A Trusted Mobile Phone Reference Architecture via Secure Kernel

Xinwen Zhang, Onur Acıiçmez, and Jean-Pierre Seifert
Samsung Information Systems America
San Jose, California, USA
xinwen.z@samsung.com, o.aciicmez@samsung.com, jeanpierreseifert@yahoo.com

ABSTRACT

Driven by the ever increasing information security demands in mobile devices, the Trusted Computing Group (TCG) formed a dedicated group — Mobile Phone Working Group (MPWG) — to address the security needs of mobile platforms. Along this direction, the MPWG has recently released a Trusted Mobile Phone Reference Architecture Specification. In order to realize trusted mobile platforms, they adapt well-known concepts like TPM, isolation, integrity measurement, etc. from the trusted PC world — with slight modifications due to the characteristics and resource limitations of mobile devices — into generic mobile phone platforms. The business needs of mobile phone industry mandate 4 different stakeholders (platform owners): device manufacturer, cellular service provider, general service provider, and of course the end-user. The specification requires separate trusted and isolated operational domains, so called Trusted Engines, for each of these stakeholders. Although the TCG MPWG does not explicitly prescribe a specific technical realization of these Trusted Engines, a general perception suggests reusing the very well established (Trusted) Virtualization concept from corresponding PC architectures. However, despite of all its merits, the current "resource devourer" Virtualization is not very well suited for mobile devices. Thus, in this paper, we propose another isolation technique, which is specifically crafted for mobile phone platforms and respects its resource limitations. We achieve this goal by realizing the TCG's Trusted Mobile Phone specification by leveraging SELinux which provides a generic domain isolation concept at the kernel level. Additional to harnessing the potential of SELinux to realize mobile phone specific (isolated) operational domains, we are also able to seamlessly integrate the important integrity measurement and verification concept into our SELinux-based Trusted Mobile Phone architecture. This is achieved by defining some SELinux policy language extensions. Thus, the present paper provides a novel, efficient and inherently secure TCG-aware Mobile Phone reference architecture.

Categories and Subject Descriptors: D.4.6 [Operating Systems]: Access controls; K.6.5 [Management of Computing and Information]:Unauthorized access.

General Terms: Security.

Keywords: Trusted mobile computing, Secure kernel, Isolation, Security architecture, SELinux.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STC'07, November 2, 2007, Alexandria, Virginia, USA.
Copyright 2007 ACM 978-1-59593-888-6/07/0011 ...$5.00.

1. INTRODUCTION

With increasing computing power and pervasive network connectivity, we have seen significant proliferation in mobile phone usage and experienced mobile phones becoming more open and general-purpose computing environments for the last decade. As a natural consequence, more and more applications and services have been developed and deployed on these devices. This has brought new usage models and business processes such as mobile payment and ticking, software as a service (SaaS), pervasive information and content (audio, video, and text) sharing, seamless collaborations, voice-over-IP (VoIP) applications, and many more. Eventually, all these advances come with new security challenges, which are fundamentally different and cannot be satisfied with the security requirements of conventional limited purpose phones. As a result, we have seen an increasing number of vulnerabilities and security attacks in mobile computing.

Several organizations have proposed security requirements and specifications for mobile phone security. For example, the Open Mobile Terminal Platform (OMTP) organization, which is an operator-sponsored forum, has proposed an application security framework to specify security requirements for applications running on a mobile phone [29]. LiMo (Linux Mobile) Foundation, which is formed by major mobile device manufacturers, is designing a generic Linux-based open software platform for mobile communication devices. Security is claimed to be an important component in this framework. However, the formal specification has not been released yet [7]. The Open Mobile Alliance (OMA) [9] is an organization founded by nearly 200 companies including mobile operators, device and network suppliers, and application/content/service providers. The Security Working Group of OMA mainly specifies protocols for secure communication between mobile clients and servers at both the transport layer and the application layer.

The Trusted Computing Group (TCG) has published specifications for a Mobile Trusted Module (MTM) [15], which is a modified version of the TPM — the counterpart for PC platforms. Typically, a mobile phone is "owned" by multiple stakeholders, including device manufacturer, network operator, 3rd party service provider, and the user (customer). One owner cannot turn off or damage the services of another owner, e.g., by compromising the protection and security mechanisms. Thus, logically, an MTM can be owned by local and remote stakeholders, which is a unique feature of MTMs. Each of these stakeholders needs to have their own basic trusted computing mechanisms such as secure boot, secure storage, integrity measurement and verification, and remote attestation, which requires deployment of separate MTMs for each stakeholder.

The TCG further develops the Mobile Phone Reference Architecture (TCG-MPRA) specification [14], which specifies a general platform architecture for mobile devices relying on the trusted services of MTMs. Besides integrity measurement, trusted storage and reporting mechanisms, the TCG-MPRA requires the resources of different stakeholders to be strongly isolated and the communications between different agents and services — representing corre-

sponding stakeholders — to be controlled according to pre-agreed security policies. Particularly, the TCG-MPRA "generalizes" the concept of a platform to mean a set of conventional TCG-enabled platforms, and calls them "engines" to differentiate them from the ensemble platform" [14].

Besides isolation, the controlled communication and resource sharing between stakeholders is another security requirement in mobile devices. An application or a service from an engine can access the services from other engines only if it has the required permission to access. Therefore, the security policies should be defined accordingly and must be strictly enforced at run-time.

The TCG-MPWG does not "mandate a specific form or strength of isolation, which should depend on the purpose of the trusted mobile platform". As a solution typically on PC and server-based platforms, virtualization-based approaches [33, 10, 23, 31] can be used to fulfill the isolation requirement of TCG-MPRA. Specifically, the resources and applications of a domain are located in a dedicated virtual machine (VM) with its own operating system (OS) and application runtime environment so that a virtual machine monitor (VMM) or hypervisor assures the isolation of individual domains. The communication between the domains is controlled by the VMM according to pre-defined policies. Although this approach successfully fulfills the isolation requirement, it has a main drawback. Typically, virtualization is realized with an additional software layer that abstracts the underlying hardware resources (e.g., CPU cycles, memory, and storage) and it introduces significant overhead to the device. Hardware based approaches to virtualization can have even worse performance figures compared to software approaches [16]. Since mobile phones are extremely resource-constrained with limited computing capabilities and restricted power consumption, virtualization is evidently not a practical isolation solution for mobile phones. Also, controlled communication and resource sharing during VMM layer is not fine-grained.

On the other side, application-level trust mechanisms are used by many approaches. For example, in the OMTP security framework [29], the concept of trust level is introduced, which is to be determined by a mobile application's certificate. Depending on its trust level, an application obtains access permissions to different sensitive data and services deployed on the platform, e.g., SIM read/write, network services, and user phonebook/call history. However, an application level approach cannot provide strong evidence of the trustworthiness of a mobile device, since some critical functions such as root-of-trust and secure storage cannot be provided and strongly protected on an increasingly open mobile platform. Furthermore, access permissions of different domains based on certificates can be easily compromised on the operating system level, e.g., through unprotected private keys and their operations. Also, the application certificate process typically is very complex and has high administration cost on the trust management infrastructure.

Strong isolation and flexible resource sharing and communication are typically contradicting objectives. On one side, we need isolation (e.g.) to prevent a potential vulnerability in an application/service affecting the security of other services. For example, many viruses and worms on mobile phones spread via SMS (Short Message Service) and MMS (Multimedia Messaging Service) messages. If these services are isolated from the rest of the software and resources on the platform, these malware cannot easily infect the entire platform and quickly spread over an entire network. To be more specific, they cannot read phone books and/or the call history which are maintained by corresponding services in the device manufacturer domain, thus they cannot freely send out infected messages to other nodes in a network. On the other hand, we need to provide efficient methods to support frequent and dynamic interactions between different processes running on a mobile phone. These methods, besides being efficient, must also uphold the security of the system; otherwise active processes may become susceptible to an easy compromise (e.g.) by direct interference from other processes, or by fake/malicious input, or by unexpected data from sources with lower integrity or sensitivity levels. We could show and endless number of such examples to support the necessity of isolation and secure solutions for resource sharing and interprocess interactions.

Towards an affordable but effective solution for mobile devices, we propose an efficient isolation and flexible communication mechanism between different engines in multi-stakeholder environments. Our approach leverages maturing mechanisms from the secure kernel research area. Traditional Unix-like operating systems only provide discretionary access control (DAC), which makes it impractical to enforce strong isolation between system services and application processes. Fortunately, several kernel level but general-purpose mandatory access control (MAC) mechanisms have been developed and integrated into main stream operating systems such as Security Enhanced Linux (SELinux) [26], AppArmor [2], LOMAC [22], and LIDS [12]. Typically, a MAC system strictly controls — according to some predefined sets of rules — the interactions between subjects (e.g., services or processes) and objects (e.g., files, sockets, etc.), which are differentiated based on the labels assigned to them. Different policies can be implemented by defining different rules to enforce (e.g.) resource separation, data confidentiality and integrity, and a general information flow control.

The main advantages of our approach on mobile devices, i.e., kernel-level MAC mechanisms to trusted mobile platform, are the following.

1. Kernel-level mechanisms are intrinsically trusted, unlike application level security mechanisms such as those of OMTP. Simply because the kernel is a part of the trusted computing base (TCB) [21]. Therefore, the security enforcement in the kernel space is clearly much more trustworthy compared to an application level enforcement.

2. The advantage of a MAC-based isolation compared to virtualization techniques is pure performance. Mobile phones are extremely resource-constrained with limited computing capabilities and restricted power consumptions, which evidently makes virtualization an impractical solution for mobile phones. At this point, one can argue that current MAC mechanisms, e.g., as in SELinux, are also resource thirsty and would result in poor performance on mobile devices. It is correct that these mechanisms consume substantial computing power on PC platforms. However, the main reason of this situation is the vast number of subjects and objects on a PC. Mobile devices, although they become more and more open and multipurpose, are still limited and cannot be comparable to PC environments in this regard. Most of the services/applications we use in a PC are not present on mobile platforms (e.g., web server). This extensively simplifies the security policies (and also the security policy development) and significantly improves the potential performance of MAC mechanisms on mobile devices.

This paper first gives an overview of the TCG mobile phone reference architecture and its security requirements in Section 1. Background information on isolation and secure kernel is presented in Section 3. We then discuss how secure kernel mechanisms can meet these requirements, particularly with SELinux, in Section 3.2. Our extensions of the SELinux policy model and its architecture are presented in Section 5 to illustrate how we integrate measurement and verification into our approach. We then conclude in Section 6.

2. THE TCG MOBILE REFERENCE ARCHITECTURE

2.1 Why Do We Need Trusted Mobile Devices?

As stated above, the advances in mobile devices come with new security challenges, which are fundamentally different and cannot be satisfied with the security requirements of conventional limited

purpose phones. The differences between current general-purpose mobile phones and their traditional counterparts mandate new security considerations. The following are only a few examples of such differences.

- Current devices store much more — both in quantity and in scale — sensitive data. For example, the user's information such as credit card numbers and e-tickets, network operator's data such as subscriber identity and billing information, data related to other mobile services and applications such as licenses for digital rights management (DRM), etc. are extremely sensitive and must be protected.
- Mobile devices are becoming more and more open and general-purpose computing environments, not only for personal computing, but also for increasing commercial and enterprise computing.
- With increasing connection capabilities such as 3G/4G voice and data network access, WiFi/WLAN, Bluetooth, WiMax, and UPnP, mobile computing has become more pervasive and ad-hoc than ever.

These main aspects result in increasing vulnerabilities and attacks on the mobile computing front. We can argue, using the evolvement of PC-based general-purpose computing platforms as evidence, that the lack of intrinsic and comprehensive security solutions may slow down the evolution and widespread adoption of new usage models on mobile platforms. Therefore, it is inevitable to reconsider the security challenges of mobile devices and define new security requirements. To address these issues, the TCG formed a working group dedicated to the mobile phone area. In this section, we will cover the mobile phone reference architecture and its security requirements as defined by TCG's Mobile Phone Working Group (TCG-MPWG).

2.2 Overview of TCG-MPWG's approach

As said above, the TCG has published specifications for a Mobile Trusted Module (MTM) [15] and a Mobile Phone Reference Architecture (MPRA) [14]. Here, the MPRA specifies a general platform architecture for mobile devices relying on the trusted services of MTMs. Also, the TCG-MPWG has defined two main security requirements that are particular interesting for us in this paper: complete isolation and controlled communication and resource sharing between different engines.

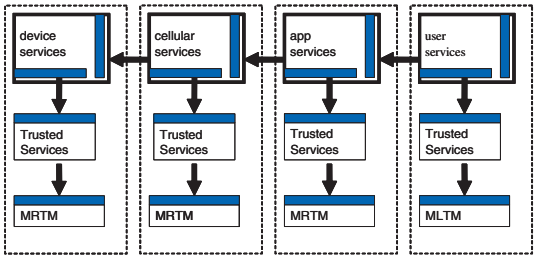

Figure 1: Example of a generalized mobile platform [15]

Figure 1 shows a generalized mobile platform, where two types of trusted modules exist: A Mobile Remote-Owner Trusted Module (MRTM) and a Mobile Local-Owner Trusted Module (MLTM). TCG requires strict control over the communications and resource sharing between different stakeholders. For example, the service dependency is indicated by the solid arrow between engines shown in Figure 1. Particularly, the device engine provides basic platform resources, which include a user interface, debug connector, a radio transmitter and receiver, random number generator, the International Mobile Equipment Identity (IMEI), and a Subscriber Identity Module (SIM) interface. The device engine provides its services to an engine that provides cellular services. The cellular engine provides its services to an application engine, and the application engine provides its services to the user.

More applications and services can be built by composing existing services from different engines. For example, Figure 2 shows that engines in different domains (owned by different stakeholders) are able to communicate (shown by thick lines) with each other and access generic services provided by a device manufacturer's engine. As another security requirement, an application or a service from an engine can access the services from other engines only if it has the required access permission. Therefore, the security policies should be accordingly defined and must be strictly enforced during run-time. Due to the characteristics of current mobile phones, a single security policy cannot satisfy the needs of all of these different stakeholders.

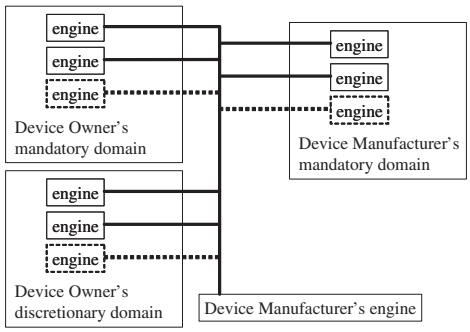

Figure 2: Communication between engines on a generalized trusted mobile platform [14]

The TCG-MPWG recognizes 4 types of owners (i.e., stakeholders) of a mobile phone: device manufacturer, network operator, 3rd party service provider, and the actual user. Each stakeholder owns dedicated trusted engines and has exclusive control over the data protection mechanisms provided by them. Note that each stakeholder can have multiple engines on a single platform as shown in Figure 2. A generic engine is shown in the following Figure 3.

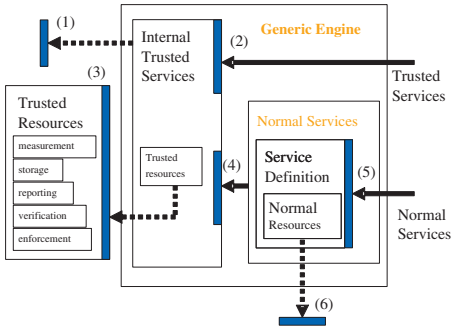

Figure 3: Overview of a generic engine [14]

The platform must support each engine with all of the necessary resources to carry out their tasks and interact with each other in a secure and trusted manner. Conceptually, each engine is a trusted stand-alone entity with the same predefined set of functionalities — although some functions may be optional for a particular engine. Therefore, each engine has its own set of trusted resources, which include the roots of trust for measurement (RTM), storage (RTS), reporting (RTR), and verification (RTV). These roots of trust mechanisms constitute the main trusted computing base, which allow an engine to provide trusted services such as authenticated boot, integrity measurement and storage, and attestation. We want to mention that a particular engine may not necessarily have all of these

roots of trust. Also, each engine provides — beside its trusted services — normal services, where the trusted services can be leveraged, for example, a normal service can be measured by an internal trusted service through an interface.

The isolation requirements on TCG defined mobile phone devices are considered in two different levels. First, we need isolation across different engines. Specifically, the resources (both trusted and normal) of an engine must be protected against malicious intervention from other engines. Second, isolation is also needed between trusted and normal resources within a single engine.

Different engines, whether they belong to a single stakeholder or different stakeholders, must be isolated from each other. But at the same time, an engine needs to provide some services to other engines through dedicated interfaces, which must also be protected against unauthorized accesses. Any communication between two different engines is always through some interfaces. According to Figure 3, for example, this engine can access services of other engines only through interfaces (1) and (6), and provides services to others only through interfaces (2) and (5).

Another level of isolation is needed between trusted and normal resources within a single engine. According to the TCG-MPRA specification, "internal trusted services must be isolated from normal services". Similar to resource sharing between different engines, within an engine, normal resources can use the trusted resources through dedicated interfaces, e.g., (4) indicated in Figure 3.

In general, we can make the following statement. *Isolation for a TCG complaint trusted mobile platform exists on resource level and interfaces can be the only permissible gateways for inter-engine communications.*

If we consider of all these stated facts, we can now define an engine as a set $\varepsilon = \{\mathcal{TR}, \mathcal{NR}, \mathcal{TS}_{\mathcal{I}}, \mathcal{NS}_{\mathcal{I}}, \mathcal{TS}_{\mathcal{O}}, \mathcal{NS}_{\mathcal{O}}\}$, where

- $\mathcal{TR}$ is a set of trusted resources;
- $\mathcal{NR}$ is a set of normal resources;
- $\mathcal{TS}_{\mathcal{I}}$ is a set of trusted services provided by other engines and used by ε;
- $\mathcal{NS}_{\mathcal{I}}$ is a set of normal services provided by other engines and used by ε;
- $\mathcal{TS}_{\mathcal{O}}$ is a set of trusted services provided by ε to other engines;
- and $\mathcal{NS}_{\mathcal{O}}$ is a set of normal services provided by ε to other engines.

An engine ε is owned by either a local or a remote stakeholder s_ε, and is managed by an administrative role, either an agent of s_ε or a delegated one. The TCG MPWG specifies four principle stakeholders: users (local), service providers (remote), communication carrier (remote), and device manufacturer (remote).

2.3 Mandatory/Discretionary Engines

TCG distinguishes *mandatory* and *discretionary* engines. A mandatory engine is always resident on a platform and provides critical and indispensable services, which has to be running in a *known-good state*. By "running in a known-good state", we mean that the integrity of a service must be verified to assure its trustworthiness, i.e., it provides expected services in an expected manner and respects the protections of sensitive data. According to TCG-MPRA, a mandatory engine must provide and support (TCG defined) trusted functions like root of trust mechanisms, secure boot, and integrity measurements to allow a remote stakeholder to verify the boot sequence and run-time state of the engine. Therefore, this type of engine is called TCG-enabled engine. Typical mandatory engines include those of the device manufacturer, the communication carrier, and enterprise servers (if the device is owned by an enterprise)

A discretionary engine, on the other hand, may or may not be resident on a platform during run-time. In a general sense, discretionary engines provide non-critical services. Typical discretionary engines include those from third party service provider (e.g., content provider) and the engines owned and controlled by the users (network service customer).

3. PRELIMINARIES

3.1 Isolation Principles

Isolation is a fundamental security problem, which has been extensively studied in separation kernels [32] and hypervisors [25]. The basic problem in the original isolation research is to prevent any kind of information flow between regimes [1]. Rushby [32] has presented the fundamental security model of this "complete" isolation and its verification. The model specification and verification are based on system states which are defined according to subjects, objects, and their labels, the operations between them, inputs, outputs, and I/O devices. Intuitively, this model is applied in the separation of virtual machines by virtual machine monitors [25]. A complete isolation model requires isolated operations, i.e., operations of one domain should not have any external effects on other domains. Also, each domain's I/O device should be independent.

However, this complete isolation model is too restrictive for a modern mobile platform — due to a simple reason: the engines on a mobile platform have to communicate and share resources with each other. For example, an engine of a service provider needs to use the voice communication channel, which is a service provided by an engine of the network carrier. For another example, an engine needs to access user information on a SIM card, e.g., for payment purposes, which is a resource typically controlled by either the network carrier's or device manufacturer's engine. We will elaborate more on different types of engines and the interactions between them in the following sections.

3.2 MAC-based Secure Kernel

The concept of security-enhanced kernels has been studied for a long time [21]. Recent approaches along this line include integration of flexible mandatory access control (MAC) mechanisms into kernels. Note that most of the conventional operating systems only support discretionary access control (DAC).

MAC can be regarded as a relaxed form of the complete isolation model. For example, in a complete separation model, s can access (e.g. read or write) o only if $\lambda(s) = \lambda(o)$, where $\lambda(s)$ is subject security level and $\lambda(o)$ is the object security classification. That is, information flow is only allowed between entities within the same domain. However, in a label-based MAC model such as Bell-LaPadula [17, 21], which forms the basis of multilevel security, s can read o if $\lambda(s) \geq \lambda(o)$, and s can write to o if $\lambda(s) \leq \lambda(o)$. That is, information flow can be allowed only from low level objects to high level objects.

Several MAC models have been proposed to secure OS kernels, e.g., Biba [18], Clark-Wilson model [20], and LOMAC [22]. Domain-type enforcement is another form of MAC which uses non-hierarchical security labels [19]. This model can be easily used to enforce resource isolation and to control communication between individual domains by specifying dedicated pipelines. SELinux uses this as its policy model, which will be explained with more details in the following sections. An important application for kernel level isolation is the recent NetTop solution with SELinux [6].

It is crucial to realize that resource isolation and controlled inter-engine communication through dedicated pipelines (called interfaces in the TCG-MPRA) are the two main mobile phone security requirements for TCG-compliance. One of our contributions in this paper is to meet these requirements by leveraging existing SELinux features and crafting them especially to realize an alternative TCG-MPR architecture.

[1]Here we use the term of "regime" which is directly from [32]. In this paper we do not distinguish the difference between regime and engine.

4. A TRUSTED MOBILE PHONE ARCHITECTURE WITH SELINUX

In the rest of this paper, we describe how SELinux can be used to achieve the main security goals of a trusted mobile platform. One of the motivations of our approach is the fact that Linux-based mobile platforms have been gaining increased market shares [7, 13]. However, for confidentiality purposes, we cannot disclose all details of our advanced security architecture concept — except the basics that we cover in this paper.

4.1 SELinux Security Mechanisms

SELinux implements strong mandatory access control (MAC) using Linux Security Modules (LSM) inside the Linux kernel — based upon the principle of least privilege [35]. This clearly enhances the security of the standard Linux, which makes use of only discretionary access control (DAC). SELinux associates attributes of the form `user:role:type` to all subjects (processes) and objects (files, IPC, sockets, etc.), which is called *security context*. Within the security context, the `type` attribute represents the type of the subject or the object, e.g., `http_t` and `user_home_t`. The identification of subjects, objects and their access enforcement by means of types is formally known as *Type Enforcement* (TE). Instead of directly associating a `user` with a `type`, SELinux associates a `user` with a `role` and a `role` with a set of `types`. The `role` merely simplifies the management of users. Therefore, access control in SELinux is primarily enforced via TE [19, 28].

A typical policy rule in SELinux is to `allow` a `type` of subjects to have some permissions on some `types` of objects. The permissions are defined according to object classes, e.g., `{read, write, append, lock}` for file related object, and `{tcp_send, tcp_recv}` for network interface objects. These rules are checked by security hook functions pre-defined inside the Linux kernel, e.g., when a file is opened or a socket is created and bound.

The main security mechanism of SELinux is to control a subject's activities with the least-privilege principle, and confine the information flow between domains. Let us give a simple example.

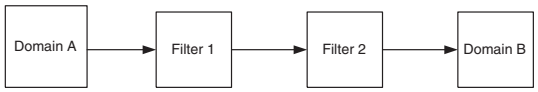

Figure 4: Domain isolation and controlled communication with SELinux

The above Figure 4 shows isolated domains A and B and their communication channel through filters. In this case, SELinux types and policy rules can be defined to ensure all of the following:

1. the information flow can only be allowed from Domain A to Domain B and not vice versa.
2. the information has to be checked by Filter 1 (e.g., virus check) followed by Filter 2 (e.g., spam check) before it goes to Domain B; i.e., the filters are not bypassable.
3. Filter 1 and Filter 2 are the only entities that can read the information sent from Domain A to Domain B;
4. Filter 1 and Filter 2 can only access the information from Domain A to Domain B and nothing else in the system.

Through these policy rules, if Filter 1 and 2 are trusted to do their correct job, any information from Domain A to Domain B can be correctly controlled. To give an example, assume that Domain A has a lower integrity level than Domain B. If Filter 1 and 2 are trusted (whose integrity can be verified in SELinux via our extensions as explained in the following and later section) to validate the format and the content of data, then the input (data or service) from Domain A to Domain B can be sanitized and thus cannot compromise the integrity of Domain B. Note that in SELinux, if there are no policy rules defined (directly or indirectly) for two domains to communicate, there cannot be any information flow between them. Information flow can happen between different domains either directly (e.g., a subject of one domain to read/write an object of another domain), or indirectly (e.g., read/write through one or more intermediate objects of different domains).

To preserve strong isolation but a controlled communication between domains, a least-privilege security administration is also needed. For example, in the above example, no security administrative permissions of Domain B cannot be assigned to any administrative roles in Domain A, otherwise the policy rules can be maliciously modified, e.g., make another uncontrolled channel between A and B, or by-pass the filters.

SELinux provides fine-grained access control. Because of this, a SELinux policy typically consists of a large number of types and policy rules; thus a complete analysis of the permissions between subjects and objects becomes impractical. The recent policy development in SELinux uses high level domain definitions and decomposition mechanisms, which are used in the cross-domain solution (CDS) framework [27, 36]. In this framework, a policy is represented as a graph, as shown in Figure 5 with an example for a daemon.

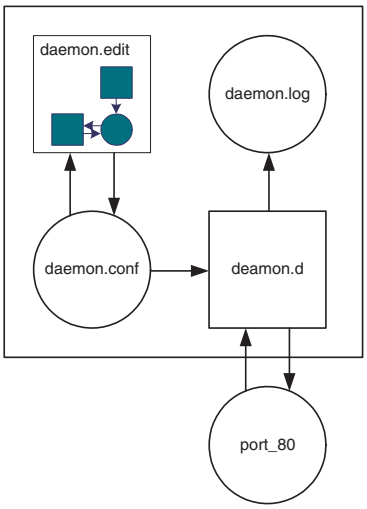

Figure 5: Graphical representation of an example policy

In this graphical policy language, a domain is represented by a box, and resources are represented by circles. A domain can have its own private resources, and can read from and write to other shared resources within other domains. Also, a domain can be decomposed into many domains. Note that in the CDS Framework, the domain concept is more general than that in the original SELinux policy rules. A domain here represents an "information boundary", which can contain a number of entities including active subjects (e.g., processes), passive objects (e.g., files and sockets), and transition objects (e.g., file entrypoints).

4.2 Domain Isolation with SELinux for a Trusted Mobile Platform

We will use the SELinux isolation mechanism on a generic trusted mobile platform. Specifically, the isolation between engines requires that an engine cannot directly access the resources of another engine, while they only can communicate through dedicated and non-bypassable interfaces. In this subsection, we explain how SELinux can be used to enforce isolation between engines on a trusted mobile platform. We use the graphical policy model to describe a generic engine of a mobile platform defined by TCG .

As aforementioned, an engine has its private trusted resources and normal resources, and can receive inputs from and provide outputs to other engines. To preserve strong isolation, two engines can

only communicate through shared resources (i.e., interfaces). The security policy can be expressed as in Figure 6.

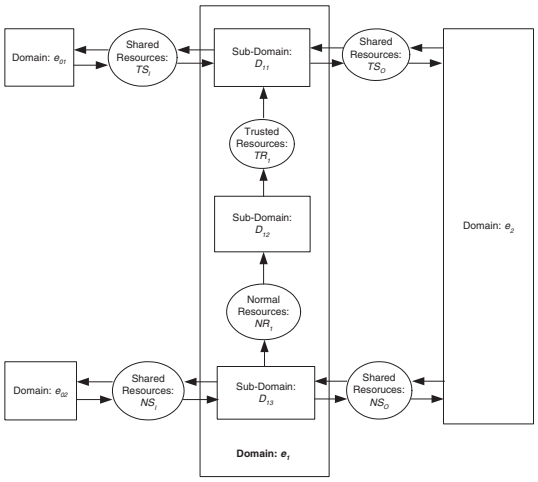

Figure 6: Graphical representation of a security policy for a generic engine.

In this policy, a generic engine e_1 is represented as a domain, which has its private trusted resources (TR_1) and normal resources (NR_1). As each engine provides internal trusted services and normal services, the sub-domain D_{11} and D_{13} indicates these service entities. In the context of a concrete SELinux system, these can be the types of individual services or application processes.

According to the generic engine specified by the TCG (refer to Figure 3), internal trusted services of e_1 can be accessed by the normal services, which is indicated between the sub-domain D_{12} of e_1. Typical functions provided by trusted services to normal services include integrity measurement, secure storage, and monotonic counters. The information flow between these services is typically one-way.

Each engine can consume services provided by other engines and provides services to others, both can be trusted resources or normal resources. These communications are represented by shared resources between e_1 and other engine domains. Specifically, domain e_1 can get trusted resources TS_I from domain e_{01} and provide other trusted resources TS_O to domain e_2. Similarly, e_1 can consume normal resources NS_I from domain e_{02} and provide normal resources NS_O to domain e_2. The one-way information flow indicates typical usage scenarios of communication between engines. For example, an engine of network carrier provides cellular service to an engine of third party service provider, and it does not consume any service from this engine. In more general scenarios, two way-information flow can be allowed if appropriate interfaces are defined.

As said earlier, a domain in the policy model of SELinux can be decomposed. In our approach, a generic engine e_1 can be represented as a domain, and it is decomposed into three sub-domains: the trusted domain D_{11}, the normal domain D_{13}, and the domain entrypoints D_{12}. At the same time, D_{11} and D_{13} act as the entities to communicate with external engines on trusted and normal services, respectively. Therefore, it is natural that each of these sub-domains can be further decomposed into a few sub-domains, each of which handles different communications between other internal and external domains. Note that domain e_{01} and e_{02} can belong to two distinct engines as well as the same engine. In the most general scenario, an engine can consume services from and provide services to the same or different engines. The actual details of possible domains and sub-domains and also the interactions between them depend on the exact implementations of the available engines on a platform and the service-level agreements between different engines.

We use the CDS Framework IDE from Tresys [4] to develop the graphical policy definition. This can be transformed to a high-level language as shown in Figure 7 and 8 and then transformed into the NSA example strict policy [8] or reference policy [11].

4.3 Separation of Security Administration

In traditional PC and server systems, a single system security administrator or the root has the ultimate control of the platform, and external entities have to trust the administrator and the policies deployed. On a mobile platform, the problem is different since naturally it is a multi-stakeholder environment. The TCG-MPRA specifies that "Mandatory engines are required to be supported by a Mobile Remote owner Trusted Module (MRTM) which supports secure boot and does not permit a local Operator to remove the stakeholder from the engine". That is, the security of a mandatory engine cannot be compromised by any other local or remote stakeholder of the platform.

Aforementioned mechanisms can satisfy runtime security requirements of resource isolation and control sharing. However, if the policies can be maliciously modified, the overall security goal will surely be compromised. SELinux provides fine-grained access control on system resources and information flow, however, it only supports coarse-grained permission control and policy management. For example, in a PC-base platform, an SELinux policy can be updated by the root with the role `sysadm_r`. In mobile platforms, each domain (or engine) should have its own security administrative role, which has the permission to modify and load a policy into a running system but has no authority over other domain's policies. Similar to general subjects, this is the least-privilege principle for security-related permissions.

The recently introduced *loadable policy modules framework* supported by the SELinux Reference Policy [11, 30] offers a flexible way to create self-contained policy modules that are linked to the existing policy without re-compiling the whole and original policy source. A policy module can define private types and attributes and then define interfaces so that other modules can use them, thus enabling communication between the subjects whose types are defined in different modules. After a module is created, it can be loaded to the kernel during runtime using the SELinux policy management infrastructure. Thus, capabilities to change and load policy modules can be defined and assigned to different administrative roles of different domains. Also, each domain has its own policy module where its internal resources can be defined with private types and only shared resources can be "accessed" through module interfaces by other policy writers (e.g., the administrative roles of other engines). This provides modularity and encapsulation for policy development and management, and self-protection for security system.

5. EXTENDING SELINUX FOR INTEGRITY MEASUREMENT AND VERIFICATION

The TCG specifies that all software executed on a secure-boot engine must be authenticated, thus requires that each software has to be measured when loaded. Also, the integrity should be verified by a security enforcement component before it communicates with existing processes in the engine, as otherwise it may compromise the overall security of the stakeholder. On the other side, the normal resource of an engine can be measured by its trusted resource, and the integrity can be verified by other engines or remote entities through attestation.

Similar to resource isolation and controlled communication, TCG does not specify the implementation of integrity measurement and verification in a mandatory engine. We propose an architecture where integrity measurement and verification are integrated into the security policy enforcement mechanism. Our architecture is extended from the SELinux security module.

Figure 9 shows our architecture based on SELinux, which is built on the Linux Security Module (LSM). With LSM enabled, a set of

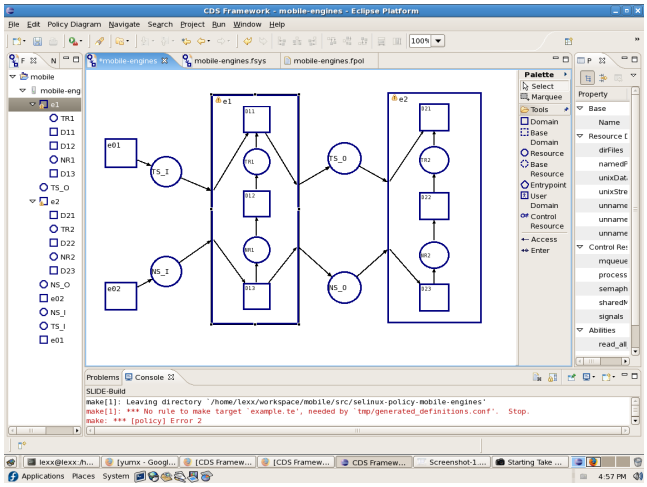

Figure 7: Graphical policy development in CDS Framework

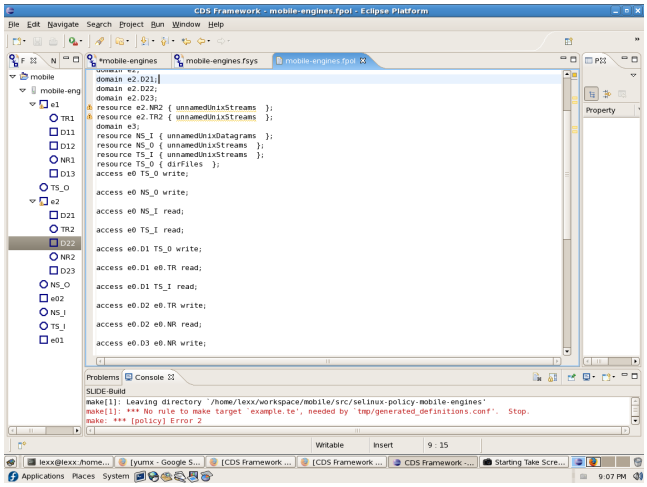

Figure 8: A policy generated by CDS Framework in Eclipse

hooks is placed inside the kernel source code such that whenever sensitive system call functions are called by high level processes, the corresponding hook checks the LSM. In SELinux LSM, the security server implements the decision based on its type enforcement policy model, where the binary policy is pre-loaded. To enable integrity measurement and verification before a software is executed, the SELinux LSM and its policy model are extended in the following way.

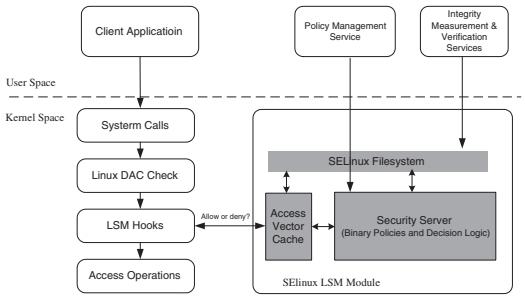

Figure 9: Extended SELinux Architecture

5.1 Policy Extension for Integrity Measurement and Verification

The current SELinux security context is limited to Type Enforcement only. In our advanced SELinux-based TCG MPRA, we extend the SELinux policy model to even include integrity-related attributes in addition to roles and types, cf. [1]. Logically, in our framework, the extended SELinux security context contains two more contextual attributes called `profile` and `system`. The extended security context is of the form `user:profile:role:type:system`.

The additional `profile` contextual attribute allows the specification of a general set of attributes that can be associated with a particular subject or object. As an example, consider a medical engine on a mobile platform. A role `medical_r` assigned to `medicalObject_t` type and its `profile` contains attributes of `NoOfDayAllowed` and `accessLocation` to confine the times of allowed access and the location. On the other hand, the contextual `system` attribute allows the specification of a general set of system related attributes that can be associated with a subject or an object. Particularly, we focus on integrity purposes. For example, in order to verify the integrity of the medical software with which medical records are processed, the following SELinux `constrain` policy expression is used:

```
constrain file (s1.integrity = 1),
```

where `integrity` is an attribute defined within the `system` contextual attribute of the subject (the medical application). The predicate `s1.integrity=1` is used to eensure that the integrity of the subject is verified and trusted. Other requirements, e.g., the system configuration, the anti-virus status, etc. can also be defined and specified in a similar fashion.

5.2 Policy Enforcement Extension for Integrity Verification

The extended policy not only specifies the domain or type of a subject, but also its integrity status in order to obtain the access permission. Therefore, the SELinux policy enforcement mechanism needs to be extended to consider also the integrity requirement.

The left part of Figure 9 is the usual SELinux access permission check module. On the right part, there is a set of services running on the user space, including the policy management and integrity verification. In the kernel space, the SELinux virtual filesystem provides interfaces to allow user space services to set boolean values for *conditional policies*. When a new engine is installed, its policy is compiled and loaded to the security server by the policy management service. For example, the following SELinux conditional policy rules states that if `medical_integrity` is true, read access is allowed from a certain medical application to some medical records.

```
  if (medical_integrity) {
    allow medicalApplication_t medicalObject_t
file:{read getattr search};
  } else {
    allow medicalApplication_t medicalRecord_t
file:{getattr};   }
```

The boolean variable `medical_integrity` is defined to indicate the integrity status of the target medical application process, which is set by the integrity verification service. Whenever there is an access request, the SELinux security server obtains the current value of the boolean variable and makes a decision.

Integrity verification is based on an efficient integrity measurement mechanism. Integrity Measurement Architecture (IMA) [34] is a system software for Linux developed by IBM to provide a verifiable evidence regarding the integrity of a system on which it is running. Instead of relying on the trustworthiness of the software environment of a system, IMA builds on a hardware extension of the measured system — the Trusted Platform Module (TPM). In the current version of IMA, a list of hash values is maintained for

all executable contents loaded into the runtime since the boot of the system. In addition to runtime environment, IMA also measures the BIOS and the boot loader. For establishing trust, remote systems can retrieve the measurements through special interfaces. The measurements represent the integrity history and are maintained in a kernel list which is protected by the TPM. Recent efforts enhances the efficiency of IMA [24]. In our project, we have re-used some parts of the IMA code for integrity measurement and protection via our SELinux policy extension.

The integrity verification service daemon is responsible for monitoring the integrity of all target files measured by the integrity measurement service. Each listed file has a corresponding SELinux boolean variable which is stored in a corresponding file inside the SELinux filesystem. For example, in our healthcare scenario, the file `/usr/share/medicalPolicy` is listed in `/sys/kernel/measure` and the boolean variable file `/selinux/booleans/user_share_medicalPolicyIntegrity` corresponds to it.

Here, the absolute path of a target file is used for the boolean variable filename to avoid conflicts between boolean variables augmented by different loadable policy modules. I.e., the absolute path of the integrity protected file is sufficient for each SELinux boolean variable. Thus, each boolean variable is formed by the name of the file and the absolute path to the integrity protected file with underscores separating the path.

The integrity verification service daemon retrieves the measurement list and the `/sys/kernel/measure` file periodically. More than one entry for an integrity protected file in the measured kernel list designates that the file is compromised. Based on the measurement list integrity verification, this daemon sets the corresponding boolean variables accordingly. For example, in our scenario, the integrity verification service sets the `usr_share_medicalPolicyIntegrity` to false if the `/usr/share/medicalPolicy` has more than one entry in the kernel list.

An important feature of SELinux is that an ongoing access can be revoked when the security context or related policies are changed. For example, in SELinux, file access permission is checked on every read and write to a file, even when the file has been already opened. With this, if the security property of the file or the accessing subject changes, the access is revoked on the next read or write attempt. With this feature, many flexible and dynamic security requirements can be supported, such as runtime integrity verification. Due to space limitations, we need to ignore here some more details.

6. CONCLUSION

In this paper, we first provided an introduction into the recently released TCG mobile platform reference architecture, which is centered around the functions of a Mobile Trusted Module. We then analyzed the security requirements for such a trusted mobile platform. As a multi-stakeholder computing environment, a trusted mobile platform inherently requires strong domain isolation and controlled resource sharing between its different domains. To solve this problems, several approaches have been outlined by the TCG. These approaches either consume too much computational resources and thus result in poor performance on resource-constrained environments, or do not provide sufficient security protections for such open and general-purpose computing platforms. In order to overcome these shortcomings, we proposed a novel technique, in fact a new high level reference architecture — how to realize a trusted mobile phone device compliant with the TCG MPRA. Our approach leverages existing mandatory access control mechanisms in SELinux and adds some novel but crucial SELinux policy extensions. This way, we could provide a strong and convenient mechanism to satisfy two key security requirements of trusted mobile phones: isolation of engines — while supporting secure and flexible inter-engine communication — and integrity assurance. In addition to the above two key points a TCG-aware trusted mobile platform actually requires more: a root-of-trust, such as the TCG MTM, and a secure boot mechanism, cf. [3, 5, 14, 15].

7. REFERENCES

[1] M. Alam, M. Hafner, J.-P. Seifert, and X. Zhang. Extending SELinux Policy Model and Enforcement Architecture for Trusted Platforms Paradigms. In *Annual SELinux Symposium 2007*.

[2] Apparmor. http://en.opensuse.org/AppArmor.

[3] J. Brizek, M. Khan, J.-P. Seifert, and D.A. Wheeler. A Platform-level Trust-Architecture for Hand-held Devices. In *CRASH (2005)*.

[4] CDS Framework IDE. http://oss.tresys.com/projects/cdsframework.

[5] T. Eisenbarth, T. Güneysu, C. Paar, A.R. Sadeghi, D. Schellekens, and M. Wolf. Reconfigurable Trusted Computing in Hardware. In *ACM STC '07*.

[6] HP NetTop: A technical overview. http://h20338.www2.hp.com/enterprise/downloads/HP_NetTop_Whitepaper2.pdf.

[7] Limo foundation. https://www.limofoundation.org.

[8] NSA Security-Enhanced Linux Example Policy. http://www.nsa.gov/selinux/.

[9] Open Mobile Alliance. http://www.openmobilealliance.org.

[10] Open trusted computing (opentc) consortium. http://www.opentc.net/.

[11] SELinux Reference Policy. http://oss.tresys.com/projects/refpolicy.

[12] The Linux Intrusion Defence System (LIDS). http://www.lids.org/.

[13] Linux phone market opening up? http://www.linuxdevices.com/news/NS8591201260.html, 2007.

[14] TCG mobile reference architecture specification version 1.0. https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-reference-architecture-1.0.pdf, June 2007.

[15] TCG Mobile Trusted Module Specification Version 1.0. https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf, June 2007.

[16] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–13, San Jose, CA, USA, October 21-25 2006.

[17] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. *Mitre Corp. Report No.M74-244, Bedford, Mass.*, 1975.

[18] K.J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, The Mitre Corporation, Bedford, MA, April 1977.

[19] W. Boebert and R. Kain. A practical alternative to hierarchical integrity policies. In *Proc. of the 8th National Computer Security Conference*, 1985.

[20] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, May 1987.

[21] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria*, December 1985. DoD 5200.28-STD.

[22] T. Fraser. LOMAC: MAC you can live with. In *Proc. of the 2001 Usenix Annual Technical Conference*, Jun 2001.

[23] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 193–206, Bolton Landing, New York, USA, October 19-22 2003.

[24] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: Policy-reduced integrity measurement architecture. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 19–28, June 2006.

[25] N. L. Kelem and R. J. Feiertag. A separation model for virtual machine monitors. In *IEEE Symposium on Research in Security and Privacy*, 1991.

[26] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of USENIX Annual Technical Conference*, pages 29 – 42, June 25-30 2001.

[27] K. MacMillan, S. Shimko, C. Sellers, F. Mayer, and A. Wilson. Lessons learned developing cross-domain solutions on selinux. In *Proc. of SELinux Symposium*, 2006.

[28] F. Mayer, K. MacMillan, and D. Caplan. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall, 2007.

[29] OMTP. Application security framework. http://www.omtp.org/docs/OMTP_Application_Security_Framework_v2_0.pdf, 2007.

[30] C. J. PeBenito, F. Mayer, and K. MacMillan. Reference policy for security enhanced linux. In *Proc. of SELinux Symposium*, 2006.

[31] R.Sailer, T. Jaeger, E. Valdez, R. Perez, S. Berger, J. L. Griffin, and L. van Doorn. Building a mac-based security architecture for the xen opensource hypervisor. Technical report, IBM Research Report RC23629, 2005.

[32] J. M. Rushby. Proof of separability: A verification technique for a class of security kernels. In *Computing Laboratory, University fo Newcastle Upon Tyne*, May 5 1981.

[33] A. Sadeghi and C. Stuble. Taming trusted platforms by operating system design. In *Proceedings of the 4th International Workshop for Information Security Applications, LNCS 2908*, pages 286–302, Berlin, Germany, August 2003.

[34] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, pages 223–238, 2004.

[35] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[36] A. Wilson. SEFramework: A new policy development framework and tool to support security engineering. In *Proc. of SELinux Symposium*, 2005.