted filesystems are untrusted subjects. Note that our approach does not prevent trusted user application from being installed on the device. For example, a trusted mobile banking application can be installed in the trusted read-write filesystem, and the process launched from it is labeled as trusted.

5.2.Securing Phone Services

The telephony server provides services to typical phone-related functions such as voice call, data network (GSM or UMTS), SIM access, messages (SMS and MMS), and GPS. An application calls telephony APIs (TAPI) to access services provided by the telephony server, which in turn connects to the wireless modem of the device to build communication channels. In our LiMo platform, message framework and data network framework are dedicated for short message and data network access services. An application first talks to these framework severs which in turn talk to the telephony server. Security controls for those services can be implemented in their daemons.

Different levels of protection can be implemented for secure voice call. For example, one policy allows that only trusted applications can make phone calls, while untrusted application cannot make any phone call, which is the case in many feature phones. For another example policy, un-trusted applications can make usual phone calls but not those of premium services such as payment-per-minute 900 numbers. Different labels can be defined for telephone numbers or their patterns. In our implementation, we allow untrusted applications to call 800 toll-free numbers only. Similar design is used in message framework.

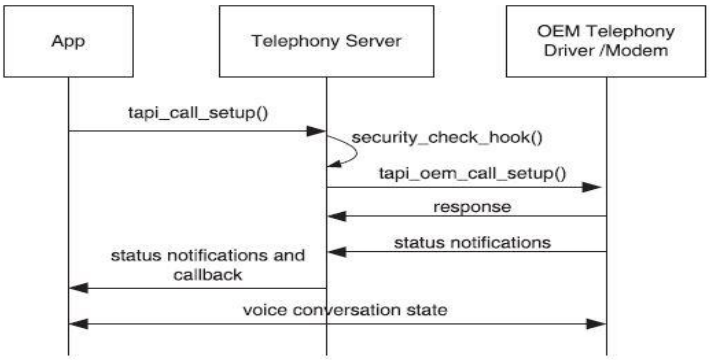

Fig 3. Secure telephony server.

Fig. 3 shows the workflow for a typical voice call. A client application calls tapi_call_setup() to initialize a phone call with TelCallSetupParams t, which includes the target phone number and type (voice call, data call, or emergency call), and a callback function to handle possible results. The telephony server provides intermediate notifications including modem and connection status to the client. Once the call is established with the modem, the telephony server sends the connected indication to the TAPI library which in turn notifies the application via the registered callback function about the status of call (connected or disconnected), and then the application handles the processing.

5.3. Securing Device and System Services

The system framework in our platform maintains all system status such as phone status, device status, memory status for out-of-memory, audio path, and volume status, and provides get/set APIs for accessing them. A status manager (a daemon) in this framework collects global system status from other frameworks, such as phone status (busy/idle/ standby) from telephony server, power levels from power manager in kernel, and device status from various kernel device drivers. Although typically every application can get the statuses via get APIs, only trusted applications can set these variables via set APIs.

Table Telephony And Message Sever Apis Integrity Check Is Enforced

Telephony Service		
Call_setup	Call_answer	Call_release
SendDTMF	Call_releaseall	Hold
Retrieve	ECT	Join
Split	Call_forward	Call_wait
Call_bar	SetRestrictCallId	
Short Message Service		
Sms_write	Sms_delete	Sms_send
Sms_setCbconfig	Sms_setPreferredbearer	Sms_setSca
Sms_setSMSParameters	Sms_SendDeliveryReport	
Access SIM Resources		
Sim_changePin	Sim_unblockPin	Sim_EnablePin
Sim_DisablePin	Sim_update	Sim_EnableFDN
Sim_DisableFDN	Sim_UpdatePBRecord	Sim_deletePBRecord

Table list some subjects of each type. The development of policy rules follows two principles: maximally enable permissions between trusted domains, and minimally enable permissions between trusted and untrusted domains. Specifically, regular per-missions are enabled between trustedI and trustedIII domains such as those defined in permission classes of process, socket, and netlinks. For untrusted domains and objects, by default we enable all read-like permissions and disable all write-like operations from untrusted domains to trusted side.

VI. EVALUATION

6.1 Performance Evaluation

Our policy size is less than 40 KB including genfscon rules for filesystem labeling. Comparing to that in typical desktop Linux distributions such as Fedora6, our policy specification footprint is small. As aforementioned, the small footprint of our security mechan-ism is result from the simple way to identify the borderline between trusted and untrusted domains, and the efficient way to control their communications.

We study the performance of our SELinux-based im-plementation with microbenchmark to investigate the over-head for various low-level system operations such as process, file, and socket accesses. Our benchmark tests are performed with the LMbench 3 suites.

VII. LIMITATIONS

Although we have implemented our design in some major services of our evaluation platform including IPC (D-Bus), telephony, device status manager, and system configuration service, obviously this is not a complete list for a whole platform. due to lack of source code, we do not have implementation on data network service and message service. In general, the framework services of a mobile phone device can be provided by many different vendors, such that a complete implementation so far is not feasible in our prototype. One of our design goals is to ease the integration of security between functional frameworks. Typically, a framework provider just needs to identify sensitive functions or APIs that need to be controlled for integrity purpose, declare a set of corresponding permission names, and insert a common security hook function into the API implementations of the service functions, which is implemented in a trusted library based on our integrity rules. We believe this significantly releases the burden of security considerations for system framework developers.

VIII. RELATED WORK

Information flow-based integrity models have been pro-posed and implemented in many different systems, includ-ing the well-known Biba, Clark-Wilson, and LOMAC. Biba integrity property restricts that a high-integrity process cannot read lower integrity data, execute lower integrity programs, or obtain lower integrity data in any other manner. In practices, there are many cases that a high-integrity process needs to read low-integrity data or receive messages from low-level integrity processes. LOMAC supports high-integrity process's reading low-integrity data, while downgrading the process's integrity level to the lowest integrity level it has ever read. PRIMA and UMIP dynamically downgrade a process's integrity level when it reads untrusted data. As a program may need to read and write to high-integrity data or communicate to high-integrity subjects after it reads low-integrity data, it needs to be relaunched by a privileged subject or user to switch to high level. Although these approaches can achieve a platform's integrity status, they are not efficient for always-running service daemons on mobile device.

IX. CONCLUSION

In this paper, we present a simple but yet effective and efficient security solution for integrity protection on mobile phone devices. Our design captures the major threats from user downloaded or unintentionally installed applications, including codes and data received from Bluetooth, MMS, and browser. We propose a set of integrity rules to control information flows according to different types of subjects in typical mobile systems. Based on easy ways to distinguish trusted and untrusted data and codes, our solution enables very simple security policy development. We have im-plemented our design on a LiMo platform and demon-strated its effectiveness by preventing a set of attacks. The performance study shows that our solution is efficient by comparing to the counterpart technology on desktop environments. We plan to port our implementation to other Linux-based platforms and develop an intuitive tool for policy development.

REFERENCES

[1] National Security Agency, "Security-Enhanced Linux," http:// www.nsa.gov/research/selinux, 2013.
[2] "OpenEZX," http://www.openezx.org, 2013.
[3] L. Potter, "Security in Qtopia Phones," LINUX J., http:// www.linuxjournal.com/article/9896, 2013.
[4] Tresys Technology, "SETools - Policy Analysis Tools for SELinux," http://oss.tresys.com/projects/setools, 2013.
[5] T. Krazit, "The Six Secrets to Mobile Computing Success," CNET, http://news.cnet.com/8301-13579_3-9929210-37.html, 2013.
[6] K.J. Biba, "Integrity Consideration for Secure Computer System," Technical Report TR-3153, Mitre Corp., 1977.
A. Bose and K. Shin, "Proactive Security for Mobile Messaging Networks," Proc. ACM Workshop Wireless Security, 2006.
[7] J. Carter, "Using GConf as an Example of How to Create a Userspace Object Manager," Proc. Security Enhanced Linux Symp., 2007.
[8] J. Cheng, S. Wong, H. Yang, and S. Lu, "SmartSiren: Virus Detection and Alert for Smartphones," Proc. ACM Conf. Mobile Systems, Applications, 2007.
[9] D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proc. IEEE Symp. Security and Privacy, 1987.
[10] W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," IEEE Security and Privacy, vol. 7, no. 1, pp. 50-57, Jan. 2009.
[11] W. Enck, P. Traynor, P. McDaniel, and T.L. Porta, "Exploiting Open Functionality in SMS-Capable Cellular Networks," Proc. 12th ACM Conf. Computer and Comm. Security (CCS), 2005.

