User's authorization in singularity operating system

Rami Matarneh[†], Suha Al_Jubori[††]

[†]Faculty of Administrative and Financial Sciences, Al-isra private university, Amman, P.O. 11622, Jordan
[††]Faculty of Administrative and Financial Sciences, Al-isra private university, Amman, P.O. 11622, Jordan

Summary
We describe a new design for authorization in operating systems. In this design two additional units are introduced, Certificate Authority which provide certificates for authorization for all participants and Access Controller which is responsible for making access decisions, and we describe the implementation of our design and its performance in the context of Singularity operating system.

Key words:
Access Controller, Certificate Authority, Access Control Lists, Singularity

1. Introduction

One of the main decisions in designing any operating system is the choosing of security model and access control.
Access control refers to the action of deciding which operations are permitted and which operations are not permitted depending on the access rights the requesting principal has. In traditional design Access Control Lists (ACL) are used to do this.
As input to each access decision, the identity of a principal is presented, the identity of an object (system resource or data protected by the system), the specific operation that the principal requests on the object; and depending on ACL which kept with each object for each possible operation decision occurred. This means that each object in the system must has a list consists of either a principals and their legally operations or identifiers for groups. A group, in turn, consists of either principals or identifiers for further groups [11].
This paper argues that the traditional design is weak from the point of the amount of search needed when making any decision, and in a dynamic system these decisions needed to be done very frequently and one can imagine the search needed.
As a remedy, this paper propose a design which shrink the search needed by working in an opposite way that is instead of providing an ACL for each object, certificates will be provided for authorization of all principals registering legal operations that principal can do. We demonstrate our design in the context of the Singularity operating system and present a security model that takes into account the fundamental aspects of Singularity such as the channel abstraction, application manifests, and software isolated processes.

2. Singularity

The Singularity project combines the expertise of researchers in operating systems, programming language and verification, and advanced compiler and optimization technology to explore novel approaches in architecting operating systems, services, and applications so as to guarantee a higher level of dependability without undue cost [7].
A key aspect of Singularity is an extension model based on Software-Isolated Processes (SIPs), which encapsulate pieces of an application or a system and provide information hiding, failure isolation, and strong interfaces [4]. SIPs do not rely on memory management hardware for address space protection as is in most modern operating system. Instead each SIP has a software protected "object space". Static analysis, type safety and other language features are used to guarantee at compile time that code within a SIP cannot access memory that does not belong to it. Software protection of processes removes much of the cost associated with context switches in a hardware based system.
All inter-process communication in Singularity is done via communication channels. These channels are a first class abstraction which is managed by the kernel and supported explicitly by the Sing# language. Because dependability is the primary goal of Singularity, shared memory between processes is not supported [2].
From the security point of view in singularity, applications are security principals, they reflect the application identity of the current SIP, an optional role in which the application is running, and an optional chain of principals through which the application was invoked or given delegated authority. Users, in the traditional sense, are roles of applications (for example, the system login program running in the role of the logged in user). Application names are derived from *Manifest-Based*

Manuscript received February 5, 2009
Manuscript revised February 20, 2009

Program (MBP) manifests which in turn carry the name and signature of the application publisher. SIPs are associated with exactly one security principal. To support this usage pattern, we allow delegation of authority over a channel to an existing SIP. All communication between SIPs occurs over channels. From the point of view of a SIP protecting resources (for example, files), each inbound channel speaks for a single security principal and that principal serves as the subject for access control decisions made with respect to that channel [stack]. ACLs are patterns against which principal names are matched.

3. Access control list

An access control list (ACL) in computer security, is a list of permissions attached to an object [12]. The list specifies who or what is allowed to access the object and what operations are allowed to be performed on the object. In an ACL-based security model, when a subject requests to perform an operation on an object, the system first checks the list for an applicable entry in order to decide whether to proceed with the operation. This needs many steps to allow or reject the request. Figure 1 illustrates the required steps to perform one request over one object.

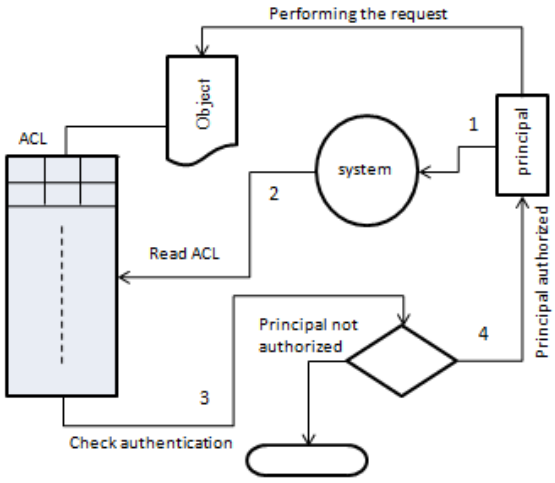

Fig. 1 User authorization in classical access control list

4. File system ACLs

Access control list is a data structure, containing entries that specify individual user or group rights to specific system objects, such as a program, a process, or a file [10]. These entries are known as access control entries (ACEs). The privileges or permissions determine specific access rights, such as whether a user can read from, write to, or execute an object, each entry in the list specifies a subject and an operation, the entry (user1, read) on the ACL for file MY_FILE gives USER1 permission to read file MY_FILE.

In some implementations an ACE can control whether or not a user, or group of users, may alter the ACL on an object [6], which is in our opinion considered not a safe behavior, and our approach depends on the fact that users and groups can't manage ACLs, so that we assign a certificate for every subject or principal which is look like as ACL but in reversed direction and label for every object.

The following rules explain the basics of the proposed model:

- System divided into groups, and every user belongs to only one group in the system
- Every group has its **certificate** GC, which is contain all rights or permissions that group can perform
- Every user has its **certificate** UC, which is contain all rights or permissions that user can perform
- User permissions include his permissions and his group permissions UC=UC+GC
- Every object has its **label** L, which is contain one field (the owner)
- Any principal can read its certificate, but it can't modify it
- Certificate creation and modification is a responsibility of a **certificate authority unit**
- Any principal can grant rights to and revoke rights from other principals through **certificate authority unit**
- **Access control** load a copy of user certificate when user login to the system, and keep track of any changes at any time for any certificate, using Boolean flag that indicates any changes occurred in the system.
- All operations in this model performed directly through **access controller** without any additional checks, except DELETE operation, it must be checked by comparing the object's label with the principal ID (to ensure that only the owner can delete the object)
- **Processes, certificate authority unit and access controller** are all run in its own SIP and communicate over channels [4], in line with singularity model

5. Certificate creation and its characteristics

When account or group created by system administrator, the **certificate authority** in its turn creates account certificate (user certificate or group certificate), at the outset the certificate is empty. In case object creation the **certificate authority** creates LABEL (object label L) attaches it to the object and fill it with the owner ID. The certificate contains the following fields:

- Permission type (read, write, execution …)
- Object (Intended object)
- Sequence (the sequence of users that delegate the permission)
- Delegation (Boolean value, **true** if delegation allowed and **false** otherwise)

Permission type	Object	Sequence	Delegation

Fig. 2 Certificate structure

When user login to the system and creates an object all permissions must be added into its certificate, we prefer to add permissions as a separate entry, the sequence field value is set to **OWNER** to indicate its ownership of the object, and so if **user1** create **file1** then, user's certificate must look like:

Permission type	Object	Sequence	Delegation
read	file1	Owner	true
write	file1	Owner	true
execute	file1	Owner	true

Fig. 3 Certificate with values

If user1 want to read or write file1, it requests the specific operation through access controller, access controller in its turn compares the request with user' certificate (which is already loaded by access controller when user1 logged into the system), if such permission found access controller allows user to perform the request, otherwise rejects it.

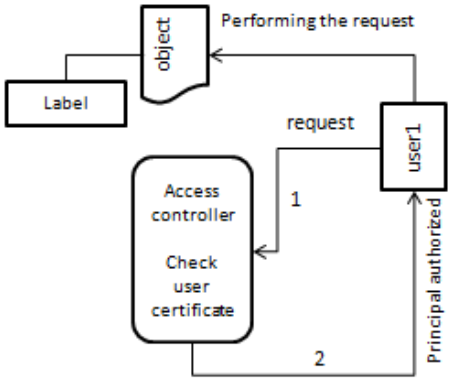

Fig. 4 User authorization sequence

To compare this model with classical ACLs, assume user1 want to perform read or write on 50 different objects, in ACLs it means that system needs to open and check 50 different ACLs, and every operation requires 4 steps (see figure 1) in total, system needs 200 steps to perform the task, in our model the system needs 100 steps to perform the request. Taking into consideration that step 2 in figure-1 may take long time comparing with other steps, which is not found in our model. At this time user1 can invoke permissions to a specific user or group, or revoke theme. In the following sections we discuss how our model manages these operations.

6. Invoke permissions

Any user can invoke permissions to other users in his group or in other groups, as follow:

1. *username@groupname* INVOKE *permission on objectname to username@groupname || groupname* +D

Where +D means delegation allowed.

2. *username@groupname* INVOKE ALL on *objectname* to *groupname* || *all* +D

Where *all* means all username in all groupname.

To understand how our model works, assume user1 want to invoke read permission to user2 in group2
User1@group1 INVOKE read on file1 to user2@group2
Access controller fetch the request and checks if user1 has such permission if true, then it sends the request to certificate authority, which is modify the specified user's or group's certificate and alter group's flag to true and signal access controller that some changes had been occurred in users or groups permissions, access controller check all flags and refresh certificates for all groups

where flag equals true, then set all flags to false (see figure 6).

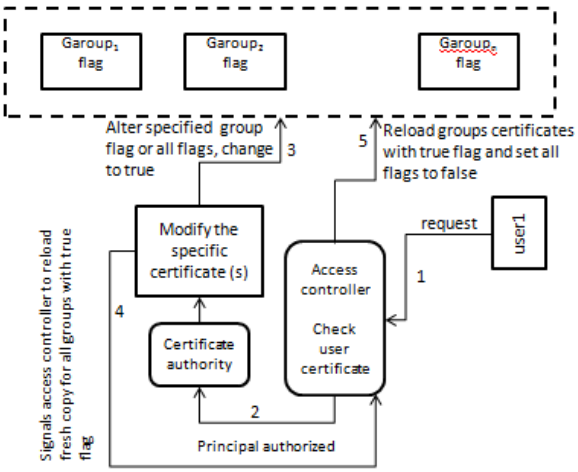

Fig.5 Permissions invoke or revoke sequence

7. Delegation

Any user can invoke permissions to other users or groups if delegation field in its certificate is set to true for the specific permission, for example in figure 6 user2 can delegate user 3 to read file2 but can't delegate to write it, because delegation allowed on read and not allowed on write (figure 6).

Permission type	Object	Sequence	Delegation
read	file2	user2	true
write	file2	user2	false

Fig.6 Certificate with values that can be delegated

To understand the whole picture, how users can delegate permissions and how to update the sequence field for every new user in the sequence. Assume the following scenario:

1. User1 has file1 (the owner).
2. User1 invoke the permission *read (file1)* with delegation to user2, user3 and user4.
3. User2 invoke the permission *read (file1)* with delegation to user5.
4. User5 invoke the permission *read (file1)* with delegation to user6 and user7.
5. User7 invoke the permission to user10.
6. User3 invoke the permission *read (file1)* with delegation to user8.
7. User8 invoke the permission *read (file1)* with delegation to user9.
8. User9 invoke the permission *read (file1)* to user11.

Figure 7 illustrates this scenario, where every user in the tree has a sequence of users who delegated him the permission.
We stress not to allow repetition of the same permission if it is already found, so if user7 for example *invoke the read permission on file1 to user6 or user9*, such request should be rejected by access controller, because user6 and user9 already have such permission, dotted lines in figure7 represents this case.

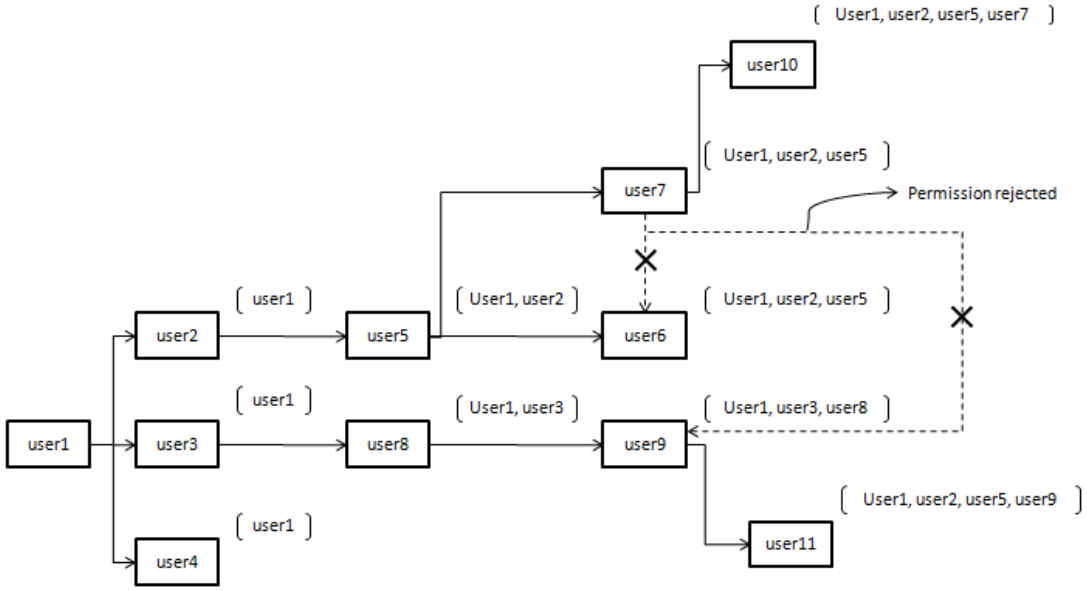

Fig. 7 Illustration of delegation scenario

8. Revoke permissions

Any user, who invoked permission to other user, can revoke it. When permission revoked from the user the same permission will be automatically revoked from all users who have obtained the permission from.
Access controller compares the username who want to revoke a specific permission from another user or group to the user's or groups sequence field, if found it will transfer the request to certificate authority unit to finish the task, otherwise will reject it.
To understand how access controller can take a decision to accept or reject a request to revoke permission from a specific user. Depending on figure 7, assume user2 requests the following independent requests:

1. REVOKE ***read*** on file1 from user10
2. REVOKE ***read*** on file1 from user8
3. REVOKE ***read*** on file1 from user5

First request will be accepted because **user2** is an element of ***user10 sequence*** set (see fig. 7).
Let us explain how access controller accepts the request from user2:

- Access controller find **user10** sequence field, which is: *[user1, user2, user5, user7]*
- Check if *user2 is an element of [user1, user2, user5, user7]*
- The answer is TRUE, so the request accepted.

Second request will be rejected because **user2** is not an element of ***user8 sequence*** set.

- Access controller find **user8** sequence field, which is: *[user1, user3]*
- Check if *user2 is an element of [user1, user3]*
- The answer is FALSE, so the request rejected.

Third request will be accepted because **user2** is an element of ***user5 sequence*** set.

- Access controller find **user5** sequence field, which is: *[user1, user2]*
- Check if *user2 is an element of [user1, user2]*
- The answer is TRUE, so the request accepted.

After the execution of the third request, access controller will ask certificate authority unit to revoke the same permission from all users who got the permission from **user5**, which is in this case will be **user6**, **user7** and **user10** (see fig. 7).
Other security approaches don't take in consideration the delegation sequence, which is mean; we can't know the sequence of delegates. So if we use other security approaches to execute the following:

REVOKE ***read*** on file1 from user5

It will revoke ***read*** permission from **user5** only, although **user6**, **user7** and **user10** got the permission from him,

which will lead us to a complex situation when we find number of users has permissions on some objects in the system and we can't know how they got such permissions. Such case can make a security vulnerability in the system can't be avoided.

9. Conclusions

In this paper a security model for authorization in singularity operating system introduced. This model based on providing certificates for very user which state all access rights given to that user and according to checks done through these certificates accesses is granted. The main novelty of the work is reducing the amount of search needed when using Access Lists which registered all operations each user could do to each object and each time an access decision made the whole list must be searched, by imagining the number of objects may exist in a system one can expect the amount of search needed. However using certificates as suggested in this work the situation reversed is that instead of searching the whole Access list each time a user needs to implement an operation to an object, we need only to check the user certificate to make our decision and by comparing the number of users to the number of objects in any system one can find that in most cases the number of users in a system are less than number of objects and as consequence the amount of search needed in our design reduced than what is needed in the traditional way.
We believe that this design allows for authentication and access control in a modern operating system, suitable for the more stringent requirements of a modern security posture in a world with diverse software.

References

[1] Galen C. Hunt, James R. Larus, Singularity: Rethinking the Software Stack, ACM SIGOPS Operating Systems Review, vol. 41, no. 2, pp. 37-49, Association for Computing Machinery, Inc., Apr. 2007

[2] Galen Hunt, Chris Hawblitzel, Orion Hodson, James Larus, Bjarne Steensgaard, Ted Wobber, Sealing OS Processes to Improve Dependability and Safety, in Proceedings of the European Conference on Computer Systems (EuroSys), Association for Computing Machinery, Inc., Lisbon, Portugal, Mar. 2007

[3] Ted Wobber, Aydan Yumerefendi, Martín Abadi, Andrew Birrell, Daniel R. Simon, Authorizing Applications in Singularity, in Proceedings of the 2007 Eurosys Conference, Association for Computing Machinery, Inc., Lisbon, Portugal, Mar. 2007

[4] Aiken, Mark, Fähndrich, Manuel, Hawblitzel, Chris, Hunt, Galen, Larus, James R., Deconstructing Process Isolation, in ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, pp. 1-10, ACM, San Jose, CA, Oct. 2006

[5] Paul Barham, Rebecca Isaacs, Richard Mortier, Tim Harris, Learning communication patterns in Singularity, in Proceedings of the First Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML), Jun. 2006

[6] Galen C. Hunt, Mark Aiken, Paul Barham, Manuel Fahndrich, Chris Hawblitzel, Orion Hodson, James R. Larus, Steven Levi, Nick Murphy, Bjarne Steensgaard, David Tarditi, Ted Wobber, Brian D. Zill, Sealing OS Processes to Improve Dependability and Security, no. MSR-TR-2006-51, pp. 14, Microsoft Research, Apr. 2006

[7] Galen Hunt, James R. Larus, Martin Abadi, Mark Aiken, Paul Barham, Manuel Fahndrich, Chris Hawblitzel, Orion Hodson, Steven Levi, Nick Murphy, Bjarne Steensgaard, David Tarditi, Ted Wobber, Brian D. Zill, An Overview of the Singularity Project, no. MSR-TR-2005-135, pp. 44, Microsoft Research, Oct. 2005

[8] Larus, James, Hunt, Galen, Tarditi, David, {End Bracket} Singularity, MSDN Magazine, vol. 21, no. 7, pp. 176, Jun. 2006

[9] Galen C. Hunt, James R. Larus, Singularity Design Motivation, no. MSR-TR-2004-105, pp. 4, Microsoft Research, Nov. 2004

[10] Martín Abadi , Andrew Birrell , Ted Wobber, Access control in a world of software diversity, Proceedings of the 10th conference on Hot Topics in Operating Systems, p.22-22, June 12-15, 2005, Santa Fe, NM

[11] R. S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman (1996), "Role-Based Access Control Models", IEEE Computer 29(2): 38-47, IEEE Press, 1996.

[12] D.F. Ferraiolo and D.R. Kuhn (1992) "Role Based Access Control" 15th National Computer Security Conference, Oct 13-16, 1992, pp. 554-563.

[13] Martín Abadi , Michael Burrows , Butler Lampson , Gordon Plotkin, A calculus for access control in distributed systems, ACM Transactions on Programming Languages and Systems (TOPLAS), v.15 n.4, p.706-734, Sept. 1993

[14] Elaine Shi , Adrian Perrig , Leendert Van Doorn, BIND: A Fine-Grained Attestation Service for Secure Distributed Systems, Proceedings of the 2005 IEEE Symposium on Security and Privacy, p.154-168, May 08-11, 2005

[15] Galen Hunt , Mark Aiken , Manuel Fähndrich , Chris Hawblitzel , Orion Hodson , James Larus , Steven Levi , Bjarne Steensgaard , David Tarditi , Ted Wobber, Sealing OS processes to improve dependability and safety, ACM SIGOPS Operating Systems Review, v.41 n.3, June 2007

[16] Aiken, Mark, Fähndrich, Manuel, Hawblitzel, Chris, Hunt, Galen, Larus, James R., Deconstructing Process Isolation, in ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, pp. 1-10, ACM, San Jose, CA, Oct. 2006

Rami Matarneh received the B.E. from Mu'tah Univ. in 1994, and M.E. degrees, from Kharkiv National University of Radio Electronics in 1997. He received the Dr. Eng. degree from Kharkiv National University of Radio Electronics in 2000. After working as an assistant professor (from 2000) in the Dept. of computer science, Philadelphia Univ. (from 2000), and an assistant professor, Al-Isra private university(from 2006). His research interest includes AI, automation design systems and security.

Suha Al_Jubori received the B.E., M. E., and Dr. Eng. degrees from Al-Nahrain Univ. in 1992, 1996, and 2006, respectively. After working an assistant professor (from 2006) in the Dept. of Management information systems, Al-Isra private university. Here research interest includes database management systems, security, and programming languages.