2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing

Security Enforcement Model for Distributed Usage Control

Xinwen Zhang and Jean-Pierre Seifert
Samsung Information Systems America
San Jose, California, USA
{xinwen.z, j.seifert}@samsung.com

Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio
ravi.sandhu@utsa.edu

Abstract

Recently proposed usage control concept and models extend traditional access control models with features for contemporary distributed computing systems, including continuous access control in dynamic computing environments where subject attributes and system states can be changed. Particularly, this is very useful in specifying security requirements to control the usage of an object after it is released into a distributed environment, which is regarded as one of the fundamental security issues in many distributed systems. However, the enabling technology for usage control is a challenging problem and the space has not been fully explored yet. In this paper we identify the general requirements of a trusted usage control enforcement in heterogeneous computing environments, and then propose a general platform architecture and enforcement mechanism by following these requirements. According to our usage control requirements, we augment the traditional SELinux MAC enforcement mechanism by considering subject/object integrity and environmental information. The result shows that our framework is effective in practice and can be seen as a general solution for usage control in distributed and pervasive computing environments with widely deployed trusted computing technologies on various computing devices.

1 Introduction

The traditional access control problem [10, 13, 18] is considered in closed environments, where identities of subjects and objects can be fully authenticated, and enforcement mechanisms are trusted by system administrators which define access control policies. However, with increasing distributed and decentralized computing systems, more computing cycles and data are processed on *leaf nodes*. This leads to two distinct access control problem spaces. The first one focuses on the reasoning of authorizations with subject attributes from different authorities. For example, in trust management [4, 9, 14, 19] systems, a user presents a set of attributes or credentials and another subject (e.g., a resource or service provider) can determine the permissions of the user based on the presented credentials. In this problem, objects are typically protected in a centralized server. The second problem focuses on continuous access control to an object after it is distributed to other (decentralized) locations or platforms, which is referred as the usage control problem proposed by researchers in literatures [15, 21, 22, 27].

Although there is no precise definition in the literature, the main goal of usage control is to enable continuous access control *after* an object is released to a different control domain from its owner or provider, especially in highly distributed and heterogeneous environments. Typically, a usage control policy is defined for a *target object* by its *stakeholder*, which specifies the conditions that accesses to the object on a *target platform* can be allowed [1]. A stakeholder can be the owner of a target object, or a service provider that is delegated by the object owner to protect the object. An object in usage control can be static data, various types of messages, or user or subject attribute or even a credential. Thus, this makes the problem pervasive in many distributed computing applications such as healthcare information systems, Web Services, and identity management systems. Different from other distributed access control problems such as trust management, in usage control, an object is located out of the controlling domain of a policy stakeholder such that (1) there are many aspects of access control decisions other than subject identities and attributes, and (2) an object stakeholder needs high assurance on the enforcement of the policy.

As Figure 1 shows, an object and its usage control pol-

[1] Some literatures present another way of usage control, which focuses on confining the usage purposes of an object, instead of security sensitive purposes like integrity and confidentiality. The typical example on this kind of usage control system is digital rights management (DRM), which controls a user's use of an object based on payment information, e.g., play and copy. However, we do not distinguish usage rights and other security sensitive rights on an object, thus this kind of usage control is a subset of the problem in this paper.

978-0-7695-3158-8/08 $25.00 © 2008 IEEE
DOI 10.1109/SUTC.2008.79

icy are distributed from a data provider to a target platform. The policy is enforced in the platform to control access to the object within a *trusted subsystem*. Typically, an access control decision is determined according to pre-defined factors specified in a policy, which, logically, can be defined based upon the information of the subject and the object of an access request, where the subject is an active entity trying to perform actions on the passive entity object. In closed access control systems such as in a local platform, policies are defined based upon the identities of subjects and objects. In traditional distributed access control systems such as trust management, policies are defined based on attributes or credentials that are certified by external authorities. However, in usage control, access control policies can be defined by very general attributes of subjects and objects, such as application-specific attributes and temporal status. Furthermore, as an object can be located on platform in a heterogeneous environment such as a mobile device, environmental restrictions and system conditions are mandatory decision factors in many applications, such as location-based service and time-limited access. An *ongoing* access should be terminated if these environmental or system conditions change which violate policies. For example, a mobile application might require that a service can be used only if a mobile device is in a particular location, which itself is activated by a user through the service agent deployed on the mobile device. Simply relying on traditional access control mechanisms on a target platform cannot satisfy these requirements since the decision factors (i.e., subject and object attributes) of these approaches are mostly static and pre-defined and cannot fit a *dynamic* computing environment.

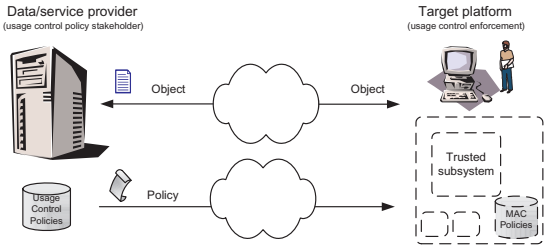

Figure 1. Distributed usage control. A trusted subsystem in client platform ensures the enforcement of usage control policies.

As usage control is naturally distributed, another challenge to enforce usage control policies is the trustworthy of the security enforcement mechanism. Typically, an access control decision is made and enforced by a reference monitor, which has the requirements of being tamper-proof, always-invoked, and small enough [5, 11] — which is relatively easy to achieve at least in closed systems. Note that in trust management systems, policy enforcement is still within the stakeholder's control domain. However, as objects or services are deployed to different domains from their stakeholders, a mandatory requirement for usage control is the trustworthy enforcement of security policies by the reference monitor. Here, through trustworthiness, a stakeholder needs to ensure that (1) all factors for usage control decisions can be obtained and their information (e.g., attribute values or environmental conditions) are authentic, (2) correct decisions are made based on these factors, (3) the reference monitor enforces access control decisions correctly, and (4) all accesses to a target object on a target platform have to go through the reference monitor. Overall, by a "trusted subsystem" we mean that it is expected to behave in a "good" manner and this manner can be verified by the policy stakeholder.

These requirements represent the essential security considerations in many distributed computing environments. For example, in our ongoing project, a healthcare service provider (e.g., a hospital) provides healthcare data to authorized service requestors (e.g., a physician). A physician or a nurse uses a desktop in a clinic to retrieve healthcare data of a patient from the service provider. To preserve the integrity and privacy of the healthcare data, the service provider typically requires that the data released to the client machine are correctly processed by authorized users only through the trusted healthcare application, and are not eavesdropped by any other hidden processes concurrently running on the client platform and during data transfers. For another example, for a data or service provider to deploy their value-added services on a mobile device, the runtime environment of the mobile device has to be trusted and satisfy some security requirements. For example, location-based services have been widely deployed and a user is allowed to use service only within a particular scope of a physical location.

Previous work on usage control focus on high level policy specifications and conceptual architectures [15, 21, 22, 27], while enabling mechanisms are mainly relied on digital rights management (DRM) approaches. However, DRM mechanisms cannot support general attributes and trusted enforcement in ubiquitous environments. Most importantly, DRM approaches cannot provide an overall solution for usage control in open and general-purpose target platforms, since they usually rely on software-enabled payment-based enforcement in relatively closed environments, e.g., through a media player by connecting to a dedicated license server. Another intuitive solution is to use cryptography algorithm. For example, a stakeholder can encrypt a target object such that it only can be decrypted on a target platform with a particular application. Fundamentally, this has the same problems as the DRM approach, since a typical DRM scheme relies on encryption/decryption with a unique key shared be-

tween a client and content server [1, 2]. Particularly, cryptography alone cannot protect the key during the runtime on a target platform such as to build a trusted subsystem [20]. For example, malicious software can easily steal a secret by exploring some vulnerability of the protection system, either when the secret is loaded in some memory location, or when the secret is stored locally.

In this paper we make the first step towards a general framework for a trusted usage control enforcement in ubiquitous computing environments. We start with an analysis of the security requirements in the usage control problem. Within these high level requirements, we identify mandatory components to build a trusted subsystem in a general client platform for usage control policy enforcement. This subsystem needs a strong protected environment such that security mechanisms in the client platform should respect its *autonomy*. Specifically, this trusted subsystem should have the final and complete control over its resources (e.g., an object downloaded from a remote stakeholder), and the security mechanisms of the local platform cannot compromise or bypass this control. Also, information flow between this subsystem and any others on the target platform has to be controlled, if allowed. For these purposes, we claim that mandatory access control (MAC) is necessary. Furthermore, to achieve the assurance of policy enforcement as aforementioned, the integrity of the subsystem has to be verifiable by the policy stakeholder. Thus, mechanisms like integrity measurement, storage, and verification are needed on such a target platform.

As one of the main contributions of this work, we consider the integrity of a subsystem in access control mechanisms. With this, not only traditional subject and object attributes are considered in access control decisions, but also the integrity of subjects and objects, and any other supporting components in a trusted subsystem. The overall goal of our approach is to build a "virtually closed" and trusted subsystem for remote usage control policy enforcement. Our work presented in this paper can be regarded as the enforcement model of usage control in PEI security framework [27].

The present paper is organized as follows. Section 2 presents the principles to build distributed usage control systems. We describe our general platform architecture to build a trusted subsystem in Section 3. Our prototype system is presented in Section 4. Related work is presented in Section 5. We eventually conclude this paper in Section 6.

2 System Design Principles

In order to enforce usage control in a trustworthy manner, we have identified a set of general design principles.

Requirement 1: Need high assured but usable security mechanism. Typically in usage control, objects are located out of the domain of a stakeholder such that high assurance of policy enforcement is desired. However, as usage control is such pervasive that it can happen in open and general-purpose platforms, a "usable security" mechanism is strongly desired to satisfy also the cost-effective objective. For example, leveraging a local host access control mechanism to enforce usage control policy is very desirable if the mechanism can be trusted to do the "right" thing. That is, the goal of pervasive usage control is not to provide a perfect solution for security but just to be "good-enough" [26].

Requirement 2: Need a comprehensive policy model. Traditional security systems distinguish policy and mechanism [17]. However, early policies such as Bell-LaPadula [7] and Biba [8] are too restrictive for convenient use within applications. They support simple policies such as one-way information flow but provide insufficient and inflexible support for general data and application integrity. Typically, usage control considers many constraints or conditional restrictions such as time and location as aforementioned. Traditional policy models cannot support these and usage control needs a comprehensive policy model to support the variants of such additional security requirements.

Requirement 3: Need MAC mechanism for trusted subsystem on a target platform. In discretionary access control (DAC), a root-privileged subject has the capability to violate the security configuration of the whole system such that the subsystem can be compromised either by a malicious user or software (e.g., a virus or Trojan horse). As evidenced by many security attacks, a virus or worm can obtain the root permission of a system by exploring some vulnerabilities, e.g., with buffer-over-flow attacks. Thus, mandatory access control (MAC) mechanism is needed. For example, with SELinux, one can label the applications and all resources of a subsystem with a particular domain and define policies to control the interactions between this domain and others for isolation and information flow control purposes.

Requirement 4: Need a policy transformation mechanism from high level usage control policies to concrete MAC policies. Typically, a stakeholder's policy is specified in different formats and semantics from those of the MAC policies on a target platform. For example, a stakeholder can be implemented as a Web Service, where a security policy is specified in XACML. This policy has to be transformed to a concrete policy that is enforced on a target platform, which follows its local MAC model. Thus, an efficient and convenient policy transformation mechanism is needed such that security properties are preserved during a transformation, i.e., the allowable permissions and information flows are the same in the policies before and after a transformation.

Requirement 5: Need security mechanism on operating system (OS) level. There has been long discussion that application level security alone cannot provide high assurance of

a system. For example, many web service based security protocols such as WS-Security [3] are built on credentials (public key certificates) between service providers and consumers. However, credential management and private key protection are critical problems for general-purpose computing platforms. Without OS level security mechanism, message-based security mechanisms cannot guarantee end-to-end security – integrity, confidentiality, and privacy can be compromised on OS level by software-based attacks (virus, spyware, worms, etc).

Requirement 6: Build trust chain for policy enforcement from system boot to application execution. The high assurance of a subsystem in a remote computing platform should origin from a root-of-trust, and then is extended to other system components upon which the policy enforcement mechanism is built. Typically, a MAC mechanism is implemented in the kernel of the OS on a platform. Thus, a trusted subsystem should include a trusted kernel while any other components booted before the kernel, such as the BIOS and the boot loader also need to be strictly trusted. To obtain the trust of the MAC mechanism in a subsystem, any other supporting components should also be trusted, including policy transformation and management, subject and object attribute acquisition, and the reference monitor itself. The fundamental goal of this trust chain is to achieve a trusted runtime environment for object access where the integrity of all related parts can be verified by a stakeholder.

Requirement 7: Build trusted subsystem with minimum trusted computing base. Related to the above requirement, to build a practical and usable trusted subsystem, a minimum trusted computing base (TCB) is desired. A TCB includes all the components in the trust chain for policy enforcement during runtime. A larger number of components in this chain results in higher costs both on system development and verification since each trusted component requires a detailed verification of the software implementation.

As policy model and formal specifications have been extensively studied in previous work, in this paper we focus on the policy enforcement issue of usage control. We propose a general platform architecture by following these principles. We have developed a prototype with emerging trusted computing technologies including a hardware-based root-of-trust. We leverage the MAC mechanism in SELinux for policy enforcement. Due to space limit we ignore the policy transformation mechanism in this paper.

3 Platform Architecture

A trusted subsystem is the foundation to enforce usage control policies. We define two phases for this purpose. First, a prerequisite for usage control is secure object and policy download. This requirement is two-fold. Before downloading, a stakeholder wants to verify that the request comes from an authentic subject on a target platform and the subject does have the permission to obtain an object. After downloading, the subject needs to verify the integrity and authenticity of an object and its policy. Secondly, during the runtime of processing a target object, a trusted subsystem ensures that usage control policies are enforced implying that only authorized processes and users can access the data, and interactions between running processes are controlled such that correct information flows are preserved. Typically, authorized accesses from "known good" processes and users ensure the confidentiality and privacy of protected objects, and information flow control ensures the integrity of the data.

3.1 Trusted Subsystem Architecture

Our trusted subsystem includes a root-of-trust, trust chain, and a policy transformation and enforcement mechanism, and also a runtime integrity measurement mechanism. Figure 2 shows our target platform architecture to enforce usage control policies. The hardware layer includes a Trusted Platform Module (TPM), a Core Root of Trust Measurement (CRTM), and other devices. The TPM and the CRTM provide the hardware-based root-of-trust. Similar to trusted or authenticated boot [6, 12, 25], the booting components of the platform, including BIOS, boot loader, and OS kernel, are measured and their integrity values are stored in particular Platform Configuration Registers (PCRs) of the TPM. Specifically, according to the TCG specification [28], the CRTM is the first component to run when the platform boots. It measures the integrity of the BIOS before the BIOS starts, which in turn, measures the boot loader and hereafter the kernel and kernel modules, recursively. Along this booting and measurement sequence, particular PCR(s) are extended with the measured values, and the result is denoted as PCR_{boot}. The TPM guarantees that PCR_{boot} is reset once the platform re-boots.

Upon a user's request on the target platform, a client application (e.g., a healthcare client software) is invoked to communicate with a data owner/provider to obtain an object. At that same time, a policy can be downloaded by the client application from a stakeholder, which can be the same as the data provider or different. For example, a data provider can delegate its policy specifications to a security service provider, which is the policy stakeholder when an object is downloaded and processed on a client platform.

When a usage control policy (e.g., an XACML policy file) is downloaded from its stakeholder, it is transformed by the policy transformation service to a MAC policy such that they can be enforced by the reference monitor. The client application is the target process that can manipulate the object and is to be protected by MAC policies. Also,

MAC policies should include rules to control accesses to the object from other applications and any configurations for the client application and the overall security system (e.g., local security policy management).

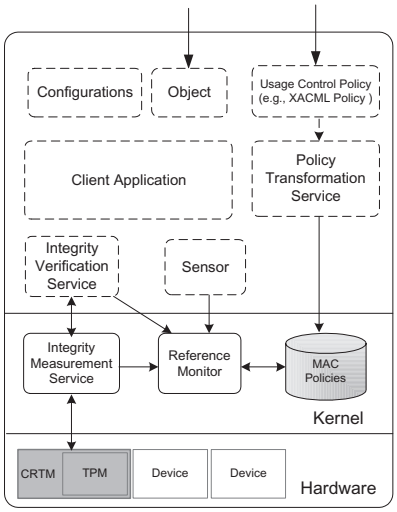

Figure 2. Platform architecture for usage control policy enforcement.

As aforementioned, usage control policies typically include environmental authorization factors such as time and location. A sensor is the component that reports these environmental information and thus can be considered by the reference monitor. For example, in a mobile application where a service can only be accessed in a particular location, the sensor reports the physical (e.g., through a cellular network provider or GPS) or logical (e.g., through a Wi-Fi access point) location of the device, such as home, office, airport, etc.

In the kernel level of the platform, the reference monitor captures an access attempt to the object and queries the MAC policies before allows the access. A fundamental requirement for the reference monitor is that it has to capture all kinds of access attempts, from the storage space of the local file system to the memory space of the object. Also, the reference monitor controls the interactions between the client application and others, locally and remotely, and especially according to the loaded MAC policies.

The integrity measurement service (IMS) is a mandatory component in a trusted subsystem, which starts right after the kernel is booted. The main function of the IMS is to measure other runtime components which consist of the TCB to enforce usage control policies. All measured events and the integrity values are stored in a measurement list and the corresponding PCRs are extended accordingly. Particularly:

- The reference monitor is measured after the kernel is booted.
- The client application, object, and its configurations are measured right before the client application is invoked.
- The integrity of the usage control policy, policy transformation service, and the sensor are measured when they are invoked and just before their execution.
- MAC policies are measured when they are loaded, either, when the platform boots or during runtime (i.e., loaded by the policy transformation service).
- Any other applications or services that need to communicate or collaborate with the client application are measured before they are invoked.

In general, in order to only allow accesses to target objects from authorized applications, and control information flow between this applications and others, IMS should measure not only the policy enforcement services such as policy transformation and platform sensor, but also all other applications that are allowed to interact with the sensitive client applications running on the same platform.

As part of the policy enforcement, the integrity verification service (IVS) verifies corresponding integrity values measured by the IMS and generates inputs to the reference monitor. As a typical example, the client application can only access the target object when its "current" integrity corresponds a known good value, where the current integrity is the one measured by the IMS.

Note that although we use data objects (e.g., files) through this paper, our usage control mechanism is applicable to other types of objects such as messages and streams. The essential requirement for the object is that its authenticity and integrity can be verified after downloaded, such that, as an input for the application on a client platform, the initial state of the platform can be trusted.

3.2 Secure Object and Policy Download

Before any access, the target platform obtains the object and the XACML policy file from the policy stakeholder through authentication and attestation protocols. Without loss of generality, we assume that the access request is generated by the client application from the target platform (cf. Figure 2). Policies can be defined in the target platform to confine that only the dedicated client application (e.g., the healthcare client software or a mobile service agent) can generate an access request to the stakeholder.

Upon receiving the request, the stakeholder generates an attestation challenge to the target platform. An attestation

agent on the target platform collects a set of integrity values measured by the IMS, signs them with an attestation identity key (AIK) of the TPM, and sends them back to the stakeholder. The integrity included in this response consists of PCR_{boot} and all mandatory components for the runtime policy enforcement, including the reference monitor, policy transformation service, IVS, sensor, and system configurations. After positively verifying these integrity values, the stakeholder decides that the data can be released, and the corresponding usage control policy can be generated. The integrity of the policy to confine which application can generate access request to the stakeholder can also be measured and attested.

Note that although we assume each object is associated with a usage control policy logically, a policy can also be associated with an application or the type of objects or services. For example, the healthcare application aforementioned can use the same policy for all patient records of a particular type of diseases.

3.3 Runtime Policy Enforcement

When a usage control policy is received, it is transformed to local MAC policies on a local platform. The policies specify the following factors for secure information processing during runtime. Firstly, the MAC policies should confine the users that can initialize applications to access certain target objects. For example, a patient's healthcare information only can be manipulated by a registered nurse, which is represented by a digital credential. Secondly, MAC policies should also specify the integrity of the client platform and the target application which allow an object access. The integrity information, typically, provides the assurance that the object is correctly processed, and there is no illegal information leakage. Thirdly but not lastly, MAC policies should consider the environmental restrictions by which an object can be accessed, e.g., the location of a mobile platform to access a service.

We now explain how these factors are considered in our trusted subsystem during runtime. With a TPM-enabled platform, all booting components are measured and their respective integrities are stored in the TPM, including the IMS in the kernel. When the kernel boots, the integrity of our reference monitor and other user space services are measured. Consider that an object and its usage control policy has been downloaded by a client platform and their integrity have been verified after downloading. The usage control policy is then transformed to a set of MAC policy rules that can be enforced by the reference monitor. These MAC rules specify all the security requirements including user, integrity, and environmental restrictions.

- When a user logins or attempts to access the object by invoking the client application, the user attributes are checked by the reference monitor based upon the authentication of the user to the system, for example, the role and necessary credentials of the user. When the client application is invoked by the user, it is measured by IMS before loaded to memory.

- During runtime, if the client application generates access requests to the target object, the measured integrity of the application is evaluated and verified by the IVS and the result is considered by the reference monitor.

- The sensor service monitors the environmental information of the computing device (e.g., location) and provides these also to the reference monitor for policy evaluation when an access happens on the platform with regarding to the policies. Whenever there is a change of any information specified in the policies, the new information is reported by the sensor service thus invokes the re-evaluating of the ongoing access.

With these mandatory components, we show that a general usage control policy can be enforced on a target platform with verifiable trustworthiness.

4 Implementation and Evaluation

4.1 Conditional SELinux Policy

As aforementioned, we have developed a policy transformation mechanism to transfer high level policies, specified by a stakeholder in XACML format, to low level policies specified by the MAC policy model inside the target platform. In our project, we use SELinux as our reference monitor, and usage control policies are realized via the conditional policies of SELinux. Figure 3 shows an example policy for a medical application. The policy is formed as an SELinux loadable policy module, where types and `allow` rules are defined within this module. The permission statement of `medicalApplication_t` is made conditional on the basis of application integrity and platform location. The boolean variable `integrity` and `accessLocation` correspond to those in the constrain expression. The overall semantics of this policy module is that, if predicate `(integrity && accessLocation)` is true, the permissions are allowed by SELinux; otherwise, only `getattr` is allowed, e.g., the file can be listed in the directory but cannot be opened.

Note that the policy in Figure 3 shows only a simple example of how to integrating integrity information with `read` file permission in SELinux. Other permissions, such as the executing of the medical application or its writing to

the object during runtime, or the integrity of other components in the platform, can be integrated with similar mechanism. Typically, the usage control policy defined by its stakeholder determines these configurations in SELinux.

```
1 policy_module(medicalApp,1.0)
2 require {
3     type fs_t;
4   }
5 type medicalObject_t medicalObject;
6 type medicalApplication_t;
7 allow medicalObject_t fs_t : filesystem associate
8 allow medicalApplicatoin_t fs_t : filesystem associate
9 If (integrity && accessLocation)  {
10        allow medicalApplicatoin_t medicalObject_t : file  {read getattr search}
11 } else {
12        allow medicalApplication_t medicalObject_t : file   {getattr}
13   }
```

Figure 3. Example SELinux loadable policy module with conditional policy.

4.2 Policy Enforcement Architecture

Figure 4.2 shows the overall architecture of our prototype. The left part of the diagram 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 transformation, integrity verification, and sensors. In the kernel space, the SELinux filesystem provides interfaces to allow user space services to set boolean values for conditional policies. When an XACML policy is transformed, a set of files are created in this filesystem and default values are set. The boolean values are updated by the user space services, e.g., with the result of integrity verification or location change. Whenever there is an access request, the SELinux security server obtains the current boolean values (if specified in the policy) and makes a decision.

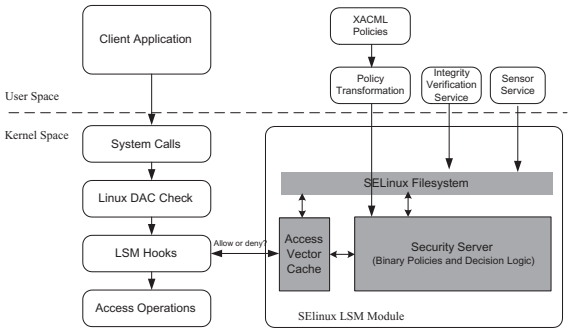

Figure 4. Usage control platform architecture via SELinux and conditional policies.

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 context of the file or the accessing subject changes, the access is revoked on the next read or write attempt [2]. With this feature, many flexible and dynamic security requirements can be supported, such as runtime integrity verification and location-based authorizations.

4.2.1 Integrity Measurement

Integrity measurement service is implemented by re-using some codes from IBM Integrity Measurement Architecture (IMA) [25]. Specifically, we re-use the `ima_main.c` and `drivers/char/tpm.c` in IMA. The measurement function is called within the SELinux hook function `file_mmap()`. Whenever, a new file is loaded into memory, the `file_mmap` is called for the verification of additional permissions by SELinux.

An independent configuration file called `/sys/kernel/measure` is maintained to indicate the files that are needed to be protected. Each entry in this file contains the file name and its absolute path. The measurement function searches the configuration file for the desired entry – passed as a parameter to `file_mmap()`. If the file passed as a parameter to the `file_mmap()` is found in `/sys/kernel/measure`, the same procedure is followed as done by the IMA for augmenting the hash to the kernel list. These measurements are then protected by the TPM.

4.2.2 Integrity and Location verification

The Integrity and location verification is done by two separate daemons. The integrity verification service (IVS) daemon is responsible for monitoring the integrity of all those files listed in `/sys/kernel/measure`. 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. 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.

The integrity verification service daemon retrieves the kernel list and the `/sys/kernel/measure` file periodically. More than one entry for an integrity protected file

[2] However, SELinux does not support access revocation to memory-mapped objects, e.g., memory-mapped file data and inter-process communication messages.

in the measured kernel list designates that the file is compromised. Based on the measurement list integrity verification, this daemon transits the corresponding boolean variables accordingly. For example, in our scenario, the IVS sets the `usr_share_medicalPolicyIntegrity` to false if the `/usr/share/medicalPolicy` has more than one entry in the kernel list.

Similarly, the sensor daemon monitors the location of the device, e.g., through the IP address or Wi-Fi access point of the device. If the device location changes, the daemon re-evaluates the location and based on a set of locations, sets the corresponding SELinux boolean variable.

4.3 Evaluation

Our implementation leverages IMA for integrity measurement. Therefore, on the kernel level, our system has the similar performance as IMA [25]. Our experiments show that it takes $5765\mu s$ to measure a medical record file and extend the PCR with the corresponding measurements. Like in IMA, this overhead is due to opening the configuration file, writing the measurement request and closing the configuration file. Further, the size of the files in our healthcare scenario is small, therefore, fingerprinting the files does not poses significant performance concerns for our implementation. We have taken these results on a desktop with 2.4 GHz processor and 1 GB of RAM.

On the user space, our system includes the extra integrity verification step to consider integrity in access control decisions. We measure the time to make an integrity verification and set the value for the boolean variable inside the SELinux filesystem, and the time to evaluate a single access control decision with SELinux, respectively. The data shows that it takes 95μs overall for integrity verification, transiting the corresponding SELinux boolean variables and accessing the protected file. Whereas, without integrity verification, it is just 88μs on average. This shows that the difference is not so much from the performance point of view. Note that the overhead of integrity verification is independent from the measured file size by IMS.

5 Related Work

A distributed usage control policy language and its enforcement requirements are presented in [15, 22]. Similar to our objective, their work targets on control over data after its release to third parties. However, the significant difference between this and our work is that our work relies on the underlying trusted subsystem of a platform, where the root-of-trust is built by a hardware TPM and extended to all mandatory components for policy enforcement. A comprehensive usage control policy model is proposed by Park and Sandhu [21]. However, it is basically a centralized approach and there is no concrete implementation mechanism existing for it yet.

Attestation-based remote access control [24] is proposed based upon the IBM integrity measurement architecture (IMA). Similar to usage control, an enterprise server deploys security policies on a client platform which are enforced based on the integrity status of the client platform. However, there are significant differences between this and our work. First, the objective of attestation-based remote access control is to filter the traffic origins from a client to a server while target objects (i.e., enterprise resources) are still located on centralized server. In usage control, the fundamental goal is to enable continuous control after an object is distributed. Second, based on the limitation of IMA, the policy enforcement in this work needs to verify all components loaded in a platform after booting, such that it is not practical to deploy it in very open and heterogeneous environments [23, 16]. Most importantly, our approach integrates IMA with SELinux with an augmented policy model. Moreover, we leverage the loadable policy module of SELinux so that we can build a relatively "closed" trusted subsystem by defining SELinux policies according to usage control requirements on remote platforms.

6 Conclusions and Future Work

Usage control focuses on the problem of enforcing security policies on a remote client platform with high assurance and verifiable trust. In this paper we present general security requirements for usage control and propose a general framework for this problem. The main idea of our approach is to build a trusted subsystem on an open platform such that a policy stakeholder can deploy sensitive data and services on this subsystem. We propose an architecture with a hardware-based TPM as the root-of-trust and consider integrity measurement/verification and other environmental restrictions in our MAC policy model. We have also implemented a real prototype system by integrating integrity measurement and SELinux. By leveraging the latest SELinux conditional policies and loadable policy modules, our approach enables verifiable assurance to build a relatively "closed" trusted subsystem for usage control.

References

[1] Fairplay. http://en.wikipedia.org/wiki/FairPlay.

[2] Windows media digital rights management (DRM). http://www.microsoft.com/windows/windowsmedia /drm/default.aspx.

[3] Web Services Security: SOAP Message Security 1.1. OASIS Web Service Security TC, 2004.

[4] M. Abadi, M. Burrows, and B. Lampson. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[5] J. P. Anderson. Computer security technology planning study volume II, ESD-TR-73-51, vol. II, electronic systems division, air force systems command, hanscom field, bedford, MA 01730. http://csrc.nist.gov/publications/history/ande72.pdf, Oct. 1972.

[6] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proc. of IEEE Conference on Security and Privacy*, pages 65–71, 1997.

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

[8] K. J. Biba. Integrity consideration for secure computer system. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass., 1977.

[9] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 164–173, Oakland, CA, May 1996.

[10] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), May 1976.

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

[12] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the ibm 4758 secure coprocessor. *IEEE Computer*, (10):57–66, 2001.

[13] M. H. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communication of ACM*, 19(8), 1976.

[14] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: assigning roles to strangers. In *Proc. of IEEE Symposium on Security and Privacy*, pages 2–14, 2000.

[15] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proc. of 10th European Symp. on Research in Computer Security*, September 2005.

[16] 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.

[17] B. Lampson. Computer security in the real world. *IEEE Computer*, (6):37–46, June 2004.

[18] B.W. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971. Reprinted in *ACM Operating Systems Review* 8(1):18–24, 1974.

[19] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proc. of IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[20] P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, J. Turner, and J. Farrell. The inevitability of failure: The flawed assumption of computer security in modern computing environments. In *Proceedings of the National Information Systems Security Conference*, October 1998.

[21] J. Park and R. Sandhu. The UCON_{abc} usage control model. *ACM Transactions on Information and Systems Security*, 7(1):128–174, February 2004.

[22] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Communications of the ACM*, (9):39–44, 2006.

[23] J. F. Reid and W. J. Caelli. Drm, trusted computing and operating system architecture. In *Australasian Information Security Workshop*, 2005.

[24] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of ACM Conference on Computer and Communication Security*, 2004.

[25] 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.

[26] R. Sandhu. Good-enough security: Toward a pragmatic business-driven discipline. *IEEE Internet Computing*, (1):66–68, 2003.

[27] R. Sandhu, K. Ranganathan, and X. Zhang. Secure information sharing enabled by trusted computing and PEI models. In *Proc. of ACM Symposium on Information, Computer, and Communication Security*, 2006.

[28] TCG TPM. Main part 1 design principles specification version 1.2, https://www.trustedcomputinggroup.org.