IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 16, 2018, accepted September 10, 2018, date of publication September 17, 2018, date of current version October 8, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2870148

Lico: A Lightweight Access Control Model for Inter-Networking Linkages

SIMIN LI[1], WEI REN[1,2,3], (Member, IEEE), TIANQING ZHU[4,5], (Member, IEEE), AND KIM-KWANG RAYMOND CHOO[1,6], (Senior Member, IEEE)

[1]School of Computer Science, China University of Geosciences, Wuhan 430074, China
[2]Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan 430074, China
[3]Guizhou Provincial Key Laboratory of Public Big Data, Guizhou University, Guizhou 550025, China
[4]School of Software, University of Technology Sydney, Ultimo, NSW 2007, Australia
[5]School of Mathematics and Computer Science, Wuhan Polytechnic University, Wuhan 430023, China
[6]Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249-0631, USA

Corresponding author: Wei Ren (weirencs@cug.edu.cn)

This work was supported in part by the Major Scientific and Technological Special Project of Guizhou Province under Grant 20183001, in part by the Open Funding of the Guizhou Provincial Key Laboratory of Public Big Data under Grant 2017BDKFJJ006, in part by the Open Funding of the Hubei Provincial Key Laboratory of Intelligent Geo-Information Processing under Grant KLIGIP2016A05, and in part by the National Natural Science Foundation of China under Grant 61502362.

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. This is also a topic that is generally less studied in existing access control models. Specifically, in this paper, we propose a lightweight directed graph-based model, LiCo, which is designed to facilitate the authorization of privileges among inter-networking processes. To the best of our knowledge, this is the first general access control model for inter-invoking processes and general inter-networking linkages.

INDEX TERMS Access control, privilege management, privilege transitivity, graph theory, process management.

I. 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 $\langle a, o, p\rangle$, 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) [2]. 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 A's privileges. For example, A's privilege for o is p_1, but A can invoke B to gain additional privilege for o to p_2. 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 A's QQ space to gain other friends' information (e.g., C's information). How to regulate privilege abuse due to inter-invoking linkages among accessors is a topic that is under-explored in the existing literature.

Inter-invoking occurs when interactions among processes are frequent, for example for collaboration (e.g., a shopping application needs to cooperate with a transaction application for product payment on mobile devices). While these different processes have different privileges (e.g., the transaction application can access the users' payment account and location), security vulnerabilities may arise during the inter-invoking processes (e.g., by invoking a process that can directly access location information, a process may indirectly

access the user's location information). In addition, current access control model for inter-invoking may regulate privileges by a switch - yes or no, which is not fine-grained.

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.

The rest of the paper is organized as follows. Sections II and III survey related work and describe the research problem, respectively. Section IV describe our proposed model, whose security and performance are then evaluated in Section V. Finally, we conclude the paper in Section VI.

II. RELATED WORK

In time-sharing system of 1970s, Jampson [3] 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 [4]. In August 2001, NIST [5] 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 are a number of other models, such as the 1997 TBAC model of Thomas and Sandhu [6], the domain based access control model DBAC of Shaifq *et al.* [7], and the uniform access control framework of Covington *et al.* [8]. Petracca *et al.* [9] 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 recent years, there have been attempts to use attributed-based encryption (ABE) in access control model. For example, Goyal *et al.* [10] developed a cryptosystem for fine-grained sharing of encrypted data, Key-Policy Attributed-Based Encryption (KP-ABE). Other research on ABE-based access control models include [11], [12] (e.g., for cloud computing), [13], [14] (e.g., web service), and so on. The popularity of HTML5-based mobile applications has also attracted the attention of access control researchers. In [15], a fine-grained access control mechanism for HTML5-based applications in Android system was proposed.

In addiiton to RBAC access control models, there have been other access control models such as those based the access decisions on contexts [16]. For example, Bijon *et al.* [17] 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 [18]. Carminati *et al.* [19] presented an access control mechanism for OSNs, based on semantic web. Specifically, the authors encoded social network information (e.g., user's 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.* [20] 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.

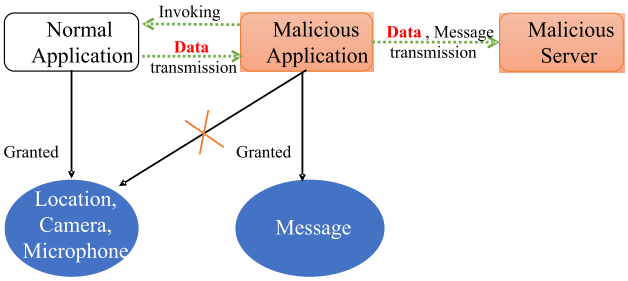

FIGURE 1. An example on how a potentially malicious application can stealthily gain access to information via inter-network invoking.

III. PROBLEM FORMULATION

Security breach is one of several consequences due to indirectly invoking of processes – see Fig. 1. 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 *A*'s 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*)

or invokes (i.e., *SB*). Thus, to enhance one's 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.

We can state *ACG* formally as follows:

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

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.

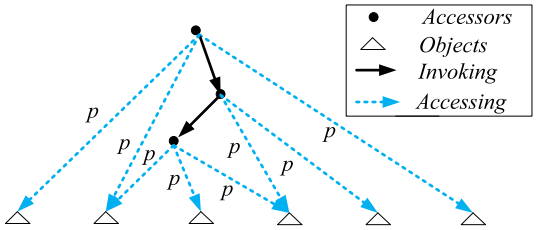

FIGURE 2. An *ACG* example.

IV. PROPOSED SCHEME

A. ACCESS CONTROL MODEL

Fig. 2 presents a simple *ACG*, which 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 (e.g., Fig. 1). 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 additional access privileges without permission), and to mitigate such collisions.

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 Cls_a) that returns all invoked accessors for a given accessor is defined as follows:

Definition 2: Closure process $Cls_a : a \in V_{Acc} \rightarrow A \subset V_{Acc}$. *$Cls_a$ takes as input an accessor $a \in V_{Acc}$ and outputs a set of invoked accessors $A \in V_{Acc}$. Initially, $A = \{a\}$. Next, let $A \Leftarrow \{v | \forall a \in A, e \in E_{Inv}, a = e.from, v = e.to\} \cup A$. Note that the above process is recursive.*

Given an accessor, all objects that can be accessed directly by this accessor are of interest. A closure function (denoted as Cls_o) that returns all directly accessible objects for a given accessor is defined as follows:

Definition 3: Closure function $Cls_o : a \in V_{Acc} \rightarrow O \subset V_{Obj}$. *$Cls_o$ takes as input an accessor $a \in V_{Acc}$ and outputs a set of accessed objects $O \subset V_{Obj}$. That is, $O = \{v | \forall a \in V_{Acc}, e \in E_{Acc}, a = e.from, v = e.to\}$.*

Given an accessor, all objects that can be accessed directly or indirectly can be returned by the following function, denoted as Cls_{ao}. It relies on $Cls_a(\cdot)$ as a subfunction.

Definition 4: Closure function $Cls_{ao} : a \in V_{Acc} \rightarrow O \subset V_{Obj}$. *$Cls_{ao}$ takes as input an accessor $a \in V_{Acc}$ and outputs a set of accessed objects $O \subset V_{Obj}$. That is, $O = \{v | \forall a' \in Cls_a(a), e \in E_{Acc}, a' = e.from, v = e.to\}$.*

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

Definition 5: Hamming distance function $HD : p_1 \in \{0,1\}^n \times p_2 \in \{0,1\}^n \rightarrow n \in \mathbb{Z}$. *HD takes as input $p_1, p_2 \in \{0,1\}^n$, and outputs n which is a Hamming Distance of p_1 and p_2.*

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

Definition 6: Privilege collisions for closure function $Cll_a : a \in V_{Acc} \rightarrow n \in \mathbb{Z}$. *$Cll_a$ takes as input an accessor $a \in V_{Acc}$ and outputs an integer number* $n = \max(\{HD(p_1, p_2) | e_1 \in E_1, e_2 \in E_2, p_1 = e_1.p, p_2 = e_2.p, e_1.to = e_2.to\})$, *where* $E_1 = \{e_1 | e_1 \in E_{Acc},$

$e_1.from = a, e_1.to \in Cls_o(a)$, $E_2 = \{e_2 | e_2.from \in Cls_a(a), e_2.to \in Cls_{ao}(a)\}$).

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

Theorem 1: $\forall a \in V_{Acc}$, *if* $Cll_a(a \in V_{Acc}) \neq 0$ *or* $Cls_{ao}(a) \cap Cls_o(a) \neq \emptyset$, *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 $Cls_o(a)$. *The accessors that are invoked by a are* $Cls_a(a)$. *Any privileges for a originally are* $e_1.p$, *where* $e_1.from = a, e_1.to \in Cls_o(a)$. *Any privileges for accessors invoked by a are* $e_2.p$, *where* $e_2.from \in Cls_a(a)$ *and* $e_2.to \in Cls_{ao}(a)$. *Their differences are* $HD(e_1.p, e_2.p)$, *where the maximal is* $Cll_a(a)$. *If* $Cll(a) \neq 0$, *then there is a privacy breach due to the invoking between accessors.*

Alternatively when $Cls_{ao}(a) \cap Cls_o(a) \neq \emptyset$, *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 simple function (denoted as Pri_c) on privilege collision, based on hamming distance.

Definition 7: Privilege collision function $Pri_c : p_1 \in \{0,1\}^n \times p_2 \in \{0,1\}^n \rightarrow b \in \{0,1\}$ *takes as input* $p_1, p_2 \in \{0,1\}^n$ *and outputs* $b = 0$ *if* $HD(p_1, p_2) \neq 0$ *(which means collision occurs). Otherwise,* $b = 1$.

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 $Pri_c(\cdot,\cdot)$ to a more general version, $Pri'_c(\cdot,\cdot)$, which specifies whether collision occurs for the given pairwise privileges. It is described as follows:

Definition 8: Privilege collision function $Pri'_c : p_1 \in Pri \times p_2 \in Pri \rightarrow b \in \{0,1\}$ *takes as input* $p_1, p_2 \in Pri$ *and outputs* $b = 0$ *if* p_1 *collides with* p_2 *(depending on concrete application logics). Otherwise,* $b = 1$.

We can map a privilege array consisting of $0, 1$ (e.g., $[1, 0, \ldots, 1, 0]$) 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 $A2PSet : p \in \{0,1\}^n \rightarrow pri_1, pri_2, \ldots, pri_n \in Pri, n = |Pri|$. $A2P$ *takes as input* $p \in \{0,1\}^n$ *and outputs a set of privileges, in which* Pri_i *is included if the i-th bit in array p is 1. Otherwise,* Pri_i *is excluded from the set.*

By using the above mapping process (i.e., $A2PSet(\cdot)$), 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 $A2C : p_1 \in \{0,1\}^n \times p_2 \in \{0,1\}^n \rightarrow b \in \{0,1\}$ *takes as input* $p_1, p_2 \in \{0,1\}^n$ *and outputs a* $b = 0$ *if* $\forall p_a \in A2PSet(p_1)$, $\exists p_b \in A2PSet(p_2)$ *such that* $Pri'_c(p_a, p_b) = 0$. *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 Cll'_a.

Definition 11: Privilege difference for closure function $Cll'_a : a \in V_{Acc} \rightarrow b \in \{0,1\}$ *takes as input an accessor* $a \in V_{Acc}$ *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 $E_1 = \{e_1 | e_1 \in E_{Acc}, e_1.from = a, e_1.to \in Cls_o(a)\}$, *and* $E_2 = \{e_2 | e_2.from \in Cls_a(a), e_2.to \in Cls_{ao}(a)\})\}$. *If* $b = 0$, *then collision occurs; otherwise, there is no privilege collision.*

We will now present a general conclusion.

Corollary 1: $\forall a \in V_{Acc}$, *if* $Cll'_a(a \in V_{Acc}) = 0$ *or* $Cls_{ao}(a) \cap Cls_o(a) \neq \emptyset$, *then privacy breaches due to inter-networking linkages.*

Proof 2: Straightforward due to Theorem 1. □

Usually, $\forall pri_1, pri_2 \in Pri$, $pri_1 < pri_2$, where $<$ means "implied by". For example, "*read*" is implied by "*write*", that is, *read* $<$ *write*. Suppose $Pri = [pri_1, \ldots, pri_n]$, and $pri_1 < pri_2 < \ldots < pri_n$. Thus, we can map privileges into ordered integers, such as $1, 2, \ldots, n$. Clearly, only the largest one is sufficient to denote privileges. Thus, $\forall e \in E_{Acc}$, $e = \langle from, to, p \rangle$ where $from \in V_{Acc}$, $to \in V_{Obj}$, $p \in \mathbb{Z}$. For this situation, we introduce $Cll''_a(\cdot)$ as follows:

Definition 12: Privilege difference for closure function $Cll''_a : a \in V_{Acc} \rightarrow b \in \{0,1\}$ *takes as input an accessor* $a \in V_{Acc}$ *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 $E_1 = \{e_1 | e_1 \in E_{Acc}, e_1.from = a, e_1.to \in Cls_o(a)\}$, $E_2 = \{e_2 | e_2.from \in Cls_a(a), e_2.to \in Cls_{ao}(a)\})\}$, *and* $IsLEQ(c \in \mathbb{N}, d \in \mathbb{N})$ *returns 1 if* $c \leq d$ *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: $\forall a \in V_{Acc}$, *if* $Cll''_a(a \in V_{Acc}) = 0$ *or* $Cls_{ao}(a) \cap Cls_o(a) \neq \emptyset$, *then privacy breaches due to linkage.*

Proof 3: Straightforward due to Theorem 1. □

B. 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 $e \in E_{Inv}$ *and invoking from accessor e.from. If* $e'_1.to = e'_2.to$, *where* $e'_1, e'_2 \in E_{Acc}$, $e.from = e'_1.from$, $e.to = e'_2.from$, *then let* $e'_2.p = e'_1.p$. *Otherwise, let* $e'_2.to = \perp$.

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 $\exists e_1, e_2, \ldots, e_n \in E_{Inv}$, $n \geq 2$ *such that* $e_1.to = e_2.from$, $e_2.to = e_3.from$, $\ldots$, $e_i.to = e_{i+1}.from$, $\ldots$, $e_{n-1}.to = e_n.from$, *then we denote this invoking path as* $Path_{Inv} = [e_1, e_2, \ldots, e_n]$.

Proposition 2: Regulating Privileges of Invoked Processes from an Accessor (Rule II: Privilege is non-increasing across an invoking path). Suppose $Path_{Inv} = [e_1, e_2, \ldots, e_n]$, $n \geq 2$ *exists and invoking is from accessor* $e_1.from$. *If* $\exists i, j \in \{1, 2, \ldots, n\}$, $i < j$ *such that* $e'_i.to = e'_j.to$, *where* $e'_i, e'_j \in E_{Acc}$, $e_i.from = e'_i.from$, $e_j.from = e'_j.from$, *then let* $e'_j.p = e'_i.p$. *If* $e'_i.to \neq e'_j.to$, *then let* $e'_j.to = \perp$.

That is, the above rule states that if an accessor (e.g., process A) invokes multiple accessors sequentially (e.g., processes B_1, B_2,...,B_n), then the privilege of invoked accessor (i.e., B_i, $i = 2, \ldots, n$) must be less than or equal to the privilege of invoking accessor (i.e., B_{i-1}) once the same object is accessed. The access control unit will normalize the privilege of invoked process B_i when B_i is invoked by B_{i-1} (for $i = 2, 3, \ldots, n$). For $i = 1$, B_1 is invoked by A, which is degenerated to Rule I.

Definition 14: Invoking Loop. If $\exists e_1, e_2, \ldots, e_n \in E_{Inv}$, $n \geq 2$ *such that* $e_1.to = e_2.from$, $e_2.to = e_3.from$, $\ldots$, $e_i.to = e_{i+1}.from$, $\ldots$, $e_{n-1}.to = e_n.from$, $e_n.to = e_1.from$, *then denote this invoking loop as* $Loop_{Inv} = [e_1, e_2, \ldots, e_n]$.

Proposition 3: Regulating Privileges of Invoked Processes from an Accessor (Rule III: Privilege is minimized in an invoking loop). Suppose $Loop_{Inv} = [e_1, e_2, \ldots, e_n]$, $n \geq 2$ *exists and invoking is from any accessor in* $\{a | e_i.from = a, i = 1, 2, \ldots, n\}$. *If* $\exists i, j \in \{1, 2, \ldots, n\}$, $i < j$ *such that* $e'_i.to = e'_j.to$ *and* $e'_i.p > e'_j.p$, *where* $e'_i, e'_j \in E_{Acc}$, $e_i.from = e'_i.from$, $e_j.from = e'_j.from$, *then let* $e'_i.p = e'_j.p$. *If* $e'_i.to \neq e'_j.to$, *then* $e'_j.to = \perp$.

In other words, the above rule states that if an accessor (e.g., process A) invokes multiple accessors sequentially (e.g., processes B_1, B_2,...,B_n), and forms a loop (i.e., B_n invokes A, then the privilege of invoked accessor (i.e., B_i, $i = 1, 2, \ldots, n$) 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 B_i, $i = 1, 2, \ldots, n$ 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 $\exists e \in E_{Inv}$. *If* $e.to$ *is invoked by* $e.from$, *and* $e_1.to = e_2.to$, *where* $e_1, e_2 \in E_{Acc}$, $e.from = e_1.from$, $e.to = e_2.from$, *then let* $e_1.p = e_2.p$. *If* $e_1.to \neq e_2.to$, *then* $e_1.to = \perp$.

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 $\exists e \in E_{Inv}$. *If* $\exists v \in Cls_a(e.from)$, $e_1.to = e_2.to$, *where* $e_1, e_2 \in E_{Acc}$, $e.from = e_1.from$, $v = e_2.from$, *then let* $e_2.p = e_1.p$. *If* $\exists v \in Cls_a(e.from)$, $e_2.to \neq e_1.to$, *where* $e_1, e_2 \in E_{Acc}$, $e.from = e_1.from$, $v = e_2.from$, *then* $e_2.to = \perp$.

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 $v \in V_{Inv}$, *if* $\exists e_1 \in\in E_{Inv}, e_2, e_3 \in E_{Acc}$ *such that* $e_1.from = f$, $e_1.to = t$, $e_2.from = f$, $e_2.to = o$, $e_3.from = t$, $e_3.to = o$, *then let* $e_3.p \Leftarrow e_2.p$ *or* $e_3.p \leq e_2.p$.

The above rule is iterative for any inter-networking processes, which becomes a container for regulating the maximum privilege for any invoking processes.

C. PROPOSED ALGORITHMS

We will now present our 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 $Cls_a(\cdot)$. Using this function, all invoked accessors, directly or indirectly, can be returned and further examined.

Algorithm 1 Compute All Accessors Invoked by Any Given Accessor $v \in V_{Acc}$

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

```
A ⇐ {v};
while A ≠ ∅ do
    Select a ∈ A;
    while ∃e ∈ E_Inv such that e.from = a do
        A ⇐ A ∪ {e.to};
    end
    A ⇐ A − {a};
end
return A;
```

Algorithm 2 is a recursive algorithm that can obtain all *accessible* objects, directly and indirectly. It can be considered an instantiation of $Cls_{ao}(\cdot)$. In this algorithm, $e.to$ denotes the objects that $e.from$ can access with

corresponding privileges $e.pri$. $v.visit \in \{0, 1\}$, *visit* is a label to denote whether a vertex has been visited.

Algorithm 2 Compute All Accessible Objects and Corresponding Privileges That *e.from* Possess

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.

Algorithm 3 Detect Privilege Collisions

Data: ACG
Result: *Yes*, *No*

```
for each v in V_Acc do
    if v.visit ≠ 1 then
        v.visit = 1; /* denote the current vertex v has been visited */
        find e1, e2 ∈ E_Acc where e1.from = v, e2.from ∈ Cls_a(v);
        if (e1.to = e2.to and e1.pri ≠ e2.pri) or (e1.to ≠ e2.to) then
            return Yes;
        end
    end
end
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.

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.

Algorithm 4 Regulate Privileges

Data: ACG
Result: ACG'

```
for each v in V_Acc do
    if v.visit ≠ 1 then
        v.visit = 1; /* denote the current vertex v has been visited */
        find e1, e2 ∈ E_Acc where e1.from = v, e2.from ∈ Cls_a(v);
        if e1.to = e2.to and e1.pri ≠ e2.pri then
            e2.pri = e1.pri;
        end
        if e1.to ≠ e2.to then
            e2.to = NULL;
        end
    end
end
return ACG';
```

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.

V. 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 $O(|V_{Acc}|)$, where where V_{Acc} is the number of accessor vertexes in ACG. □

VI. CONCLUSION

In this paper, we proposed a lightweight graph-based model for access control among inter-networking processes. Our 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.

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

ACKNOWLEDGEMENT

The authors thank the discussions and comments from W. Jiang, Y. Chen, Z. Kang, and M. Lei.

REFERENCES

[1] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.
[2] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[3] B. W. Lampson, "Protection," in *Proc. 5th Princetion Symp. Inf. Sci. Syst.*, vol. 1, 1974, pp. 18–24.
[4] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," in *Proc. Int. School Found. Secur. Anal. Des.*, 2001, pp. 137–196.
[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. 224–274, 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. 166–181.
[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. 1557–1577, 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. 10–20.
[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. 379–396.
[10] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.
[11] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[12] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 735–737.
[13] E. Yuan and J. Tong, "Attributed based access control (ABAC) for Web services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2005, p. 569.
[14] 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. 74–79.
[15] 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. 309–318.
[16] 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. 222–230.
[17] K. Z. Bijon, R. Krishnan, and R. Sandhu, "A framework for risk-aware role based access control," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 462–469.
[18] 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. 1614–1627, Jul. 2013.
[19] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "Semantic Web-based social network access control," *Comput. Secur.*, vol. 30, nos. 2–3, pp. 108–115, 2011.
[20] 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. 29–35, Nov. 2017.

SIMIN LI is currently pursuing the degree with the School of Computer Science, China University of Geosciences, Wuhan, China. Her research interests include access control and privacy protection.

WEI REN (M'09) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China. He was with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, USA, from 2007 to 2008, the School of Computer Science, University of Nevada Las Vegas, USA, from 2006 to 2007, and the Department of Computer Science, The Hong Kong University of Science and Technology, from 2004 to 2005. He is currently a Professor with the School of Computer Science, China University of Geosciences, Wuhan, China. He has published more than 70 refereed papers, one monograph, and four textbooks. He holds 10 patents. He is a Senior Member of the China Computer Federation. He received five innovation awards.

TIANQING ZHU received the B.Eng. and M.Eng. degrees from Wuhan University, China, in 2000 and 2004, respectively, and the Ph.D. degree in computer science from Deakin University, Australia, in 2014. She was a Lecturer with the School of Information Technology, Deakin Unversity, from 2014 to 2018. She is currently a Senior Lecturer with the School of Software, University of Technology Sydney, Australia. Her research interests include privacy preserving, data mining, and network security.

KIM-KWANG RAYMOND CHOO (SM'15) received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio (UTSA) and has a courtesy appointment at the University of South Australia. He is a fellow of the Australian Computer Society and an Honorary Commander of the 502nd Air Base Wing, Joint Base San Antonio-Fort Sam Houston. In 2016, he was named the Cybersecurity Educator of the Year—APAC (Cybersecurity Excellence Awards are produced in cooperation with the Information Security Community on LinkedIn), and in 2015, he and his team won the Digital Forensics Research Challenge organized by the University of Erlangen-Nuremberg, Germany. He was a recipient of the 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty, the ESORICS 2015 Best Paper Award, the 2014 Highly Commended Award of the Australia New Zealand Policing Advisory Agency, a Fulbright Scholarship in 2009, the 2008 Australia Day Achievement Medallion, and the British Computer Society's Wilkes Award in 2008.