LICO: A LIGHTWEIGHT ACCESS CONTROL MODEL FOR INTER-NETWORKING LINKAGES

Seminar Report

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

In

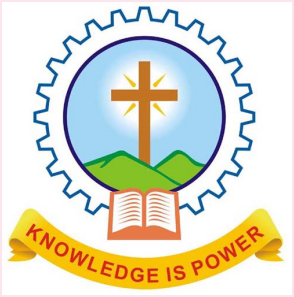
COMPUTER SCIENCE AND ENGINEERING

of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Submitted By

MIDHUN P



Department of Computer Science & Engineering
Mar Athanasius College Of Engineering Kothamangalam

LICO: A LIGHTWEIGHT ACCESS CONTROL MODEL FOR INTER-NETWORKING LINKAGES

Seminar Report

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Submitted By

MIDHUN P

Department of Computer Science & Engineering
Mar Athanasius College Of Engineering Kothamangalam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING
KOTHAMANGALAM

CERTIFICATE

This is to certify that the report entitled **Lico: A Lightweight Access Control Model for Inter-Networking Linkages** *submitted by* **Mr. MIDHUN P**, *Reg.No.***MAC15CS038** *towards partial fulfillment of the requirement for the award of Degree of Bachelor of Technology in Computer science and Engineering from APJ Abdul Kalam Technological University for December 2018 is a bonafide record of the seminar carried out by him under our supervision and guidance.*

.....	.....	.....
Prof. Joby George	**Prof. Neethu Subash**	**Dr. Surekha Mariam Varghese**
Faculty Guide	***Faculty Guide***	***Head of the Department***

Date: Dept. Seal

ACKNOWLEDGEMENT

First and foremost, I sincerely thank the God Almighty for his grace for the successful and timely completion of the seminar.

I express my sincere gratitude and thanks to Dr. Solly George, Principal and Dr. Surekha Mariam Varghese, Head Of the Department for providing the necessary facilities and their encouragement and support.

I owe special thanks to the staff-in-charge Prof. Joby george, Prof. Neethu Subash and Prof. Joby Anu Mathew for their corrections, suggestions and sincere efforts to co-ordinate the seminar under a tight schedule.

I express my sincere thanks to staff members in the Department of Computer Science and Engineering who have taken sincere efforts in helping me to conduct this seminar.

Finally, I would like to acknowledge the heartfelt efforts, comments, criticisms, co-operation and tremendous support given to me by my dear friends during the preparation of the seminar and also during the presentation without whose support this work would have been all the more difficult to accomplish.

ABSTRACT

Processes in operating systems are assigned different privileges to access different resources. A process may invoke other processes whose privileges are different; thus, its privileges are expanded (or escalated) due to such improper inheritance. Inter-networking can also occur between processes, either transitively or iteratively. This complicates the monitoring of inappropriate privilege assignment/escalation, which can result in information leakage. Such information leakage occurs due to privilege transitivity and inheritance and can be defined as a general access control problem for inter-networking linkages. Inter-networking is generally less studied in existing access control models. The proposed solution is a lightweight directed graph-based model, LiCo, which is designed to facilitate the authorization of privileges among inter-networking processes. Lico is the first general access control model for inter-invoking processes and general inter-networking linkages.

Contents

List of Figures

LIST OF ABBREVIATION

ACG	Access Control Graph
OSN	Online Social Networks
DAC	Direct Access Control
ABE	Attribute-Based Encryption
HABE	Hierarchical Attribute-Based Encryption
CP-ABE	Cipher text Policy Attribute-Based Encryption
KP-ABE	Key Policy Attribute-Based Encryption

Introduction

Access control is crucial for modern day systems, as it prevents unauthorized access, for example to resources in operating systems and application layers. Conventional access control models focus on direct access control, in which an accessor can be granted access to certain objects (resources), based on the corresponding privileges. For example, access control is defined directly by a tuple <a, o, p>, where a is an accessor, o is an object, and p is a privilege. An accessor can also be assigned to a role (i.e. role-based access control), and in such model, the privileges are bind to the roles (e.g., faculty members, department heads, and deans). These schemes have been widely studied in the literature.

In indirect context, however, one accessor (e.g., A) may invoke another accessor (e.g., B). This can potentially result in an unauthorized extension of As privileges. For example, As privilege for o is p1, but A can invoke B to gain additional privilege for o to p2. Thus, we need a model to formalize how and when we can permit such additional privilege extension. In social networks (e.g., Tencent QQ space), for example, A shares a photo with her friends (e.g., B and C). The friends may leave some comments (information) relating to the shared photo. One of the friends (e.g., B) may access As QQ space to gain other friends information (e.g., Cs information). How to regulate privilege abuse due to inter-invoking linkages among accessors is a topic that is under-explored in the existing literature.

Online social networks (OSNs) such as Facebook, Google, and Twitter are inherently designed to enable people to share personal and public information and make social connections with friends, coworkers, colleagues, family, and even with strangers. In recent years, we have seen unprecedented growth in the application of OSNs. For example, Facebook, one of representative social network sites, claims that it has more than 800 million active users and over 30 billion pieces of content (web links, news stories, blog posts, notes, photo albums, and so on.) shared each month. To protect user data, access control has become a central feature of OSNs.

A typical OSN provides each user with a virtual space containing profile information, a

list of the users friends, and webpages, such as wall in Facebook, where users and friends can post content and leave messages. A user profile usually includes information with respect to the users birthday, gender, interests, education, and work history, and contact information. In addition, users can not only upload a content into their own or others spaces but also tag other users who appear in the content. Each tag is an explicit reference that links to a users space. For the protection of user data, current OSNs indirectly require users to be system and policy administrators for regulating their data, where users can restrict data sharing to a specific set of trusted users. OSNs often use user relationship and group membership to distinguish between trusted and untrusted users. For example, in Facebook, users can allow friends, friends of friends (FOF), groups, or public to access their data, depending on their personal authorization and privacy requirements.

Although OSNs currently provide simple access control mechanisms allowing users to govern access to information contained in their own spaces, users, unfortunately, have no control over data residing outside their spaces.

For instance, if a user posts a comment in a friends space, she/he cannot specify which users can view the comment. In another case, when a user uploads a photo and tags friends who appear in the photo, the tagged friends cannot restrict who can see this photo, even though the tagged friends may have different privacy concerns about the photo.

To address such a critical issue, preliminary protection mechanisms have been offered by existing OSNs. For example, Facebook allows tagged users to remove the tags linked to their profiles or report violations asking Facebook managers to remove the contents that they do not want to share with the public. However, these simple protection mechanisms suffer from several limitations. On one hand, removing a tag from a photo can only prevent other members from seeing a users profile by means of the association link, but the users image is still contained in the photo. Since original access control policies cannot be changed, the users image continues to be revealed to all authorized users. On the other hand, reporting to OSNs only allows us to either keep or delete the content. Such a binary decision from OSN managers is either too loose or too restrictive, relying on the OSNs administration and requiring several people to report their request on the same content. Hence, it is essential to develop an effective and flexible

access control mechanism for OSNs, accommodating the special authorization requirements coming from multiple associated users for managing the shared data collaboratively.

It can be challenging to define an access control model for inter-networking linkages, as linkages can be either transitively or iteratively. There may also exist an invoking loop, and the inter-networking can be arbitrary linkage structures. There is also a need to design access control model that can be generalizable to different scenarios, such as social networks. This is the gap we seek to address in this paper. Specifically, we design a lightweight access control model (hereafter referred to as LiCo) to facilitate the authorization of fine-grained privileges among inter-invoking processes. The model is based on directed graph and comprises several key algorithms. LiCo is designed to efficiently detect privilege collisions, raise an alert, and properly authorize the right privileges.

Existing methods

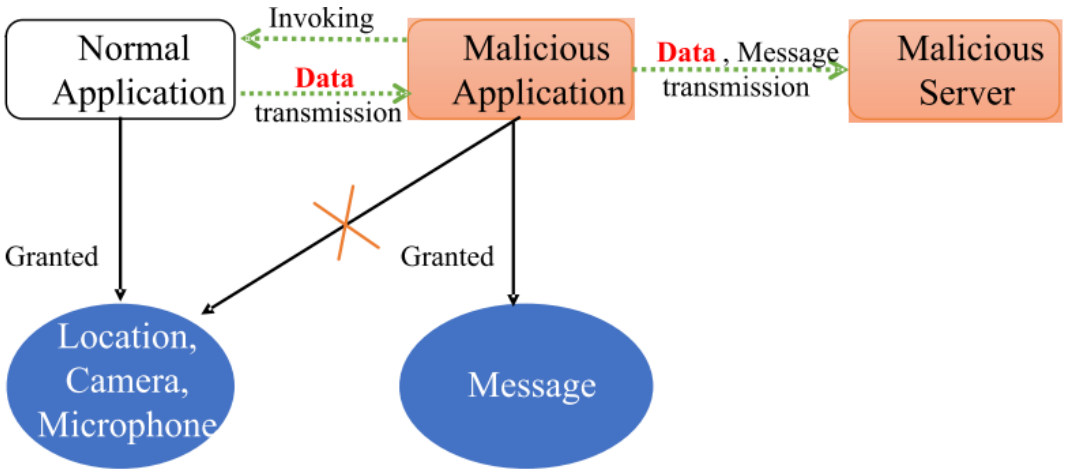


Fig. 2.1: An example on how a potentially malicious application can stealthily gain access to information via inter-network invoking

In time-sharing system of 1970s, Jampson proposed a DAC model for the data protection of multiuser computer system, mainly to set up the relationship table between the host and the guest(s).This model defines the level of security attributes on subject and object, to determine whether the subject has access to the object.

In August 2001, NIST published the first RBAC standard, which includes two parts: RBAC reference model and RBAC functional specification. It realized the logic separation between users and permissions, simplifying authorization processes.

There area number of other models, such as the 1997 TBAC model of Thomas and Sandhu, the domain based access control model DBAC of Shaifq et al. and the uniform access control framework of Covington et al. Petracca et al. presented an access control model to handle privacy-sensitive permission on mobile devices. In the approach, the operation requests of applications were verified to determine if these requests are within the users expectations explicitly.

In addiiton to RBAC access control models, there have been other access control models

such as those based the access decisions on contexts. For example, Bijon et al. studied the differences between conventional constraint-based access control and risk-aware approaches in RBAC, from which a framework are built for risk-awareness in RBAC models incorporating quantified-risks.

Access control has applications in a broad range of settings, such as online social networks (OSNs) where user-specific information are being shared. Carminati et al. presented an access control mechanism for OSNs, based on semantic web. Specifically, the authors encoded social network information (e.g., users profiles, relationships among users) using an ontology, based on which the access control model can be achieved and such mechanism can then be adapted for other OSN platforms by modifying the ontology. Ren et al. also designed and implemented a lightweight tree-based model called SeGoAC, which supports self-defined privilege grant and revocation, as well as proxy-enabled and group-oriented access control for file storage in mobile cloud computing.
Some of the methods for ensuring access control systems are

2.1 Attribute-based encryption

An attribute based encryption scheme was introduced by Sahai and Waters in 2005 and the goal is to provide security and access control Sahai et al 2007. Attribute-based encryption (ABE) is a public-key algorithm based one to many encryptions that allows users to encrypt and decrypt data based on user attributes. In their context, the role of the parties is taken by the attributes. Thus, the access structure will contain the authorized sets of attributes. They restrict the attention to monotone access structures. However, it is also possible to realize general access structures using the techniques by having the attribute as a separate attribute altogether. Thus, the number of attributes in the system will be doubled. From now on, unless stated otherwise, by an access structure we mean a monotone access structure.

2.2 Cipher text policy attribute-based encryption

Another modified form of ABE called CP-ABE was introduced by Sahai. In a CP-ABE scheme, every ciphertext is associated with an access policy on attributes, and every users private key is associated with a set of attributes. A user is able to decrypt a ciphertext only if the set of attributes associated with the users private key satisfies the access policy associated with the ciphertext. CP-ABE works in the reverse way of Key Policy Attribute-Based Encryption (KP-ABE). The access structure built in the encrypted data can let the encrypted data choose which key can recover the data; it means the user's key with attributes just satisfies the access structure of the encrypted data. The concept of this scheme is similar to the traditional access control schemes. The encryptor specifies the threshold access structure for his/her interested attributes while encrypting a message. Based on this access structure, the message is then encrypted such that only those whose attributes satisfy the access structure can decrypt it.

2.3 Key policy attribute-based encryption

KP-ABE is the modified form of classical model of ABE. Users are assigned with an access tree structure over the data attributes. Threshold gates are the nodes of the access tree. The attributes are associated by leaf nodes. To reflect the access tree structure, the secret key of the user is defined, Changji Wang et al 2013. Ciphertexts are labeled with sets of attributes and private keys are associated with monotonic access structures that control which ciphertexts a user is able to decrypt. Key Policy Attribute Based Encryption (KP-ABE) scheme is designed for one-to-many communications.

2.4 Hierarchical attribute-based encryption

The Hierarchical Attribute-Based Encryption (HABE) is derived by Wang et al The HABE model consists of a Root Master (RM) that corresponds to the Third Trusted Party (TTP),Multiple Domain Masters (DMs) in which the top-level DMs correspond to multiple enterprise users, and numerous users that correspond to all personnel in an enterprise. This scheme used the property of hierarchical generation of keys in HIBE scheme to generate keys. This scheme can satisfy the property of fine grained access control, scalability and full delegation. It can share data for users in the cloud in an enterprise environment. Furthermore, it can apply to achieve proxy re-encryption. But in practice, it is unsuitable to implement. Since all attributes in one conjunctive clause in this scheme may be administered by the same domain authority, the same attribute may be administered by multiple domain authorities.

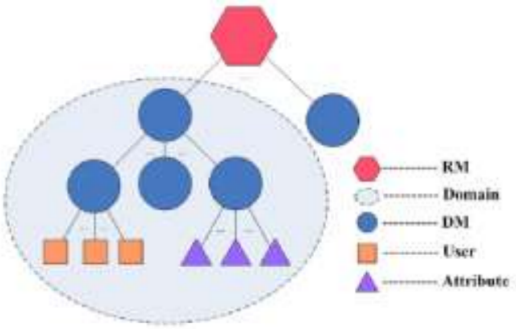
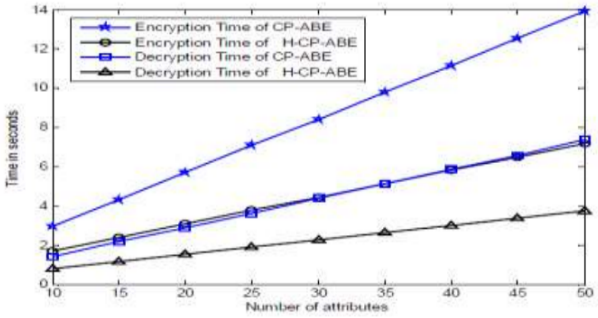


Fig. 2.2: A three-level HABE Model

In HABE model, the RMs role closely follows the root private key generator (PKG) in a HIBE system, is responsible for the generation and distribution of system parameters and domain keys. The DM, whose role integrates both the properties of the domain PKG in a HIBE system and AA in a CP-ABE system.



It is responsible for delegating keys to DMs at the next level and distributing keys to users. Specifically, we enable the leftmost DM at the second level to administer all the users in a domain, just as the personnel office administers all personnel in an enterprise, and not to administer any attribute. Also other DMs administer an arbitrary number of disjoint attributes, and have full control over the structure and semantics of their attributes. In the HABE model, we first mark each DM and attribute with a unique identifier (ID), but mark each user with both an ID and a set of descriptive attributes. Then,as Gentry et al 2002, we enable an entitys secret key to be extracted from the DM administering itself, and an entitys public key, which denotes its position in the HABE model, to be an ID tuple consisting of the public key of the DM administering itself and its ID.

Proposed method

Security breach is one of several consequences due to indirectly invoking of processes. Although potentially malicious application cannot directly access certain resources such as location, camera, and microphone, such application can indirectly gain access to these resources by invoking another application that has been granted permission to these resources.

Note that the above threat model can be generalized. For example, some information accessed in one application (e.g., app A) may expand the accessible objects of another application (e.g., app B) who invokes this application (i.e., app A). This situation can occur not only during process invoking, but also in OSNs. For example, in Tecent QQ, if one (e.g., user A) makes his/her QQ space public to say users B and C, then users B and C may obtain additional information of another user, say user D by accessing user As QQ space. Such leakage is always ignored by users and service providers, as it is challenging to objectively define the type of information that is being leaked.

Despite the challenge in not being able to objectively define leakage in such a situation, we argue that the leakage risk can be modeled as a privilege collision problem. Once the privileges of a subject (SA) for an object collides with the privileges of another subject (SB) with links to SA, we should regulate the privilege that are invoked (i.e., SA) S. Li et al.: Lico: Lightweight Access Control Model for Inter-Networking Linkages or invokes (i.e., SB). Thus, to enhance ones understanding, we propose a graph model for access control that can visualize the inter-relation among accessors (e.g., processes) and the objects (resources, such as location, camera, and microphone) as follows:

Definition 1: Access Control Graph (ACG). It is a graph taht consists of vertexes and directed edges. There are two types of vertexes, namely: accessors (presented by black points), and objects (presented by white triangles). There are also two types of directed edges, namely: directed edges from an accessor to another, representing invoking relations, and directed edges from an accessor to an object, which represent accessing relations.

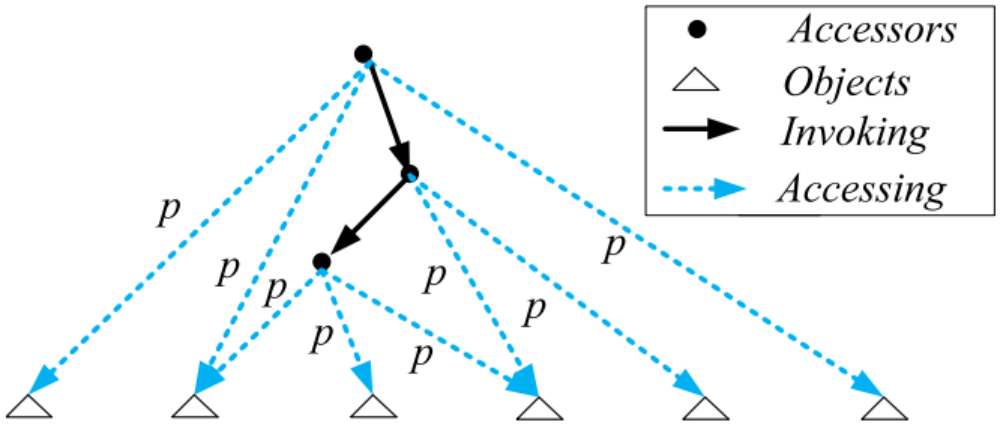


Fig. 3.1: An example of ACG

We can state ACG formally as follows:

$$
\begin{aligned}
ACG &::= < V, E >; \\
V &= V_{Acc} \cup V_{Obj}; \\
E &= E_{Inv} \cup E_{Acc}; \\
E_{Inv} &::= \langle V_{from}, V_{to} \rangle, \quad V_{from} \in V_{Acc},\ V_{to} \in V_{Acc}; \\
E_{Acc} &::= \langle V_{from}, V_{to} \rangle, \quad V_{from} \in V_{Acc},\ V_{to} \in V_{Obj}.
\end{aligned}
$$

Where V is a set of vertexes containing Accessors and Resources, E is a set of edges containing Invoking edges and Accessing edges. Invoking edges are the edges whose both source vertex and destination vertex are Acessors, Accessing edges are the edges whose source vertex is an Accessor and destination vertex is a Resource.

The design of ACG is motivated by differentiating accessors and relations. An abstract model presented by a graph has the following advantages: it is general for diverse applications, it is easy to describe, and it is easy to understand. Moreover, some concepts in graph theory can be used to convey ideas, such as transitive closure, cyclic graph, and clusters. Graph algorithms are also be available as a basic tool.

3.1 Access control model

A simple ACG comprises accessors (processes) that can access objects (resources) by both direct and indirect means. There are, however, situations where a processor may require indirect information from another process (e.g., a shopping application may request for location data) and the inter-networking process is one attack vector that may be exploited .In our approach, the goal is to detect privilege collisions due to such indirect paths that may lead to the abuse of privileges (e.g., gain extra access privileges without permission), and to mitigate such collisions.

Access Control Graph is a declarative way to define access rights, task assignments, recipients and content in information systems. The access rights are granted to objects like files or documents, but also business objects like an account. It can also be used for the assignment of agents to tasks in workflow environments. Organizations are modeled as a specific kind of semantic graph comprising the organizational units, the roles and functions as well as the human and automatic agents (i.a. persons, machines). Compared to other approaches like role-based access control or attribute-based access control, the main difference is that in ACG access rights are defined using an organizational query language instead of total enumeration. Our Access control graph consisting of two types of vertexes and two types of edges. Vertexes consisting Accessors and Resources and Edges consisting of Invoking edges and Accessing edges. Specifically, ACG can be defined as follows:

1) $ACG ::= \langle V, E \rangle$, where V is a set of vertexes and E is a set of edges.
2) $V = V_{Acc} \cup V_{Obj}$, where V_{Acc} is a set of accessors and V_{Obj} is a set of objects.
3) $E = E_{Inv} \cup E_{Acc}$, where E_{Inv} is a set of invoking edges and E_{Acc} is a set of accessing edges.
4) $E_{Inv} ::= \langle V_{from}, V_{to} \rangle$, $V_{from} \in V_{Acc}$, $V_{to} \in V_{Acc}$.
5) $E_{Acc} ::= \langle V_{from}, V_{to}, P \rangle$, $V_{from} \in V_{Acc}$, $V_{to} \in V_{Obj}$, $P \in Pri$.
6) $\forall e \in E_{Acc}$, $e = \langle from, to, p \rangle$ where $from \in V_{Acc}$, $to \in V_{Obj}$, $p = [p_1, p_2, \ldots, p_n] \in \{0, 1\}^n$, $n = |Pri|$.

7) $\forall e \in E_{Inv}$, $e = \langle from, to \rangle$ where $from \in V_{Acc}$, $to \in V_{Acc}$.
8) Pri is a set of privileges. $Pri ::= \langle pri_1, pri_2, \ldots, pri_n \rangle$, where $|Pri| = n$. E.g., $Pri ::= \langle read, write, update \rangle$.
9) $\forall e \in E_{Acc}$, if $p_i = 1$, then accessor $e.from$ can access object $e.to$ with privilege $pri_i \in Pri$. Otherwise, $p_i = 0$ means $e.from$ access $e.to$ without pri_i. For example, $p = 110$ means *read* and *write*; $p = 101$ means *read* and *update*.

Given an accessor, all accessors that can be invoked, directly or indirectly by this accessor, are of interest and they can be modeled as a closure. A closure process (denoted as Clsa) that returns all invoked accessors for a given accessor is defined as follows:

Definition 2: Closure process Clsa : a belongs to VAcc gives A which is a subset of VAcc. Clsa takes as input an accessor a belongs to VAcc and outputs a set of invoked accessors A belongs to VAcc. Initially, A = {a}. Next, let A gives {v for all a belongs to A, e belongs to EInv, a = e.from, v = e.to} Union A. Note that the above process is recursive.

Given an accessor, all objects that can be accessed directly or indirectly can be returned by the following function, denoted as Clsao. It relies on Clsa() as a subfunction.

Definition 4: Closure function Clsao : a belongs to VAcc gives O which is a subset of VObj. Clsao takes as input an accessor a belongs to VAcc and outputs a set of accessed objects O which is a subset of VObj. That is, O = {v, for all a' belongs to Clsa(a), e belongs to EAcc, a' = e.from, v = e.to}

The following function, denoted as HD, returns the hamming distance that can reveal whether collision occurs between two privileges.

Definition 5: Hamming distance function HD : p1 belongs to {0, 1} n x p2 belongs to 0, 1n gives n belongs to Z. HD takes as input p1, p2 belongs to {0, 1}n , and outputs n which is a Hamming Distance of p1 and p2.

The following function (denoted as Clla) returns collisions between privileges for the same objects - privileges granted initially and privileges gained by transitively invoking.

Definition 6: Privilege collisions for closure function Clla : a belongs to VAcc gives n Z. Clla takes as input an accessor a belongs to VAcc and outputs an integer number n = max({HD(p1,

p2)|e1 belongs to E1, e2 belongs to E2, p1 = e1.p, p2 = e2.p, e1.to = e2.to}), where E1 = {e1|e1 belongs to EAcc,e1.from = a, e1.to belongs to Clso(a), E2 = e2|e2.from belongs to Clsa(a), e2.to belongs to Clsao(a)})

We will now present the theorem relating to the detection of privilege breaches.

Theorem 1: for all a belongs to VAcc, if Clla(a belongs to VAcc) not equal to 0 or interscetion between Clsao(a) and Clso(a) is not null, then this results in privacy breach due to inter-networking linkages.

Proof 1: For any accessor a in accessors, the objects that a can access are Clso(a). The accessors that are invoked by a are Clsa(a). Any privileges for a originally are e1.p, where e1.from = a, e1.to belongs to Clso(a). Any privileges for accessors invoked by a are e2.p, where e2.from belongs to Clsa(a) and e2.to belongs to Clsao(a). Their differences are HD (e1.p, e2.p), where the maximal is Clla(a). If Cll(a) not equal to 0, then there is a privacy breach due to the invoking.Alternatively when intersection between Clsao(a) and Clso(a) is not null, there exists an object that cannot be accessed by a originally but can be accessed by an accessor invoked by a.

Privilege collision is related to the definition of individual privilege and the relation among them. For example, if each privilege is independent, then collision will occur once there exist differences. If privilege is dependent (e.g., subset relation), then the collision will be more complicated. We will now define a function (Pric) on privilege collision, based on hamming distance.

Definition 7: Privilege collision function Pric : p1 belongs to {0, 1} n x p2 belongs to 0, 1 n gives b belongs to {0, 1} takes as input p1, p2 belongs to {0, 1} n and outputs b = 0 if HD (p1, p2) is not equal to 0 (which means collision occurs). Otherwise, the output b = 1.

Privileges may, however, not collide with each other. For example, write does not collide with update. That is, privileges between each other are not entirely exclusive. In this situation, we need to extend Pri'c(,) to a more general version, Pri'c (,), which specifies whether collision occurs for the given pairwise privileges. It is described as follows:

Definition 8: Privilege collision function Pri'c : p1 belongs to Pri x p2 belongs to Pri gives b belongs to {0, 1} takes as input p1, p2 belongs to Pri and outputs b = 0 if p1 collides with p2 (depending on concrete application logics). Otherwise, b = 1.

We can map a privilege array consisting of 0, 1 (e.g., [1, 0, . . . , 1, 0]) to represent individual concrete privileges. We, thus, define a new process (denoted as A2PSet) to compute this mapping.

Definition 9: Array maps to privilege process A2PSet : p belongs to {0, 1} n gives pri1, pri2, . . . , prin belongs to Pri, n = |Pri |. A2P takes as input p belongs to {0, 1} n an outputs a set of privileges, in which Prii is included if the i-th bit in array p is 1. Otherwise, Prii is excluded from the set.

By using the above mapping process (i.e., A2PSet()), we can define a generalized collision detection function (denoted as A2C) that can handle different relation types between underlying individual privileges. It is described as follows:

Definition 10: Privilege collision by array function A2C : p1 belongs to {0, 1} n x p2 belongs to {0, 1} n gives b belongs to {0, 1} takes as input p1, p2 belongs to 0, 1 n and outputs a b = 0 if for all pa belongs to A2PSet(p1) there exists pb belongs to A2PSet(p2) such that Pri'c (pa, pb) = 0. If b = 1, then no privilege collision occurs.

By using the above generalized version of collision detection function (i.e., A2C), we can define the following function called Clla'.

Definition 11: Privilege difference for closure function Cll'a : a belongs to VAcc gives b belongs to {0, 1} takes as input an accessor a belongs to VAcc and outputs.

$$b = \prod_{\forall e_1 \in E_1, \forall e_2 \in E_2, e_1.to = e_2.to} A2C(e_1.p, e_2.p),$$

Where E1 = {e1|e1 belongs to EAcc, e1.from = a, e1.to belongs to Clso(a)}, and E2 = {e2|e2.from belongs to Clsa(a), e2.to belongs to Clsao(a)})}. If b = 0, then collision occurs; otherwise, there is no privilege collision.

We will now present a general conclusion. Corollary 1: for all a belongs to VAcc, if Cll'a (a belongs to VAcc) = 0 or intersection between Clsao(a) and Clso(a) is not null, then privacy breaches due to inter-networking linkages.

Proof 2: Straightforward due to Theorem 1. Usually, for all pri1, pri2 belongs to Pri, pri1 $<$ pri2, where $<$ means implied by. For example, read is implied by write, that is, read ¡ write.

Suppose Pri = [pri1, . . . , prin], and pri1 $<$ pri2 $<$. . . $<$ prin. Thus, we can map privileges into ordered integers, such as 1, 2, . . . , n. Clearly, only the largest one is sufficient to denote privileges. Thus, for all e belongs to EAcc, e = ¡from, to, p¿ where from belongs to VAcc, to belongs to VObj, p belongs to Z. For this situation, we introduce Cll"a(.) as follows:
Definition 12: Privilege difference for closure function Cll"a : a belongs to VAcc gives b belongs to $\{0, 1\}$ takes as input an accessor a belongs to VAcc and outputs.

$$b = \prod_{\forall e_1 \in E_1, \forall e_2 \in E_2, e_1.to = e_2.to} IsLEQ(e_2.p, e_1.p),$$

Where E1= {e1 |e1 belongs to EAcc , e1.from = a, e1.to belongs to Clso(a)}, E2 = {e2 |e2.fro belongs to Clsa(a), e2.to belongs to Clsao (a))}, and IsLEQ(c belongs to N, d belongs to N) returns 1 if c d and returns 0 otherwise. If b = 0, then collision occurs. Otherwise, no privilege collision occurs.

We can simplify the conclusion to the following:
Corollary 2: for all a belongs to VAcc , if Cll"a (a belongs to VAcc) = 0 or intersection between Clsao(a) and Clso(a) is not null, then privacy breaches due to linkage.
Proof 3: Straightforward due to Theorem 1.

3.2 Proposed authorizing rules

Next, we will propose relevant access control rules, based on the basic model described in the preceding section.
Proposition 1: Regulating Privileges of Invoked Processes from an Accessor (Rule I: Privilege is non-increasing for an invoking edge) Suppose e belongs to EInv and invoking from accessor e.from. If e'1.to = e'2.to, where e'1,e'2 belongs to EAcc, e.from = e'1.from,e.to = e'2.from, then let e'2.p = e'1.p. Otherwise,let e'2.to = false.

In other words, the above rule states that if an accessor (e.g., process A) invokes another accessor (e.g., process B), then the privilege of invoked accessor (i.e., process B) must be less than or equal to the privilege of invoking accessor (i.e., process A) once the same object is

accessed. The access control unit will normalize the privilege of invoked process B when B is invoked by A.

Definition 13: Invoking Path. If there exists e1,e2,...,en belongs to EInv, n greater than or equal to 2 such that e1.to = e2.from, e2.to = e3.from, ...,ei.to = ei+1.from,..., en1.to = en.from, then we denote this invoking path as Path Inv = [e1,e2,...en].

Proposition 2: Regulating Privileges of Invoked Processes from an Accessor (Rule II: Privilege is non-increasing across an invoking path). Suppose Path Inv = [e1,e2,...,en], n greater than or equal to 2 exists and invoking is from accessor e1.from. If there exists i,j belongs to {1,2,...,n}, i less than j such that e'i.to = e'j.to, where e'i,e'j belongs t E Acc, ei.from = e'i.from, ej.from = e'j.from, then let e'j.p= e'i.p. If e'i.to not equal to e'j.to, then let e'j.to = false.

That is, the above rule states that if an accessor (e.g., process A) invokes multiple accessors sequentially (e.g., processes B1,B2,..., Bn), then the privilege of invoked accessor (i.e., Bi, i=2,...,n) must be less than or equal to the privilege of invoking accessor (i.e., Bi1) once the same object is accessed. The access control unit will normalize the privilege of invoked process Bi when Bi is invoked by Bi1 (for i =2,3,...,n). For i=1, B1 is invoked by A, which is degenerated to Rule I.

Definition 14: Invoking Loop. If there exists e1,e2,...,en belongs to EInv, n greater than or equal to 2 such that e1.to = e2.from, e2.to = e3.from, ..., ei.to = ei+1.from,..., en1.to = en.from, en.to = e1.from, then denote this invoking loop as Loop Inv = [e1,e2,...,en].

Proposition 3: Regulating Privileges of Invoked Processes from an Accessor (Rule III: Privilege is minimized in an invoking loop). Suppose Loop Inv = [e1,e2,...,en], n greater than or equal to 2 exists and invoking is from any accessor in a |ei.from = a, i = 1,2,...,n. If there exists i,j belongs to {1,2,...,n}, i less than j such that e'.to = e'j.to and e'i.p greater than e'j.p where e'i, e'j belongs to EAcc, ei.from = e'i.from, ej.from = e'j.from, then let e'i.p = e'j.p. If e'i.to = e'j.to, then e'.j.to = false.

In other words, the above rule states that if an accessor (e.g., process A) invokes multiple accessors sequentially (e.g., processes B1,B2,...,Bn), and forms a loop (i.e.,Bn invokes A, then the privilege of invoked accessor (i.e.,Bi, i = 1 , 2 ,... , n) must be less than or equal to the privilege of invoking accessor (i.e.,A) once the same object is accessed by all.

The access control unit will normalize the privileges of all invoked processes Bi, i = 1,2,...,n to the least among them.

Proposition 4: Regulating Privileges of Invoking Processes from an Accessor (Rule IV: Privilege must be non-decreasing for invoking processes). Suppose There exists e belongs to EInv. If e.to is invoked by e.from, and e1.to = e2.to, where e1,e2 belongs to EAcc, e.from = e1.from, e.to = e2.from, then let e1.p = e2.p. If e1.to not equal to e2.to, then e1.to is false.

In other words, the above rule states that if an accessor (e.g., process A) is invoked by an accessor (e.g., process B), then the privilege of invoking accessor (i.e., process B) must be larger than or equal to the privilege of invoked accessor (i.e., process A). The access control unit will normalize the privilege of process B to be larger than or equal to that of A.

Proposition 5: Regulating Privileges of Inter-network Invoking Process Closure from an Accessor (Rule V: Privilege for process closure is non-increasing). Suppose there exists e belongs to EInv. If there exists v belongsto Clsa(e.from), e1.to = e2.to, where e1,e2 belongs to EAcc, e.from = e1.from, v = e2.from, then let e2.p = e1.p. If there exists v belongs to Clsa(e.from), e2.to not equal to e1.to, where e1,e2 belongs to EAcc, e.from = e1.from, v = e2.from, then e2.to is false.

The above rule states that if an accessor (e.g., process A) invokes another accessor in its invoking closure set (e.g., process B), then the privilege of invoked accessor (i.e., process B) must be less or equal to the privilege of invoking accessor (i.e., process A). Access control unit will normalize the privilege of process B to be less than or equal to that of process A.

Proposition 6: Regulating Privileges Container (Rule VI: Privilege container). Given any v belongs to VInv, if there exists e1 belongs to EInv, e2,e3 belongs to EAcc such that e1.from = f , e1.to = t, e2.from = f, e2.to = o, e3.from = t, e3.to = o, then let e3.p be e2.p or e3.p less than or equal to e2.p. The above rule is iterative for any inter-networking processes, which becomes a container for regulating the maximum privilege for any invoking processes.

These all authorizing rules are used to find the privacy breach in inter networking linkages and to make the authorizing algorithms. After all these proposed authorization rules unauthorized privacy breaches through inter networking linkages are found. Then we have two choices, either we can allow or deny the access.

3.3 Proposed algorithms

Now there is a couple of algorithms to achieve the above directed graph based access control model. Although our model can formally specify the rationale in access control mechanisms, these proposed algorithms can facilitate the understanding of programmers in their implementations.

Algorithm 1 returns all invoking accessors given any accessor in ACG. It can be considered the instantiation of Clsa(.). Using this function, all invoked accessors, directly or indirectly, can be returned and further examined.

In this algorithm first we will assign the set of vertex to A. Then we will select a single vertex a from A. While the set A becomes null we will find each vertex which can invoked directly or indirectly is identified. The result is added with A. The vertex a can call itself. So for the final result we will remove a from the result set.

Data: ACG, $v \in V_{Acc}$
Result: $A = Cls_a(v)$.

```
1 A ⇐ {v};
2 while A ≠ ∅ do
3 |   Select a ∈ A;
4 |   while ∃e ∈ E_Inv such that e.from = a do
5 |   |   A ⇐ A ∪ {e.to};
6 |   end
7 |   A ⇐ A − {a};
8 end
9 return A;
```

Algorithm 2 is a recursive algorithm that can obtain all accessible objects, directly and indirectly. It can be considered an instantiation of Clsao(.). In this algorithm, e.to denotes the objects that e.from can access with corresponding privileges e.pri.v.visit belongs to{0,1}, visit is a label to denote whether a vertex has been visited.

In this algorithm we will visit each vertex by assigning v'.visit = 1. Then we will find each accessors which can accessed directly by the vertex v'.

Data: ACG, $v \in V_{Acc}$
Result: $< o, pri >, o \in O \subset V_{Obj}, pri \in Pri$ where $o \in O, \exists e \in E_{Acc}, e.to = o, e.pri \in Pri, e.from \in Cls_a(v)$.

```
for each v' in Cls_a(v) do
    if v'.visit ≠ 1 then
        v'.visit ⇐ 1; /* denote the current vertex v' has been visited */
        /* add all of those objects and their corresponding privileges belonged to v' ∈ Cls_a(v) to a list */
        find e ∈ E_Acc where e.from = v';
        result ⇐ result ∪ {e.to, e.pri};
    end
end
return result;
```

Algorithm 3 can detect privilege collisions for a given access control graph. In other words, once inter-networking relations are given as well as the original accessible privileges, the privilege collisions due to inter-process invoking can be detected by this algorithm.

In this algorithm we will visit each vertex by assigning v'.visit = 1. Then we will find each accessors which can accessed directly and indirectly by the vertex v'. Then we will check whether these both accessors going to the same location. If yes then we will check that whether both accssors have same priority. If there is no equal priority that means there is a privacy breach due to inter networking linkage. Or else if the source of accessors are same but having different location then also this algorithm return Yes. Which will indicate the privacy breach due to inter network linkage .

```
   Data: ACG
   Result: Yes, No
1  for each v in V_Acc do
2      if v.visit ≠ 1 then
3          v.visit = 1; /* denote the current vertex v has
           been visited */
4          find e1, e2 ∈ E_Acc where
           e1.from = v, e2.from ∈ Cls_a(v);
5          if (e1.to = e2.to and
           e1.pri ≠ e2.pri) or (e1.to ≠ e2.to) then
6              return Yes;
7          end
8      end
9  end
10 return No;
```

Algorithm 4 can regulate privileges for a given access control graph. In other words, it can be implemented as an access control module to regulate concrete accessing policies and avoid privilege breaches.

In this algorithm we will visit each vertex by assigning v'.visit = 1. Then we will find each accessors which can accessed directly and indirectly by the vertex v'. Then we will take two accessors who have the same source. Then check if the destinations of the both are same. If both the destinations are same but both having different priority we will assign equal priority for the two accessors. Else if the source are same but the destinations are different we will make the destination of second accessor invalid.

```
   Data: ACG
   Result: ACG'
1  for each v in V_Acc do
2      if v.visit ≠ 1 then
3          v.visit = 1; /* denote the current vertex v has
           been visited */
4          find e1, e2 ∈ E_Acc where
           e1.from = v, e2.from ∈ Cls_a(v);
5          if e1.to = e2.to and e1.pri ≠ e2.pri then
6              e2.pri = e1.pri;
7          end
8          if e1.to ≠ e2.to then
9              e2.to = NULL;
10         end
11     end
12 end
13 return ACG';
```

3.4 Examples

Example I: Process A invokes another process (e.g., process B) in order to access object o. The control unit will detect privilege collisions and decide whether process A can expand its privileges to that of process B or process A has to limit the privileges to its own.

Example I can take place in operating system, web services, application programming interface, dynamic link library, developing frameworks, remote process calling, and so on.

Example II: Process A is invoked by another process (e.g., B) in order to access object o. Process A will consult the control unit to check the original privileges of process B for o and then process A is limited to these privileges.

The distinction between Examples I and II is in the control domain of the control unit, that is, the former is at invoking whilst the latter is at the invoked process.

Example III (A General Case): In OSN applications (e.g., Facebook or QQ), if one user (e.g., user A) shares some information such as a photo or video with others, these other users may comment on the shared material. User A can access such user-generated comments, say

of users B and C . However, in some context, we need to determine whether user B can access the comments from user C and vice versa, as the comments may reveal information about the comment originator.

3.5 Security and performance analysis

In this section, we will evaluate the security of Lico.

Proposition 7: Algorithm 1 returns all invoking processes that link to a given process.

Proof 4: The proof is straightforward. The algorithm returns the closure set of a given accessor node in ACG.

Proposition 8: All objects that can be accessed directly or indirectly, and their corresponding privileges can be computed by Algorithm 2.

Proof 5: The proof is straightforward. Algorithm 2 can return all directly and indirectly accessed objects and their corresponding privileges by iterative searching.

Proposition 9: The privilege collision can be detected by Algorithm 3 and avoided by Algorithm 4.

Proof 6: Algorithm 3 can return privilege collisions (or privilege breaches) by detecting the non-property of accessing objects or the distinction of accessing privileges for identical objects. Thus, privilege abuse can be detected and avoided.

The following proposition states the security of the proposed model.

Proposition 10: Lico is secure against privilege breaches from inter-process invoking among inter-networking processes.

Proof 7: The risks from process invoking comes from object or privilege transition among processes. Proposition 8 proves that the proposed model can search all accessible objects and corresponding privileges, which will be regulated automatically by the system, or will be alerted by the users (e.g., administrators) and they can take certain mitigation, such as explicitly rejecting the permission.

We will now evaluate the performance of the proposed model.

Proposition 11: The proposed model is lightweight.

Proof 8: The proof is straightforward. The model is formalized by ACG and instantiated by lightweight algorithms, whose performance is O(|VAcc |), where where VAcc is the number of accessor vertexes in ACG.

Conclusion

This journal focuses on privacy beaches in Inter-networking linkage. The proposed solution is a lightweight graph-based model for access control among inter-networking processes.This design is motivated by the observation that privilege misuses can occur due to inter-invoking among processes. The proposed model is designed to be generalizable and can be applied for access control in inter-networking linkages. This extends conventional access control models such as RBAC. The proposed graph-based model is also lightweight, and the cost is only O(n), where n is the number of accessor vertexes in the access control graph.

Research includes a more comprehensive evaluation of its security and performance in a real-world implementation.

REFERENCES

[1] H.-B. Shen and F. Hong, An attribute-based access control model for Web services, in Proc. IEEE 7th Int. Conf. Parallel Distrib. Comput., Appl. Technol. (PDCAT), Dec. 2006, pp. 7479.

[2] X. Jin, L. Wang, T. Luo, and W. Du, Fine-grained access control for HTML5-based mobile applications in Android, in Proc. 16th Int. Conf. Inf. Secur., Cham, Switzerland: Springer, 2015, pp. 309318.

[3] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger, Fuzzy multi-level security: An experiment on quantified risk-adaptive access control, in Proc. IEEE Symp. Secur. Privacy (SP), May 2007, pp. 222230.

[4] H. Hu, G. J. Ahn, and J. Jorgensen, Multiparty access control for online social networks: Model and mechanisms, IEEE Trans. Knowl. Data Eng., vol. 25, no. 7, pp. 16141627, Jul. 2013.

[5] D. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, Proposed NIST standard for role-based access control, ACM Trans. Inf. Syst. Secur., vol. 4, no. 3, pp. 224274, 2001.

[6] R. K. Thomas and R. S. Sandhu, Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management, in Proc. 11th IFIP WG11.3 Conf. Database Secur., Lake Tahoe, NV, USA, 1997, pp. 166181.

[7] B. Shafiq, J. B. D. Joshi, E. Bertino, and A. Ghafoor, Secure interoperation in a multidomain environment employing RBAC policies, IEEE Trans. Knowl. Data Eng., vol. 17, no. 11, pp. 15571577, Nov. 2005.

[8] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, Securing context-aware applications using environment roles, in Proc. 6th ACM Symp. Access Control Models Technol., vol. 5. Chantilly, VA, USA: ACM, 2001, pp. 1020.

[9] G. Petracca, A.-A. Reineh, Y. Sun, J. Grossklags, and T. Jaeger, AWare: Preventing abuse of privacy-sensitive sensors via operation bindings, in Proc. 26th USENIX Secur. Symp. Vancouver, BC, Canada: USENIX Association, 2017, pp. 379396.

[10] W. Ren, R. Liu, M. Lei, and K. K. R. Choo, SeGoAC: A tree-based model for self-defined, proxy-enabled and group-oriented access control in mobile cloud computing, Comput. Standards Interfaces, vol. 54, pp. 2935, Nov. 2017.