International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:10, 2013

A General Mandatory Access Control Framework in Distributed Environments

Feng Yang, Xuehai Zhou, Dalei Hu

***Abstract*—**In this paper, we propose a general mandatory access framework for distributed systems. The framework can be applied into multiple operating systems and can handle multiple stakeholders. Despite considerable advancements in the area of mandatory access control, a certain approach to enforcing mandatory access control can only be applied in a specific operating system. Other than PC market in which windows captures the overwhelming shares, there are a number of popular operating systems in the emerging smart phone environment, i.e. Android, Windows mobile, Symbian, RIM. It should be noted that more and more stakeholders are involved in smartphone software, such as devices owners, service providers and application providers. Our framework includes three parts—local decision layer, the middle layer and the remote decision layer. The middle layer takes charge of managing security contexts, OS API, operations and policy combination. The design of the remote decision layer doesn't depend on certain operating systems because of the middle layer's existence. We implement the framework in windows, linux and other popular embedded systems.

***Keywords*—**Mandatory Access Control, Distributed System, General Platform.

I. Introduction

HANDSET devices like smart phones and netbooks are very popular these days. There are two features of these devices. First, various operating systems are used in these devices because they are embedded essentially. Second, multiple stakeholders are involved like communication service providers, device manufacturers and the owners of downloaded applications. Operating systems security is the cornerstone of system security; access control mechanism has an important meaning for the operating system; however at present the majority of the operating system access control mechanisms are designed for a single system, and to implement access control mechanisms on other systems we need to re-implement it, which is a waste of a lot of work and makes the security configurations of different systems differs.

A general mandatory access control framework is proposed in this paper to avoid duplication of efforts. We use smart phone systems as the instance in this paper although the framework can be easily applied in other distributed systems.

The main work and contributions of this paper includes:

(1) Research of the differences of operating systems, as well as the middle layer design.
(2) Designing a general mandatory access framework for distributed systems.
(3) Implementing the framework in Windows, Linux and other operating systems.

The rest of the paper is organized as follows. In Section II we review the background and related works. We propose the framework design in Section III. We then present the implementation details and the overhead involved in our solution in Section IV. Discussion of the framework is presented in Section V. We conclude the paper in Section VI.

II. Background and Related Works

Several mandatory access control models have been proposed in the literature such as BLP [1], [2], BIBA [3], [4], RBAC [5], TE [6]. Generally, there are two phases in the access control. The first is assigning levels (BLP, BIBA), types (TE), roles (RBAC) or other labels to the new subjects and objects. The second is access enforcement which answerhs the query "Can subject S perform action A on Object O".

In smart phone systems, traditional access controls are static. Like Symbian [7] and Android [8], static policies should be loaded at install time. These systems do not consider the dynamic policy decision and multiple stakeholders. Obviously, the access control mechanism in one system is different from those in others.

A. Herzberg et al. extend existing RBAC mechanisms and present a mechanism that allows a business to define a policy to map accessed users to roles, based on certificates received from the user and collected automatically by the system [9]. Considerable advancements in the area of composing access control policies are achieved. We leverage the results in [10] and [11].

V. Rao et al. revise SE linux and propose a dynamic mandatory access control for multiple stakeholders [12]. However, it can only be applied in Linux and it is hard to be used in other operating systems.

We can see from above that a general mandatory access control framework that can be applied in multiple operating systems is needed.

Feng Yang is with the University of Science and Technology of China Hefei, China, (phone: 86-13913584736; e-mail: yfus@ mail.ustc.edu.cn).

Xuehai Zhou is with the University of Science and Technology of China Hefei, China.

Dalei Hu is with the University of Science and Technology of China Hefei, China.

III. Approach

A. Framework Architecture

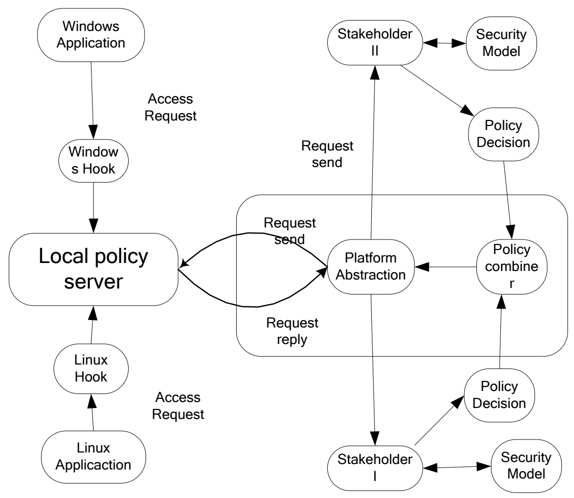

Fig. 1 Overview of the framework

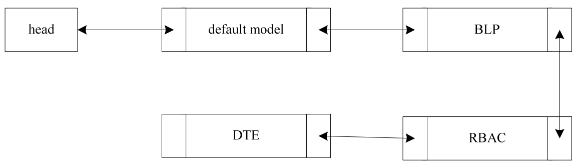

Fig. 2 Overview of the security model manager

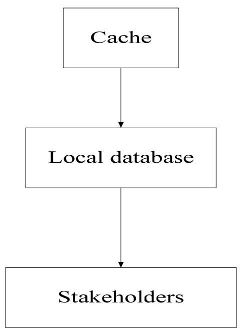

Fig. 3 Hierarchy of decision lookup

The architecture of our framework is shown in Fig. 1. The main components are the remote decision layer, the middle layer and the local decision layer. Each stakeholder makes decisions using various policy models. The middle layer takes charges of combining decisions and managing security contexts, OS kernel APIs and the set of operations. The design of remote decision layer could be applied in various operating systems due to the existence of the middle layer. The local decision layer takes charge of enhancing the access control. We have been implementing our framework in Windows, Linux and other operating systems.

B. The Remote Decision Layer

The remote decision layer has two functionalities. One is assigning labels to new objects and subjects; the other is replying to access control requests. Due to the existence of the middle layer, the remote server in our framework is universal.

1. Security Model Manager

We implement popular mandatory access control models including BLP, BIBA, RBAC and TE. Security models are organized as a double-linked list. Stakeholders assign labels to new objects or subjects and reply to access control requests according to the decision of the security model. Stakeholders use the security model manger to manage these models and the security model is organized as a double-linked list as shown in Fig 2. Each security model corresponds to a set of rules. A stakeholder receives requests from the middle layer and makes decisions according to the rules corresponding to the security model it uses. For example, a certain stakeholder chooses BLP(BIBA) as his/her security model. In BLP, a subject with higher level cannot read an object with lower level. If the stakeholder receives a request from the middle layer like (<scontext,10>, <ocontext,7>, read) which means a subject with level 10 wants to read an object with level 7. According to the rules in BLP model, the stakeholder denies the request.

2. State Information Transferred to the Server

The information carried by subjects can help stakeholders make decisions. The information includes certifications that subjects carry, temporal states of subjects and conflicts sets that subjects belong to, etc. In the RBAC model, applications may carry some certificates from issuers that are either known in advance or provide sufficient certificates to be considered a trusted authority [assigning roles to strangers]. Some polices are time-related. For example, telecom operators may restrict some applications to conserve bandwidth during peak hours. Some devices in the smart phones like microphone and camera are exclusive. If the exclusive device is used, the visit to the device should be denied directly. The information is useful for stakeholders to make decisions.

C. The Middle Layer

The middle layer receives requests from local servers and sends requests to stakeholders using standard forms. After that the middle layer receives decisions from stakeholders and combines the decisions. Final decisions are sent to local servers.

1. Policy Combiner

We leverage results already obtained in [10] and [11]. There are three ways for policy combination as follows.

(1) Intersection (∩) – The final decision is “allow” only if all policy decisions are “approve”
(2) Union (∪) – The final decision is “allow” if one policy decision is “approve”.
(3) Difference (–) – “A – B” means the final decision is only accordance with A.

To resolve conflicts, several rules can be used.

(1) Strict – The final decision is “deny” if any policy decision

is "deny".

(2) Majority – The final decision is "allow" if more than half decisions are "allow".

(3) Priority – The final decision is only accordance with the decision that made by the stakeholder with the highest priority.

(4) Weight – Each stakeholder i has a weight number w[i]. The sum of stakeholders' weight numbers whose decision is "allow" is A and the sum of stakeholders' weight numbers whose decision is "deny" is B. If A>B, the final decision is "allow"; otherwise, the final decision is "deny".

2. The Platform Abstraction

In order to apply our framework into various operating systems, we design a platform abstraction module composed of three components.

(1) Security Context Manager

Although operating systems are different from each other, there are processes, threads, files, signals, sockets, etc in most operation systems. These common concepts are the basis of context abstraction. As it should be, each OS has some specific concepts like superbloc, inode in Linux. We consider general concepts as well as specific concepts and design a context manager which manages a context set. The set is composed of most existing contexts in popular operating systems and an interface is designed to add new contexts. Meanwhile, forms of contexts are different in different security models. For example, subjects and objects are related to some security levels in BIBA and are related to some roles in RBAC. We consider these factors and construct a subset for each security model.

(2) OS API Abstraction

Kernel APIs are different in different operating systems. We design a set composed of OS APIs in the middle layer. The APIs include lock_operation, input, output, atomic operations, fork, etc. Interfaces are provided to local systems. Each system could communicate with the middle layer as long as it implements these interfaces. For example, when a new process B is created by A in Linux, the API fork() is invoked. The fork() is in the set of APIs in the middle layer. Therefore, it takes B as A's child and sends the information to stakeholder for assigning labels to the new process B.

(3) Operation Set

Operation sets are different in various systems. Although most operations are equivalent in different systems, they maybe have different names. We design a superset of operations can be done in most systems such as "read", "write", "create". The security server doesn't care what are stored in the operation set.

D. The Local Decision Layer

1. Architecture of the Local Server

In Fig. 3, we show the hierarchy of the decision lookup in which the local server includes the policy cache and the local policy database. Cache is used for easy revocation and fast lookup. More details can be found in the next section.

Enhancement of the access control in an operating system differs from that in other operating systems. Generally, a hook is inserted when a subject needs to visit an object and a request is sent to the local server or the remote server. The access is approved or denied according to the reply information.

2. Policy Addition and Revocation

A two-phase insertion is performed when updating the local policies. The permission replied from the policy server is inserted into the cache instead of modifying the local policy database directly. When the permission is visited more than N times (N is a predetermined parameter), it is inserted into the local policy database. The reason is to save space and revoke more easily. Time stamps can be made to limit the period of time that the permission lasts. Revocation is the recalling of a grant. In telecommunication systems, a large number of paid services are provided. Therefore, revocation should be done when users are out of credit. Meanwhile, users installed and uninstalled applications downloaded from internet all the time and revocation is an important aspect of the access control framework. Therefore, we take revocation into account in our framework. For policies just in cache, revocation means simple cache invalidation. For policies in the policy database, revocation is a little difficult. In this case, the module in the database should be removed and a new module is reinstalled.

IV. Implementation & Evaluation

A. Linux

We use the LSM module in the kernel 2.8.5 to implement the os-related part on linux and modify it to keep accordance with our framework. Both subjects and objects own a domain for security in LSM. The index of each security context is stored here and it is linked to the subject or object when it is created. For example, an index of task_security_struct is added to the security domain when a new process is created. In LSM, a hook is inserted when an access request(creating, reading, writing, etc) occurs.

B. Windows

There is not a ready security framework for mandatory access control in windows like LSM in linux. Therefore, we implement the OS-related part of the framework in Windows NT. Security contexts are managed in two ways: first, an index is added if the structure of a subject or object has enough unused fields. Second, we maintain a list for associating the subject or object with the corresponding security context. Extended attributes exist in NTFS. The index of the security context of a file is stored as an extended attribute. For process, we define a data structure called _PROCESS_ENTRY which represents the description of a specified process. The definition of _PROCESS_ENTRY is as follows:

```
typedef struct _PROCESS_ENTRY {
  struct _PROCESS_ENTRY* pre;
  struct _PROCESS_ENTRY* next;
  ULONG processID;
  PEPROCESS eProcessInfo;
  PVOID processSecurityContext;
  } PROCESS_ENTRY, *PPROCESS_ENTRY;
```

The process Security Context could store the structure of security context. In our implementation, we use the double linked list in which each element corresponds to a security context.

C. Other Embedded Operating Systems

To make our framework more general, we implement it on a number of embedded operating systems. In this section, we illustrate the implementation on RTEMS (Real-Time Operating System for Multiprocessor Systems). RTEMS is a real-time executive which provides a high performance environment for embedded applications on a range of processors and embedded hardware. Details about RTEMS can be found in [13]. We hook the operations on process, file and communication. Table II illustrates the hook points we set on process operations. The policy cache has 512 items, each of which is 20B. Adding the control structure, the overall space that the policy cache occupies is not more than 11KB which is acceptable in RTEMS.

More details about the implementations can be found on our website [14]. Please contact the authors for access permission.

D. Performance

Introduction of our framework into Linux is relatively easy because of the existence of LSM. Meanwhile, we could compare the performance of our implementation with SELinux directly. The hardware used was a Dell Optiplex 755 machine with Intel Pentium Dual Core processor @2.33 GHZ. We use the LSM module in the Linux 2.8.5 kernel to implement the local server and enhancing mandatory access control.

We record the number of cycles needed to enhance mandatory access control when the cache hit and missed. During the measurement, the network round trip delay is recorded independently because it varies according to the network environment. We can see from Table I that the bottleneck is the network round trip delay. The advantages we obtain from the local cache include improved performance and the minimum overhead incurred by the framework is 0.302µs which is acceptable. The real performance depends on the cache size, cache replacement and local policy database size, etc, and we leave the optimization as future works.

TABLE I
PERFORMANCE EVALUATION FOR ACCESS CONTROL

Processor Speed	2.33 GHZ
SELinux	703 cycles = 0.302 µs
Referring to Cache	703 cycles = 0.302 µs
Referring to local server	1271 cycles = 0.545 µs
Referring to stakeholders	1546 cycles = 0.664 µs
Network round trip delay	1.7 ms

TABLE II
PROCESS HOOK POINTS IN RTEMS

Rtems_task_create()
Rtems_task_start()
Rtems_task_restart()
Rtems_task_delete()
Rtems_task_suspend()
Rtems_task_resume()
Rtems_task_set_priority()
Rtems_task_get_priority()

V. DISCUSSION

A. Auditing

Permission requests can be logged and uploaded to the remote server at reasonable intervals. An auditing and intrusion-detection modular can be easily coupled with the existing server. The outbreak of viruses could affect many smart phones and cause noticeable changes in their behavior. A behavior-detection module in conjunction to the proposed framework is our future work.

B. Offline Operations

If the user is disconnection from the remote server, the cache and the local database is refer to. If the corresponding policy cannot be found in the cache and the local database, the access is denied in our framework. For clarity, the user only takes charge of enforcing mandatory access control and doesn't participate in the policy decision process in the proposed framework. However, the framework can be easily used in the situation that the user is one of the judgers. In case of disconnection from remote servers, the user is referred to if the corresponding policy cannot be found in the cache and the local database.

C. Dynamic Properties of the Mechanism

In ordinary mandatory access control mechanisms of smart phone systems, the default "deny" policy is used. A new application cannot be executed unless registered during the installed period. In our mechanism, when the applications need to visit some critical resources, the local server or stakeholders are referred to. The mechanism is dynamic because stakeholders can generate policies during the execution process and the related information is provided to stakeholders for reference. Meanwhile, an application's unused privileged functionality doesn't affect its operation.

VI. Conclusion

In this paper, we present a general mandatory access control framework in distributed environments. The framework is universal which means that it can be applied on various operating systems. The main feature of the framework is the middle layer which communicates with local servers and stakeholders. The framework is support with multiple operating systems, multiple stakeholders. Dynamic decisions and easy revocation are obtained in the framework. We implement the framework on Windows, Linux, Rtems, etc. The performance overhead is acceptable as the experiments show.

Acknowledgments

The authors would like to thank associate professor Songwu LU in UCLA for his constructive advice. This work was supported in part by Natural Science Foundation of China.

References

[1] L. Lapadula 1996. Secure computer systems: a mathematical model. MITRE Technical Report, Vol I.
[2] L. Lapadula 1996. Secure computer systems: a mathematical model. MITRE Technical Report, Vol II.
[3] K.J. Biba 1977. Integrity Considerations for Secure Computer Systems:[ESD-TR-76-372]. Electronic Systems Division.
[4] C.E. Landwehr. 1981. Formal Models for Computer Security. ACM Computing Surveys, 13(3).
[5] S.R. Ferraiolo DF, S. Gavrila. 2001. Proposed NIST Standard for Role-based Access Control. ACM Transactions on Information and System Security.
[6] W.E. Boebert, R. Y. Kain. 1985. A Practical Alternative to Hierarchical Integrity Policies. In Proceedings of the 8PthP National Computer Security Conference.
[7] Symbian Limited. Symbian OS – the mobile operating system. HTUhttp://www.symbian.comUTH, 2006.
[8] W. Enck, M. Ongtang, and P. McDaniel. Automated Cellphone Application Certification in Android. Technical report, Pennsylvania State University, 2008.
[9] A. Herzberg, Y. Mass, J. Michaeli, Y. Ravid, D. Naor. 2000. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In the Proceedings of the 2000 IEEE Symposium on Security and Privacy.
[10] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. 2002. An Algebra for Composing Access Control Policies. HACM Transactions on Information and System SecurityH.
[11] H. H. Hosmer. 1992. Metapolicies II. In Proceedings of the 15PthP National Computer Security Conference.
[12] V. Rao, T. Jaeger. 2009. Dynamic Mandatory Access Control for Multiple Stakeholders. In the Proceedings of 2009 HSymposium on Access control Models and TechnologiesH.
[13] Rtems HTUhttp://www.rtems.com/UTH.
[14] http://web-b.embedded.ustcsz.edu.cn/projects/PFAC.