Research Article

A Noninterference Model for Mobile OS Information Flow Control and Its Policy Verification

Zhanhui Yuan¹, Wenfa Li², Zhi Yang¹, Lei Sun¹, Xuehui Du¹ and Hongqi Zhang¹

¹Information Engineering University, Zhengzhou 450001, China

²Beijing Union University, Beijing 100101, China

Correspondence should be addressed to Wenfa Li; 644085325@qq.com

Received 30 April 2021; Accepted 7 August 2021; Published 25 August 2021

Academic Editor: Joni Zhong

Copyright © 2021 Zhanhui Yuan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Mobile operating systems such as Android are facing serious security risk. First, they have a large number of users and store a large number of users' private data, which have become major targets of network attack; second, their openness leads to high security risks; third, their coarse-grained static permission control mechanism leads to a large number of privacy leaks. Recent decentralized information flow control (DIFC) operating systems such as Asbestos, HiStar, and Flume dynamically adjust the label of each process. Asbestos contains inherent covert channels due to this implicit label adjustment. The others close these covert channels through the use of explicit label change, but this impedes communication and increases performance overhead. We present an enhanced implicit label change model (EILCM) for mobile operating systems that can close the known covert channel in these models with implicit label change and supports dynamic constraints on tags for separation of duty. We also formally analyze the reasons why EILCM can close the known covert channels and prove that abstract EILCM systems have the security property of noninterference with declassification by virtue of the model checker tool FDR. We also prove that the problem of EILCM policy verification is NP-complete and propose a backtrack-based search algorithm to solve the problem. Experiments are presented to show that the algorithm is effective.

1. Introduction

At present, mobile operating systems have a large number of users and are key targets of network attack. Their privacy security problem is very serious. In 2020, China Internet Security Center intercepted 332000 new malicious app samples related to privacy theft on mobile operating systems and monitored 1.12 billion malicious attacks with an average of 3.06 million attacks per day. The openness of the mobile operating system leads to high security risks. The characteristics of mobile system interacting with the outside from multiple channels make their vulnerabilities easier to be exposed and utilized. The quality and security of apps in online app stores (such as Google play) are uneven, and it is difficult to ensure that there are no Trojans and viruses implanted by hackers. Hackers can steal data by embedding spy programs into normal applications to obtain illegitimate benefits [1]. Mobile operating systems are facing more severe privacy and security challenges than

traditional PC operating systems [2–4]. Most mobile systems adopt discretionary access control (DAC) as the security model, but the DAC policy cannot effectively express the information flow control [5], which makes it difficult to effectively prevent Trojan stealing secret data. Mobile operating systems need to implement mandatory access control [6]. The essence of mandatory access control is to prevent information in a secret domain from leaking out of the domain. However, for the mobile operating systems with diversified and dynamic interaction with the outside world, the static isolation and blocking method cannot effectively balance availability and security [7].

Decentralized information flow control (DIFC) [8] addresses the weaknesses of earlier approaches to the protection of secrecy and integrity in an operating system containing applications with potential exploits or third-party components [9], even in situations of mutual distrust [7]. DIFC allows processes to control the flow of their

information without imposing the rigid constraints of a traditional multilevel security system [10]. DIFC allows programs to craft their own MAC security schemes. Each user has the power to declassify his own data, but declassification does not leak other users' data. Even if a process is compromised, it will not endanger the data security of others [11]. DIFC has been widely applied to the protection of operating systems [12–18], databases [19], programming languages [6, 20, 21], web applications [22], etc.

Asbestos [16], HiStar [17], and Flume [18] are classical DIFC operating systems. Asbestos adopts a “taint-propagation” policy, wherein each process P has two labels, a send label P_S and a receive label P_R . P_S represents the process's current contamination, while P_R represents the maximum contamination it is able to accept from others. P may send to Q if P_S is not more than Q_R , once communication is tainted by P . Q_S is the least upper bound on the two send labels.

Despite its benefits, the Asbestos design contains inherent covert channels. Consider a contaminated process A attempting to communicate a bit of sensitive information to an uncontaminated process C . An attacker might construct two uncontaminated processes. These processes, B_0 and B_1 , both repeatedly send heartbeat messages to C . If it wants to communicate the value i to C , A will send a message to the B_i . If B_i tries to receive it, it will be contaminated and become a tainted process. If C does not receive a heartbeat message from B_i in the next time slot, it knows that A has sent value i to it. Now C has the sensitive information, copied bit-for-bit from A . There are many deformations of the covert channel, and if this mechanism is used for integrity protection, this covert channel still exists. Such channels are inherent to any system in which a contaminated process can change the labels of a less contaminated process implicitly. Because Asbestos's event process abstraction makes forking very fast, this channel on Asbestos can leak kilobits of data per second [18].

HiStar is a successor system to Asbestos. It avoids these channels by requiring the processes to contaminate themselves. The HiStar thread actively changes its label before reading the contaminated data. Flume later adopted a similar strategy, but within a streamlined label system. However, compared with implicit label changes, explicit label changes mean that the processes become more involved in information flow control, which will result in poor communication if these processes cannot adjust their label in a timely manner. Moreover, this also introduces performance overhead due to the system calls relative to it.

This paper presents a model for information flow control at the operating system level, called enhanced implicit label change model (EILCM), to solve the contradiction. This model uses Flume labels and capabilities in terms of their set operations based on type tags, a well-understood formalism to express the dynamic processes isolation which can effectively prevent Trojans and spyware attacks. Our model has the following characteristics: firstly, it allows implicit label change but tries to close the known covert channels; secondly, it solves the contradictions between users' sharing of common software and authorization constraints for

separation of duty or least privilege through mutually exclusive tag constraints; and thirdly, its rule covers more information flow types such as file operation and program execution in addition to inter-process communication (IPC). We do not introduce privilege transfer or delegation of authority, considering that it is difficult for most commercial DIFC programs to predict their future runtime environment or to know in what situation it does or does not transfer privilege. In that case, policy management tasks will be ultimately passed on from the DIFC program to the system administrators and users. In contrast, delegation of authority adds more uncontrollable factors.

In order to prove that there are no covert channels in an EILCM system, we adopt an approach similar to that of Krohn et al. [18]. We describe the model specification using the communicating sequential processes (CSP) to clarify the formal semantics of information flow control behaviors of the EILCM system; then we prove that an abstract EILCM system has the security property of noninterference based on CSP formulas. Our methods and that of [18] have the following differences. (1) Our definition of noninterference with declassification uses a standard CSP-oriented specification. Therefore, our proof uses both automated and manual analysis, while Krohn and Tromer's definition [18] is property-oriented specification, so its proof is completely manual. (2) Our model adopts the centralized control of information flow and captures more information flow types, while the Flume model of [18] adopts distributed control and concentrates on IPC control, resulting in distinct differences in their formal semantics.

Another problem studied in this paper is the verification of EILCM policies. Because of the richness and complexity of label management, whether the label configurations completely and consistently express the high-level policies and goals is an important problem to verify. In the traditional one-way information flow model, policy verification is quite simple. However, in some flexible systems to control information flow, such as Vista, Asbestos, Flume, etc., policy verification problems are complex [23, 24] due to their introduction of declassification functions or transfer privilege functions. These functions diffuse the individual policies and obscure their combined effect. Misconfigurations may make processes lose due privileges or obtain unduly privileges via a privilege exploit attack. Policy analysis and verification are especially necessary for the large systems that have hundreds of processes or users and tens of thousands of resources.

In sum, the novel contributions of this paper are as follows:

- (i) In Section 2, we present an enhanced implicit label change model (EILCM) that can close the known covert channels in the existing implicit label change models for information flow control at the operating system level. Moreover, the EILCM introduces mutually exclusive tag constraints to solve the contradictions between users' sharing of common software and authorization constraints for

separation of duty or least privilege. Based on the ISO general access control framework, we also use the CSP language to describe the formal semantics of information flow control behaviors of EILCM systems, which lays the foundation for non-interference proofs.

- (ii) In Section 3, we formally analyze the reason that EILCM can close the known covert channels. Further, through the model checker tool FDR, we prove that an abstract EILCM system has the security property of noninterference with declassification.
- (iii) In Section 4, we prove that the problem of EILCM policy verification is an NP-complete problem. This conclusion reflects, to some extent, the computational complexity of the problem of Flume policy verification, due to their partial similarity. We also present a backtrack-based search algorithm to solve the problem. Experiments are presented to show that the algorithm is effective.

We discuss related work in Section 5 and conclude in Section 6.

2. EILCM Descriptions

2.1. Informal Descriptions of EILCM. In an EILCM protection system, let O be the set of all objects (e.g., files) and U be the set of all subjects (e.g., processes). Let DS be the set of secrecy (export) protection tags, with each tag denoting a secrecy domain type. It is with its secrecy level containing the same tag x that a requestor can observe the information in an x -type secrecy domain. Correspondingly, let DI be the set of integrity (import) protection tags, with each tag denoting an integrity domain type. A requestor whose integrity level does not contain the same y cannot alter the information in a y -type integrity domain.

Definition 1 (label). For each object or subject $p \in O \cup U$, its label L_p is a triple (S_p, I_p, C_p) , where $L_p \subseteq DS$ denotes its secrecy level, $I_p \subseteq DI$ denotes its integrity level, and $C_p \subseteq (DS \cup DI) \times \{+, -\}$ denotes its management capability for its own label.

Secrecy levels and integrity levels, respectively, form a lattice under the partial order of the set relation $\subseteq$, such that $S_p \geq S_q$ if and only if $S_p \supseteq S_q$, and $I_p \geq I_q$ if and only if $I_p \supseteq I_q$. In EILCM, information can flow from low secrecy-level and high integrity-level objects or subjects to high secrecy-level and low integrity-level objects or subjects.

A subject p with $d^+ \in C_p$ can add a tag d to its own S_p if $d \in DS$ or to its own I_p if $d \in DI$, while with $d^- \in C_p$, it can remove d from its own S_p or I_p . Generally speaking, for a secrecy tag d_1 , an untrustworthy process p with $d_1^+ \in C_p$ and $d_1^- \notin C_p$ can read a secret file f with $S_f = \{d_1\}$ by raising its own to $S_p \cup \{d_1\}$ without the ability to reveal it. If $d_1^- \in C_p$, the system conservatively assumes that p has declassification ability and it does not propagate f 's data. If there are multiple tags in S_f , then p requires independent consent for each tag before reading f . Similarly, for a integrity tag d_2 ,

a system-level process q with $d_2^- \in C_q$ and $d_2^+ \notin C_q$ can receive network data by reducing its own I_q to $I_q - \{d_2\}$; however, once it reads this unsafe data, it loses the ability to modify the system configurations. If $d_2^+ \in C_q$, the system conservatively assumes q has decontamination ability and it can still modify the system configurations.

The subject's capability is derived from the properties of the executable object where it was born, which is set by the authority administrator. We define $C_o = \varphi$, $o \in O$, which means that an object cannot change its level itself.

To simplify notation, a subject p 's "add-tag" capability is given by $C_p^+ = \{d \mid d^+ \in C_p\}$, and its "remove-tag" capability is given by $C_p^- = \{d \mid d^- \in C_p\}$. The notation $C_p^\pm = \{d \mid d^+ \in C_p \wedge d^- \in C_p\}$ represents p 's dual-privilege capability.

Definition 2 (constraints). $R \subseteq 2^{DS} \cup 2^{DI}$ is a set of mutually exclusive tag constraints with the property that for each $\{d_1, d_2, \dots, d_m\} \in R$, the secrecy/integrity level X of any object or subject satisfies $|X \cap \{d_1, d_2, \dots, d_m\}| < 2$, which we call "XsatR."

We introduce mutually exclusive tag constraints to solve the contradictions between users' sharing of common software and authorization constraints for separation of duty or least privilege.

Example 1 (secrecy/least privilege). Alice and Bob share the word-processing program w to, respectively, edit their private documents with their secrecy levels ds_A and ds_B . Then, w 's capability should involve ds_A^+ and ds_B^+ in order to read these documents. Assume that Alice executes w to generate a new process w_A with $ds_A \in S_{w_A}$ using her shell process, and w_A inherits w 's capability. In order to prevent Alice from reading Bob's document by means of w_A , w_A should be deprived of ds_B^+ capability at that time, which means that the process should run with as few privileges as possible. We can solve this problem by defining constraint $\{ds_A, ds_B\} \in R$.

Example 2 (integrity/separation of duty). General information systems emphasize the separation of powers and do not allow any individual to have sufficient authority within the system to single handedly perpetrate fraud. The normal division of authority is into security management, system configurations, and audit management. Assume these management domains are denoted, respectively, by integrity tags of di_m , di_c , and di_a , and the domain administrators are used to using same program w to modify their domain information, which requires that w 's capability should contain di_m , di_c , and di_a together. However, we need be assured that when it is created by some domain administrator and derived from w , the new process has no chance of modifying other domain information although it inherits w 's capability. We can solve this problem by defining the constraint $\{di_m, di_c, di_a\} \in R$.

In order to prevent any subject from implicitly bypassing these constraints, EILCM does not entitle any subject to delete those tags that exist in R from its level.

Further, we introduce the relation s_sat for Definition 3. Given an object or subject p in an EILCM system, we say ps_satR if and only if $\forall r \in R, |r \cap C_p^+ \cap DS| > 1 \implies |r \cap S_p| = 1$.

Before defining the information flow rule, we reconsider the example of covert channels mentioned in Section 1. The former models assume that if B_i tries to receive a message from A when A does not send anything, B_i should not be contaminated. However, no transmitting data actually should be regarded as a kind of implicit communication without formal sending and receiving behavior if planned by adversaries. Therefore, in the EILCM, if it can be contaminated within its capability and tries to receive a message from A , B_i will be contaminated automatically whether or not A sends a message to it. In addition, HiStar and Flume introduce the explicit change label method, in which each process must make a decision itself about whether to adjust its level if it wants to communicate with other process successfully. We will formally analyze the reason why these methods can close the known covert channels in Section 3.1. Compared with our method, the explicit change label method increases the process's interventions, which results in unsmooth communication and introduces overhead for system calls.

Definition 3 (secure information flow and implicit change label). The information flow from p to q is safe iff

$$\begin{aligned} S_p - C_p^\pm &\subseteq S_q \cup C_q^\pm, \\ I_p \cup C_p^\pm &\supseteq I_q - C_q^-, \\ q &s\_sat R. \end{aligned} \quad (1)$$

When the flow is finished, q 's level is implicitly changed, that is:

$$\begin{aligned} S_q &\leftarrow S_q \cup (S_p - C_p^\pm), \\ I_q &\leftarrow I_q - (I_q - (I_p \cup C_p^\pm)). \end{aligned} \quad (2)$$

Definition 3 demands that secure information flow is from low secrecy-level and high integrity-level objects or subjects to high secrecy-level and low integrity-level objects or subjects, where both party's capabilities of adjusting their own level are considered adequate. For different types of concrete operations, its operational semantics are as follows:

- (i) Subject p can receive message from subject q , iff
- (a) $p s\_sat R$
  - (b) q exists
  - (c) $S_q - C_q^\pm \subseteq S_p \cup C_p^+$ and $I_q \cup C_q^\pm \supseteq I_p - C_p^-$
  - (d) $S_p \cup (S_q - C_q^\pm) sat R$ (e) q has sent message to p .

When p triggers the related receive system call, p 's level will be adjusted according to

If (a)–(e) hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup (S_q - C_q^\pm), \\ I_p &\leftarrow I_p - (I_p - (I_q \cup C_q^\pm)). \end{aligned} \quad (3)$$

If (a) holds and (b) or (c) or (d) does not hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup (C_p^+ \cap DS) - [R], \\ I_p &\leftarrow I_p - C_p^-. \end{aligned} \quad (4)$$

Otherwise, p 's level will remain unchanged.

- (ii) Subject p can write/delete object q , iff

- (a) q exists
- (b) $S_p - C_p^\pm \subseteq S_q$ and $I_p \cup C_p^\pm \supseteq I_q$

- (iii) Subject p can read/load object q , iff

- (a) $p s\_sat R$
- (b) q exists
- (c) $S_q \subseteq S_p \cup C_p^+$ and $I_q \supseteq I_p - C_p^-$
- (d) $S_p \cup S_q sat R$

When p triggers the related read/load system call, p 's level will be adjusted according to

If (a)–(d) hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup S_q, \\ I_p &\leftarrow I_p - (I_p - I_q). \end{aligned} \quad (5)$$

If (a) holds and (b) or (c) or (d) does not hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup (C_p^+ \cap DS) - [R], \\ I_p &\leftarrow I_p - C_p^-. \end{aligned} \quad (6)$$

Otherwise, p 's level will remain unchanged.

- (iv) Subject p can create object q with its secrecy/integrity levels being S_q/I_q , iff

- (a) q does not exist
- (b) $S_p - C_p^\pm \subseteq S_q$ and $I_p - C_p^- \supseteq I_q$

- (v) Subject p can execute object o , and create a new subject, iff

- (a) $p s\_sat R$
- (b) o exists
- (c) $S_o \subseteq S_p \cup C_p^+$ and $I_o \supseteq I_p - C_p^-$
- (d) $S_p \cup S_o sat R$
- (e) $S_p - C_p^\pm \subseteq S_o \cup C_q^+$ and $I_p \cup C_p^\pm \supseteq I_o - C_q^-$,

where q is the id that the system allocates for the new subject and q 's capability is derived from the properties of o .

When p triggers the execute system call, p 's level will be adjusted according to

If (a)–(e) hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup S_o, \\ I_p &\leftarrow I_p - (I_p - I_o). \end{aligned} \quad (7)$$

If (a) holds and (b) or (c) or (d) or (e) does not hold, then

$$\begin{aligned} S_p &\leftarrow S_p \cup (C_p^+ \cap DS) - [R], \\ I_p &\leftarrow I_p - C_p^-. \end{aligned} \quad (8)$$

Otherwise, p 's level will remain unchanged.

When q is born, its level will be set according to

$$\begin{aligned} S_q &\leftarrow (S_p - C_p^\pm) \cup S_o, \\ I_q &\leftarrow I_p \cap I_o, \end{aligned} \quad (9)$$

and its capability will be set according to

$$\begin{aligned} C_q^+ &\leftarrow C_p^+ \cap C_o^+, \\ C_q^- &\leftarrow C_p^- \cap C_o^-. \end{aligned} \quad (10)$$

This process can be broken down into the process of p “reading” o and of p “writing” q . In the former process, p can know, at least, whether o exists. In the latter process, the parameters are the information p directly tells q if p starts q with parameters. Without parameters, p starting q is just the sensitive information that p tells q if only p can start q .

Definition 4 (secure explicit change label). For a subject p , let X be S_p or I_p and let X' be the new value that p requests about X . The change from X to X' is safe if and only if

$$\begin{aligned} X' - X &\subseteq C_s^+, \\ X - X' &\subseteq C_s^-, \\ X' &\text{ sat } R. \end{aligned} \quad (11)$$

Correspondingly, for an object o and a subject p , let Y be S_o or I_o and let Y' be the new value that p requests about Y . The change from Y to Y' is safe iff

$$\begin{aligned} X - C_s^\pm &\subseteq Y \subseteq X \cup C_s^\pm, \\ X - C_s^\pm &\subseteq Y'. \end{aligned} \quad (12)$$

The subject can adjust their own level within their capacity. For example, a word-processing process can produce the first confidential documents in the system by raising its own secrecy level, and PGP software can reduce its own secrecy level in order to send the encrypted document. The explicit label change is an effective supplement to implicit label change. Now, the condition of secure information flow can be relaxed and becomes $S_p - C_p^\pm \subseteq S_q \cup C_q^+$ and $I_p - C_p^\pm \subseteq I_q \cup C_q^+$.

The requirement for a subject adjusting an object's level is consistent with that of the subject adjusting its own level and Definition 3. In the same condition, a subject p can finish adjusting an object o 's level through the following process: p reads the contents of $o \rightarrow p$ writes content to a temp file $f \rightarrow p$ deletes $o \rightarrow p$ creates o with level being $Y' \rightarrow p$ reads $f \rightarrow p$ writes $o \rightarrow p$ deletes f . We specify this requirement to simplify the system processes.

Based on the above definition, the next section gives the precise semantics of the model.

2.2. Formal Specifications of EILCM. In order to formally analyze the security of EILCM and the known covert channels, we describe EILCM's formal specifications using communicating sequential processes (CSP) [25]. CSP can capture effectively the required or expected behaviors of a concurrency system or component, and this is suitable for analyzing the

security properties of the information flow model [26]. These specifications involve file operation, program execution, inter-process communication, etc. Since they are primary, a real EILCM system might not be constructed from them. For example, the specifications do not capture how to reliably obtain the object label but assume that all object labels have been stored in the object manager; the specifications do not capture the reliable communication but simply make the new message cover the old message that has not been taken away when the message buffer is full; the specifications do not capture the fairness scheduling of processes but permit any interleaving of processes. However, our specification is fundamental. It expresses the essential safety requirements of EILCM and is easy to expand, refine, and adjust in its nonsecurity aspects.

Figure 1 depicts the composition and control principles of EILCM. We refer to the ISO general access control framework, which isolates access control policy decisions and enforcement to provide flexibility in access control. Our framework mainly consists of Access Control Enforcement Function (AEF), Access Control Decision Function (ADF), and Object Management (OM). Each subject i calls the system service via channel api_i , which is captured by AEF. AEF will get a variety of Access Decision Information (ADI) about this system call from OM via the channel $q\text{state}$ and then sends ADI to ADF via the channel $q\text{policy}$. Based on ADI and information flow rules, ADF makes a decision and returns it to AEF. Finally, AEF enforces this decision. The AEF and OM constitute the security kernel. The security kernel should be small enough to ensure efficiency and facilitate security analysis.

2.2.1. User Processes. In general, the user processes access system resources through system calls. There is no direct interaction between them, so we use the interleaving operator to express their relations. The notation USER_i denotes the user process i . The behaviors of all user processes are specified as follows:

$$\text{USERS} = (|||i: P - \{1\} @ \text{USER}_i ||| \text{USER\_LIVE}_1, \quad (13)$$

where P denotes the set of all process identifiers and USER_LIVE_1 denotes the first process with $C_{\text{USER\_LIVE}_1} = DS \cup DI \times \{+, -\}$ at the system startup, which can execute any object to create a new process for the startup needs. There is no deriving operator in CSP, but the actual systems usually restrict the maximum size of a process; thus, we can prebuild all the possible user processes. Any processes will be in an idle state before they receive the signal. The signal born makes it transition into the live state, and the exit signal makes it return to the idle state again. The behaviors of a single user process are specified as follows:

$$\text{USER}_i = \text{born}_i ? \text{msg} \rightarrow \text{USER\_LIVE}_i. \quad (14)$$

In the live state, USER_i can issue a system call:

$$\begin{aligned} \text{USER\_LIVE}_i = & \text{SEND}_i | \text{RECV}_i | \text{WRITE}_i | \text{READ}_i | \text{CREATE}_i \\ & | \text{DEL}_i | \text{EXEC}_i | \text{EXIT}_i | \text{CHANGELABEL}_i. \end{aligned} \quad (15)$$

FIGURE 1: Composition and control principles of EILCM.

The subprocess $SEND_i$ describes sending a message msg to user process $USER_{to}$:

$$SEND_i = api_i!(send, to, msg) \dots \longrightarrow USER\_LIVE_i. \quad (16)$$

The subprocess $RECV_i$ describes receiving a message from $USER_{from}$, where msg is the receive buffer and $result = \{ok, error\}$ returns the success/failure result of this system call.

$$RECV_i = api_i!(recv, from) \longrightarrow api_i?(result, msg).. \longrightarrow USER\_LIVE. \quad (17)$$

The subprocess $EXEC_i$ describes executing an object which, where OM allocates randomly the PID of the new process.

$$EXEC_i = api_i!(exec, which, msg) \longrightarrow USER\_LIVE_i. \quad (18)$$

The subprocess $EXIT_i$ describes process's state transition from live to idle.

$$EXIT_i = api_i!(exit) \longrightarrow USER_i. \quad (19)$$

The subprocesses $WRITE_i$, $READ_i$, $CREATE_i$, DEL_i , and $CHANGELABEL_i$ have similar behaviors, and we omit the description of their behaviors.

2.2.2. AEF. AEF receives and processes concurrently system calls from different users, each subprocess AEF_ITEM_i listening to the corresponding $USER_i$'s calls via channel api_i and controlling access to the system. The process SYN_SO guarantees the state consistency of the security kernel when more than two system calls involve the same subject or object simultaneously.

$$AEF = \left((|||i: P@AEF\_ITEM_i) \parallel_{aSYN} SYN\_SO \right). \quad (20)$$

Each system call gets its own dedicated subprocess, and AEF_ITEM_i consists of these subprocesses through the choice operator.

$$AEF\_ITEM_i = AEF\_SEND_i | AEF\_RECV_i | AEF\_EXEC_i | \dots \quad (21)$$

The process SYN implements the "mutex" lock by virtue of the parallel operator in order to serialize the access to the same subject or object. In order to prevent deadlock in locking the multiple subject or object involved in a system call, the priority of locking is set by subject first, and from small id to large id.

$$SYN = SYN\_S ||| SYN\_O. \quad (22)$$

The alphabet of process SYN is $aSYN = \{enterS, leaveS, enterO, leaveO\}$.

The subject-access-synchronism process is $SYN\_S = |||i: P@SYN\_S_i$, where $SYN\_S_i = enterS.i \longrightarrow leaveS.i \longrightarrow SYN\_S_i$.

Similarly, the object-access-synchronism process is $SYN\_O = |||i: O@SYN\_O_i$, where $SYN\_O_j = enterO.j_j \longrightarrow leaveO.j \longrightarrow SYN\_O$, and O is the set of all possible object ids.

We set the buffer to cache and forward the messages in process communication. If $USER_i$ wants to send msg to $USER_j$, AEF_SEND_i will first set the msg level in terms of the current level of $USER_i$ and store it in the message slot $BUFF\_2_{i,j,full,msg}$, whose specifications are given in the following descriptions of OM. Subsequently, if $USER_j$ tries to obtain themsg, AEF_RECV_j will collect the related ADI and send it to ADF for control policy. Only when a request is permitted by ADF, AEF_RECV_j will return msg to $USER_j$. Moreover, AEF_RECV_j might adjust $USER_i$'s level in terms of AEF's requirements.

$$\begin{aligned}
& \text{AEF\_SEND}_i = \text{api}_i?(\text{send}, j, \text{msg}) \longrightarrow \text{if } i == j \text{ then AEF\_ITEM}_i \\
& \text{else enterS}.i \longrightarrow \text{qstate}_{S_i}.\text{getlabel}? (S_i, I_i, C_i, \text{exist}_i, s\text{-sat}_i) \\
& \longrightarrow \text{qstate}_{i,j}.\text{buf\_in}!(\text{msg}, S_i - C_i^\pm, I_i \cup C_i^\pm) \longrightarrow \text{leaveS}.i \longrightarrow \text{AEF\_ITEM}_i \\
& \text{AEF\_RCV}_i = \text{api}_i?(\text{recv}, j) \longrightarrow \text{if } i == j \text{ then api}_i!(\text{error}, \text{null}) \longrightarrow \text{AEF\_ITEM}_i \\
& \text{else lock}(i, j) \longrightarrow \text{qstate}_{S_i}.\text{getlabel}? (S_i, I_i, C_i, \text{exist}_i) \longrightarrow \text{AEF\_ITEM}_i \\
& \text{else lock}(i, j) \longrightarrow \text{qstate}_{S_i}.\text{getlabel}? (S_i, I_i, C_i, \text{exist}_i) \longrightarrow \text{qstate}_{j,i}.\text{buff\_test}?(\text{full}, S_j, I_j) \\
& \longrightarrow \text{qstate}_{S_j}.\text{getlabel}? (S'_j, I'_j, C_j, \text{exist}_j) \longrightarrow \text{qpolicy}_i!(i\text{recv}, S_i, I_i, C_i, \text{full}, S_j, I_j, \text{exist}_j, S'_j, I'_j) \\
& \longrightarrow \text{qpolicy}_i?(\text{policy}, S'_j, I'_j, \text{renew}) \longrightarrow \text{if policy} == \text{accept then qstate}_{j,i}.\text{buff\_out}?msg \\
& \longrightarrow \text{api}_i!(\text{ok}, \text{msg}) \longrightarrow \text{if renew then qstate}_{S_i}.\text{setlabel}!(S'_i, I'_i, C_i, 1) \\
& \longrightarrow \text{unlock}(i, j) \longrightarrow \text{AEF\_ITEM}_i \text{ else unlock}(i, j) \longrightarrow \text{AEF\_ITEM}_i \text{ else api}_i!(\text{error}, \text{empty}) \\
& \longrightarrow \text{if renew then qstate}_{S_i}.\text{setlabel}!(S'_i, I'_i, C_i, 1) \longrightarrow \text{unlock}(i, j) \longrightarrow \\
& \text{AEF\_ITEM}_i \text{ else unlock}(i, j) \longrightarrow \text{AEF\_ITEM}_i,
\end{aligned} \tag{23}$$

where $\text{lock}(i, j) = \text{enterS}.\min(i, j) \longrightarrow \text{enterS}.\max(i, j)$
and $\text{unlock}(i, j) = \text{leaveS}.\min(i, j) \longrightarrow \text{leaveS}.\max(i, j)$.

When USER_i has permission to access object j , AEF expresses the related object operation using the event in channel io , e.g.,

$$\text{AEF\_WRITE}_i = \dots \longrightarrow \text{if policy} == \text{accept then io}!(\text{writefile}, j, \text{msg}) \longrightarrow \dots \tag{24}$$

If USER_i requests creating a new subject, AEF will first request the new pid from OM and OM will select the new pid randomly from the set of available process identifiers. Then

AEF will adjust the levels of USER_i and $\text{USER}_{\text{newid}}$ and give off the born signal to activate the process $\text{USER}_{\text{newid}}$.

$$\begin{aligned}
& \text{AEF\_EXEC}_i = \text{api}_i?(\text{exec}, j, \text{msg}) \longrightarrow \\
& \text{enterO}.j \longrightarrow \text{enterS}.i \longrightarrow \text{qstate}_{S_i}.\text{getlabel}? (S_i, I_i, C_i, \text{exist}_i) \\
& \longrightarrow \text{qstate}_{O_j}.\text{getlabel}? (S_j, I_j, C_j, \text{exist}_j) \longrightarrow \text{qpolicy}_i!(i\text{exec}, S_i, I_i, C_i, S_j, I_j, C_j, \text{exist}_j) \\
& \longrightarrow \text{qpolicy}_i?(\text{policy}, S'_i, I'_i, S'_j, I'_j, C'_j, \text{renew}) \longrightarrow \text{if renew then} \\
& \text{EXEC\_RET}_{i,j,\text{error}} \text{ else qstate}_{S_i}.\text{setlabel}!(S'_i, I'_i, C_i, 1) \longrightarrow \text{if} \\
& \text{policy} == \text{accept then qstate}_{\text{PID}}.\text{getpid} \longrightarrow \text{qstate}_{\text{PID}}.\text{choose}?(\text{result.newid}) \longrightarrow \text{if} \\
& \text{result then qstate}_{S_{\text{newid}}}.\text{setlabel}? (S'_j, I'_j, C'_j, 1) \longrightarrow \text{born}_{\text{newid}}!\text{msg} \longrightarrow \text{EXEC\_RET}_{i,j,\text{ok}} \\
& \text{else EXEC\_RET}_{i,j,\text{error}} \text{ else EXEC\_RET}_{i,j,\text{error}}, \\
& \text{where EXEC\_RET}_{i,j,\text{result}} = \text{leaveS}.i \longrightarrow \text{leaveO}.j \longrightarrow \text{AEF\_ITEM}_i.
\end{aligned} \tag{25}$$

When a subject exits, AEF will clear its label information in OM and update the set of available subject identifiers. In addition, it clears its send buffer.

$$\begin{aligned}
& \text{AEF\_EXIT}_i = \text{api}_i? \text{exit} \longrightarrow \text{enterS}.i \longrightarrow \text{RM\_eEXIT}_{i,|P|} \text{RM\_eEXIT}_{\text{from,to}} \\
& = \text{if to} \\
& == 0 \text{ then} \longrightarrow \text{qstate}_{S_i}.\text{setlabel}!(, , , 0) \longrightarrow \text{qstate}_{\text{PID}}.\text{delpid}!i \longrightarrow \text{leaveS}.\text{from} \\
& \longrightarrow \text{RM\_ITEM}_{\text{from}} \text{ else } \text{buff}_{\text{from,to}}.\text{out}? \text{msg} \longrightarrow \text{RM\_eEXIT}_{\text{from,to-1}}.
\end{aligned} \tag{26}$$

The system calls for changing level are mainly used to update the label information in OM.

2.2.3. OM. The process OM manages the system states information, i.e., the labels and existence of all the subjects and objects, the set of available process identifiers, and the messages in process communication.

We have introduced its subprocesses one by one. OM_S_LIVE_{pid,s,i,c,exist} provides the query and modification service for the label and existence of all the subjects, where pid denotes the process id, s, i, and c, respectively, denote USER_{pid}'s secrecy level, integrity level, and capabilities, and exist denotes whether the subject exists.

$$\begin{aligned}
& \text{OM\_S\_LIVE}_{\text{pid},s,i,c,\text{exist}} = \text{qstate}_{S_{\text{pid}}}.\text{getlabel}!(s, i, c, \text{exist}) \longrightarrow \\
& \text{OM\_S\_LIVE}_{\text{pid},s,i,c,\text{exist}} | \text{qstate}_{S_{\text{pid}}}.\text{setlabel}?(s', i', c', \text{exist}') \longrightarrow \text{OM\_S\_LIVE}_{\text{pid},s',i',c',\text{exist}',s_{\text{sat}}}
\end{aligned} \tag{27}$$

OM_S is an interleaving of all the OM_S_LIVE_{pid,s,i,c,exist}. We assume that the USER₁ with $C_1 = DS \cup DI \times \{+, -\}$ is the only live user process at system startup. Therefore, we

can set up any initial state of the system, for any type of process and object can be created by USER₁.

$$\text{OM\_S} = (||| \text{pid}: P - \{1\} @ \text{OM\_S\_LIVE}_{\text{pid},\{\},\{\},\{\},0} ||| \text{OM\_S\_LIVE}_{1,\{\},\{\},DS \cup DI \times \{+,-\},1} \tag{28}$$

Similarly, OM_O_LIVE_{oid,s,i,c,exist,exe} provides the query and set service about the label and existence of all the objects, and OM_O is an interleaving of all the

OM_O_LIVE_{oid,s,i,c,exist}. The “capability” attributes of the object are set by authority administrator, and we model it as a random event in the channel nonce.SIC.

$$\begin{aligned}
& \text{OM\_O\_LIVE}_{\text{oid},s,i,c,\text{exist}} = \text{qstate}_{O_{\text{oid}}}.\text{getlabel}!(s, i, c, \text{exist}) \longrightarrow \\
& \text{OM\_O\_LIVE}_{\text{oid},s,i,c,\text{exist}} | \text{qstate}_{O_{\text{oid}}}.\text{setlabel}?(s', i', c', \text{exist}') \longrightarrow \text{OM\_O\_LIVE}_{\text{oid},s',i',c',\text{exist}'} \\
& \text{OM\_O} = ||| \text{oid}: O @ \text{nonce.SIC}?(s, i, c, \text{exist}) \longrightarrow \\
& \text{OM\_O\_LIVE}_{\text{oid},s,i,c,\text{exist}}.
\end{aligned} \tag{29}$$

OM_ID_MGR provides the management of available process identifiers via channel qstate_{PID}, which allocates a new id when a new process is created and reclaims the id

when a process exits. In our specifications, access to channel qstate_{PID} is mutually exclusive.

$$\begin{aligned}
& \text{OM\_ID\_MGR}_{\text{occupy},\text{free}} = \text{qstate}_{\text{PID}}.\text{getpid}i \longrightarrow \text{if empty}(\text{free}) \\
& \text{then } \text{qstate}_{\text{PID}}.\text{choose}!(1, 0) \longrightarrow \text{OM\_ID\_MGR}_{\text{occupy},\text{free}} \text{ else nonce.} \\
& \text{Pid}?i: \text{free} \longrightarrow \text{qstate}_{\text{PID}}.\text{choose}!(0, i) \longrightarrow \text{OM\_ID\_MGR}_{\text{occupy} \cup \{i\}, \text{free} - \{i\}} | \text{qstate}_{\text{PID}}.\text{delpid}?i \longrightarrow \\
& \text{OM\_ID\_MGR}_{\text{occupy} - \{i\}, \text{free} \cup \{i\}}.
\end{aligned} \tag{30}$$

BUFF describes the behaviors of the message buffer in process communication, which caches and forwards the message. For simplicity, a message slot is set for each pair of sender and receiver.

The process OM_BUFF specifies the behavior of the process communication buffers.

$$\begin{aligned} \text{OM\_BUFF} &= ||| \text{from: } P@BUFF\_1_{\text{from}}, \\ \text{BUFF\_1}_{\text{from}} &= (||| \text{to: } P@BUFF\_2_{\text{from,to,0,msg,\{ \}}, \end{aligned} \quad (31)$$

$$\begin{aligned} \text{BUFF\_2}_{\text{from,to,full,msg,S,I}} &= q\text{state}_{\text{from,to}}.\text{buff\_in?}(\text{msg}', S', I') \longrightarrow \\ \text{BUFF\_2}_{\text{from,to,1,msg}',S',I'} &| q\text{state}_{\text{from,to}}.\text{buff\_out!msg} \longrightarrow \\ \text{BUFF\_2}_{\text{from,to,0,null,\{ \}} &| q\text{state}_{\text{from,to}}.\text{buff\_test!}(\text{full}, S, I) \longrightarrow \text{BUFF\_2}_{\text{from,to,full,msg,S,I}}. \end{aligned} \quad (32)$$

Finally, the OM is an interleaving of these service subprocesses:

$$\text{OM} = \text{OM\_S} ||| \text{OM\_O} ||| \text{OM\_ID\_MGR}_{\{1\},P-\{1\}} ||| \text{OM\_BUFF}. \quad (33)$$

The message slot provides the storage and removal of information. We specify simply that the new message will cover the old message that has not been taken away when the message buffer is full. The stored information also includes the current level of the sender. Its transition is as follows:

2.2.4. *ADF*. ADF also concurrently deals with the request of each of the subprocesses of AEF. According to the ADI and the rules, ADF returns its decision and requirements for state transition. Its behaviors are specified as follows:

$$\begin{aligned} \text{ADF} &= (||| i: P@ADF\_ITEM_i) ||| \text{ADF\_SAT}_{R,\text{pid}} \\ \text{where } \text{ADF\_ITEM}_i &= \text{ADF\_RECV}_i | \text{ADF\_WRITE}_i | \text{ADF\_READ}_i | \dots \end{aligned} \quad (34)$$

The process communication is controlled when the receiver tries to receive the message. For the request of api_i . (recv, j), SS_RECV_i will determine whether to permit it and develop the policy of adjusting the USER_i 's level in

terms of Definition 3. Since the system will clear the corresponding buffer when any subject exits, the message existing in the buffer must be fresh to guarantee the data consistency.

$$\begin{aligned} \text{ADF\_RECV}_i &= q\text{policy}_i?(\text{irecv}, S_i, I_i, C_i, \text{full}, S_j, I_j, \text{exist}_j, S'_j, I'_j) \longrightarrow \\ \text{constratin}_{\text{SSAT}_i} \text{query!}(S_i, C_i) &\longrightarrow \text{constrain}_{\text{SSAT}_{\text{pid}}} \text{query?s\_sat}_i \longrightarrow \text{if s\_sat}_i \text{ then} \\ q\text{policy}_i!(\text{reject}, \{ \}, \{ \}, 0) &\longrightarrow \text{ADF\_ITEM}_i \text{ else} \\ \text{constratin}_{\text{SAT}_i} \text{query!}(S_i \cup S_j, C_i) &\longrightarrow \text{constrain}_{\text{SSAT}_{\text{pid}}} \text{query?sat}_i \longrightarrow \\ \text{if exist}_j \wedge S_j \subseteq S_i \cup C_i^+ \wedge I_j \subseteq I_i - C_i^- \wedge \text{sat}_i \wedge \text{full} &\text{ then } q\text{policy}_i! \\ (\text{accept}, S_i \cup S_j, I_i - (I_i - I_j), 1) &\longrightarrow \text{ADF\_ITEM}_i \text{ else if exist}_j \wedge \\ (\text{full} \wedge S'_j \subseteq S_i \cup C_i^+ \wedge I'_j \subseteq I_i - C_i^- \wedge \text{sat}_i) &\text{ then} \\ q\text{policy}_i!(\text{reject}, S_i \cup S_j, I_i - (I_i - I_j), 1) &\longrightarrow \text{ADF\_ITEM}_i \text{ else} \\ q\text{policy}_i!(\text{reject}, S_i \cup C_i^+ \cap \text{DS} - [R], I_i - C_i^-, 1) &\longrightarrow \text{ADF\_ITEM}_i. \end{aligned} \quad (35)$$

The decision process for a request to read/write/create/delete/execute an object is similar to ADF_RECV_i mentioned above. The "capability" attributes of new object are set

by authority administrator, and we model it as a random event in the channel nonce.C. We omit the unnecessary details.

$$\begin{aligned} \text{ADF\_CREATE}_i = qstate_i?(icreate, j, S_j, I_j) \longrightarrow \dots \longrightarrow \\ \text{nonce}C?C_j \longrightarrow \dots qstate_i!(S_j, I_j, C_j). \end{aligned} \quad (36)$$

$\text{ADF\_SAT}_{R,\text{pid}}$ stores the mutually exclusive tag constraints R and provides the service to verify whether a given label is sat or s_satR .

$$\begin{aligned} \text{ADF\_SAT}_{R,\text{pid}} = \text{constrain}_{\text{SAT pid} \cdot \text{query}}?(s, i, c) \longrightarrow \text{SAT}_{R,\text{pid},s,i,c} | \text{constrain}_{\text{SSAT pid} \cdot \text{query}}?(s, c) \longrightarrow \text{SSAT}_{R,\text{pid},s,c} \text{SAT}_{\text{remain},\text{pid},s,i,c} \\ = \text{if empty}(\text{remain}) \text{ then } \text{constrain}_{\text{SAT pid} \cdot \text{query}}!\text{true} \longrightarrow \text{ADF\_SAT}_{R,\text{pid}} \text{ else if } |\text{head}(\text{remain}) \cap s| \\ > 1 \vee |\text{head}(\text{remain}) \cap i| > 1 \text{ then } \text{constrain}_{\text{SAT pid} \cdot \text{query}}!\text{false} \longrightarrow \text{ADF\_SAT}_{R,\text{pid}} \text{ else } \text{SAT}_{\text{tail}(\text{remain}),\text{pid},s,i,c} \text{SSAT}_{\text{remain},\text{pid},s,c} \\ = \text{if empty}(\text{remain}) \text{ then } \text{constrain}_{\text{SSAT pid} \cdot \text{query}}!\text{true} \\ \longrightarrow \text{ADF\_SAT}_{R,\text{pid}} \text{ else if } (|\text{head}(\text{remain}) \cap c^+ \cap DS| > 1 \wedge |\text{head}(\text{remain}) \cap s| = 0) \text{ then } \text{constrain}_{\text{SSAT pid} \cdot \text{query}}!\text{false} \\ \longrightarrow \text{ADF\_SAT}_{R,\text{pid}} \text{ else } \text{SSAT}_{\text{tail}(\text{remain}),\text{pid},s,c}. \end{aligned} \quad (37)$$

2.2.5. Abstract EILCM System. Based on the components above, the security kernel can be described as the concurrent implementation of AEF, OM, and ADF.

$$\text{KERNEL} = (\text{AEF} \parallel \text{OM}) \parallel \text{ADF}. \quad (38)$$

$qstate \quad qpolicy$

The overall system is the concurrent implementation of security kernel and user processes.

$$\text{SYSTEM}_{\text{EILCM}} = \frac{(\text{USER} \parallel \text{KERNEL})}{\text{api}}, \quad (39)$$

$a\text{KERNEL}$

where $a\text{KERNEL} = \{qstate, qpolicy, nonce, io, a\text{SYN}, \dots\}$.

We use the hide operator in $\text{SYSTEM}_{\text{EILCM}}$ to hide the inner transitions, e.g., the communication between AEF and ADF. In fact, the inner implementation of the system is a black box to user, and the user can at most see other user's behaviors, other than the inner traces [18].

2.3. Application of EILCM. The application case is a desktop system, as shown in Figure 2. Typically, the system includes an instant messaging tool *IM*, a map navigation software *MN*, a data encryption and transmission software *EMail*, an antivirus tool *Killer*, a web browser *UC*, and a Shell *Explorer*.

The user's chat information is stored in *IM* data. The daily operations of a user include communicating with contacts via *IM*, sending encrypted important *IM* files or *MN* data to his/her colleagues via *EMail*, and assisting driving via *MN*. The important configuration information of this system is stored in OS config file. *Killer* can detect and kill viruses, but it needs to update its own virus database online regularly. The security requirements of this system are as follows:

- (1) *IM* and *MN* are untrusted. They are forbidden to read each other's data.
- (2) *Killer* is allowed to read *IM* data, *MN* data, and OS config file in order to detect and kill potential viruses in time. However, it is forbidden to disclose these private data when updating the virus database.
- (3) Secret *IM* data and *MN* data can be encrypted and sent out by *EMail*. Since a program can make an

aggregation analysis of *IM* data and *MN* data and thus obtain more privacy information of users, the system requires privilege constraint about simultaneous reading of *IM* data and *MN* data.

- (4) *Explorer* can no longer modify OS config files after it loads the insecure libraries that are downloaded from network by *UC*.
- (5) The updated OS packages that are downloaded from network by *UC* cannot modify the configuration files unless they have been already checked by *Killer*.
- (6) *IM* sets the label of *IM* data. *MN* sets the label of *MN* data.

The label and capability scheme for building EILCM is as follows. In order to protect the secrecy of *IM* data and *MN* data, secrecy tags ds_{IM} and ds_{MN} are created, respectively, to track and control the propagation flow of these secret information. In order to protect the important data integrity of the operating system, an integrity label di_{OS} is established. The network is considered to have the lowest integrity level and secrecy level. Establish secrecy/least privilege rule: $\{ds_{MN}, ds_{IM}\} \in R$. The initial settings of each label are shown in Table 1.

In Table 1, since *IM* process is the owner of *IM* data, and its ability is set to $\{ds_{IM}^{\pm}\}$, indicating that it can manage these data. Similarly, the ability of *MN* process is set to $\{ds_{MN}^{\pm}\}$. The ability of *EMail* process is set to $\{ds_{IM}^{\pm}, ds_{MN}^{\pm}\}$, so that it can read *IM* and *MN* data and encrypt them before sending. The ability of *Killer* is set to $\{ds_{IM}^+, ds_{MN}^+, di_{OS}^{\pm}\}$, which means that it can read any secret information, but it does not have the ability to release secret information to network. *Killer* can lower the integrity level to scan any entity or add di_{OS} tags to itself or the data that are considered safe after scanning. The ability of *UC* process and *Explorer* process is set to $\{di_{OS}^-\}$. They are not given the ability to add integrity tag di_{OS} , so they cannot destroy the integrity of system data even if they are infected. It can be seen that EILCM system can better meet these security requirements. At the same time, because it belongs to the taint propagation model and supports the implicit change of tags, the system can reasonably automatically adjust the subject and object tags

FIGURE 2: A typical application scenario of a desktop operating system.

TABLE 1: Setting of initial labels and capabilities of subject and object by EILCM system.

Subjects	Secrecy label	Integrity label	Capabilities	Objects	Secrecy label	Integrity label
<i>IM</i>	{}	$\{di_{IM}\}$	$\{ds_{IM}^{\pm}\}$	<i>IM data</i>	$\{ds_{IM}\}$	{}
<i>MN</i>	{}	{}	$\{ds_{MN}^{\pm}\}$	<i>MN data</i>	$\{ds_{MN}\}$	{}
<i>Email</i>	{}	{}	$\{ds_{IM}^{\pm}, ds_{MN}^{\pm}\}$	<i>ETP data</i>	{}	{}
<i>Killer</i>	{}	{}	$\{ds_{IM}^+, ds_{MN}^+, di_{os}^{\pm}\}$	<i>OS config files</i>	{}	$\{di_{os}\}$
<i>UC</i>	{}	{}	$\{di_{os}^-\}$	<i>Download data</i>	{}	{}
<i>Explorer</i>	{}	{}	$\{di_{os}^-\}$	<i>OS Updated packages</i>	{}	$\{di_{os}\}$

according to the access requirements. In addition, it also supports the explicit change of labels, which makes the system more flexible and improves the availability.

3. Noninterference Analysis for EILCM

We will analyze the security properties of $SYSTEM_{EILCM}$ in this section. Goguen and Meseguer [27] took an important step towards a formal specification and analysis of information flow with the concept of noninterference. Intuitively, user *A* not interfering with user *B* means that that *A*'s behavior has no effect on what *B* can observe, that is, *B*'s view of the system is independent of *A*'s behavior. It is usually assumed that a system satisfying noninterference security has no covert channels.

Many variants of Goguen and Meseguer's definition [27] of noninterference were proposed with finite-state automaton [28], relational Hoare-type theory [29], Petri nets [30], game theory [26], etc. Ryan and Schneider [26] redefined the noninterference concept using CSP verification model [31] based on process equivalence. Bossi et al. [32] and Krohn and Tromer [18] partitioned the space of symbols into three sets: high, low, and declassify, and we also adopt similar idea. Krohn and Tromer and Bossi et al. [18, 32] presented the formula of noninterference with declassification in CSP

property-oriented specifications, which lacks the effective support of CSP tools for automated analysis when compared with standard CSP process-oriented specification [31]. In this section, we first use the definition of noninterference from [26] to formally detect the known covert channels and analyze the reasons why EILCM can close them; secondly, we propose our definition of noninterference with declassification and prove that the abstract $SYSTEM_{EILCM}$ fulfills the noninterference.

3.1. Analysis of Covert Channel in EILCM System. We used Ryan and Schneider's definition [26] of noninterference to detect the known cover channels above. We called their definition basic noninterference. In this section, we used the concept of CSP model and equivalence [31].

Definition 5. (basic noninterference). Assume that high-level users interact with the system SYS via the high events (*aHIGH*) and low-level users do so via the low events (*aLOW*). Further assume that these two types of events partition the full range of events (alphabet) of SYS . Then, the high-level user is said to be noninterfering with the low-level user via S if and only if

$$\forall tr, tr' \in \text{traces}(SYS), tr \upharpoonright aLOW = tr' \upharpoonright aLOW \Rightarrow \left(\frac{(S/tr)}{aLOW} \stackrel{\text{failure}}{=} \frac{(S/tr')}{aLOW} \right) \quad (40)$$

That is, after being advanced by tr and tr' , the two processes must accept all of the same traces (projected to low) and refuse all of the same refusal sets (projected to low) [18]. The definition above is a property-oriented

specification. In order to avoid the need to explicitly quantify over traces, Ryan and Schneider expressed a characterization directly as a standard CSP equivalence:

$$\frac{(\text{SYS} \parallel \text{STOP})}{a_{\text{HIGH}}} =_{\text{failure}} \frac{\text{SYS}}{a_{\text{HIGH}}} \quad (41)$$

Nevertheless, equation (40) is strictly stronger than equation (41) [26] because the latter does not consider the divergences in terms of semantics of the stable failure model. An alternative is not to hide the high events but to camouflage them [26]:

$$(\text{SYS} \parallel \text{STOP}) \parallel \text{RUN}_{a_{\text{HIGH}}} =_{\text{failure}} \text{SYS} \parallel \text{RUN}_{a_{\text{HIGH}}} \quad (42)$$

Obviously, this will greatly increase the state space for verification. In our view, when selecting the equivalence semantics from traces, failures, and divergences, we should consider safety, efficiency, and the characteristics of the actual verified system simultaneously. In general, based on the restriction of these models, we order these models from weak to strong and the complexity from low to high as follows: trace model, stable failure model, and failures/divergences. The trace model checks all the possible traces of process. This is appropriate for verifying safety properties [31], which are generally of the form “something bad will not happen.” However, it cannot verify liveness properties, generally in the form of “something good will happen.” This

is because the internal and external choice operators have the same trace semantics. The stable failure model solves the problem through increasing the failure check, but it does not capture the divergence behaviors. In general, adding the hide operator to a process expression might result in process divergence behaviors for the process. The failure/divergence model will check both failure and divergences, which are usually taken as the standard model. Of course, it can add some considerable overhead to the state space exploration. We think that if not for the existing divergence behavior of the actual system SYS but for the virtual system $\text{SYS}/a_{\text{HIGH}}$, a divergence attack does not exist and thus we need not require the equivalence of divergences.

According to Definition 5, for each event, we need to know the current levels of its sponsors and its properties in order to analyze the noninterference of the system. While the levels of all subjects and objects are stored in OM, the events themselves have no state information. Therefore, we explicitly label each system call $\text{api}_i(\dots)$ with the caller's current levels and capability. Now these system calls $\text{api}_i(\dots)$ have the form $\text{api}_{i,S,I,C}(\dots)$, where S , I , and C denote, respectively, its secrecy level, integrity level, and the caller's capability. The user process USER_i gets its label once born:

$$\begin{aligned} \text{USER}_i &= \text{born}_i?(S, I, C, \text{msg}) \longrightarrow \text{USER\_LIVE}_{i,S,I,C}, \\ \text{USER\_LIVE}_{i,S,I,C} &= \text{SEND}_{i,S,I,C} | \text{RECV}_{i,S,I,C} | \text{WRITE}_{i,S,I,C} | \text{READ}_{i,S,I,C} | \text{CREATE}_{i,S,I,O} | \text{DEL}_{i,S,I,O} | \dots \end{aligned} \quad (43)$$

In an EILCM system, the system calls that might cause the changing of caller's levels include $\text{api}_{i,S,I,C}(\text{recv}, \dots)$, $\text{api}_{i,S,I,C}(\text{read}, \dots)$, $\text{api}_{i,S,I,C}(\text{changelabel}, \dots)$, and $\text{api}_{i,S,I,C}(\text{exec}, \dots)$. Our implicit labeling for the system calls should reflect these changes in time. We introduce the

virtual system call $\text{api}_{i,S,I,C}(\text{getlabel}, \dots)$ and its kernel response to know the latest state of caller. The involved user subprocesses include $\text{RECV}_{i,S,I,C}$, $\text{READ}_{i,S,I,C}$, CHANGELABEL_i , and $\text{EXEC}_{i,S,I,C}$. We take $\text{RECV}_{i,S,I,C}$ as an example.

$$\begin{aligned} \text{RECV}_{i,S,I,C} &= \text{api}_{i,S,I,C}!(\text{recv}, \text{from}) \longrightarrow \text{api}_{i,S,I,C}?(\text{getlabel}, S', I') \\ &\longrightarrow \text{api}_{i,S',I',C'}?(\text{result}, \text{msg}) \longrightarrow \text{USER\_LIVE}_{i,S',I',C'} \end{aligned} \quad (44)$$

Other subprocesses need not update their label state; we take $\text{SEND}_{i,S,I,C}$ as an example.

$$\text{SEND}_{i,S,I,C} = \text{api}_{i,S,I,C}!(\text{send}, \text{to}, \text{msg}) \dots \longrightarrow \text{USER\_LIVE}_{i,S,I,C} \quad (45)$$

The modified $\text{SYSTEM}_{\text{EILCM}}$ is notated $\text{SYS}_{\text{EILCM}}$. Evidently, the label query call cannot change the secure states and state transitions, and the ADI used for ADF's decision is still derived from OM. The explicit label for

system calls only aids in the verification, and the implementation of $\text{SYSTEM}_{\text{EILCM}}$ need not consider it. $\text{SYS}_{\text{EILCM}}$ is equivalent to $\text{SYSTEM}_{\text{EILCM}}$ in security properties.

We use Krohn and Tromer's definition [18] to classify the $\text{SYS}_{\text{EILCM}}$ user-level events in terms of their labels.

Definition 6 (high event/low event). Given a system $\text{SYS}_{\text{EILCM}}$, for each secrecy protection tag d , define the high and low alphabets as follows.

The high events are

$$a\text{HIGH}_d = \{\text{api}_{i,S,I,C}(\dots)|d \in S, i \in P\} \cup \{\text{born}_i.(S, I, C \dots)|d \in S, i \in P\}. \quad (46)$$

The low events are the complement sets:

$$a\text{LOW}_d = \{\text{api}_{i,S,I,C}(\dots)|d \notin S, i \in P\} \cup \{\text{born}_i.(S, I, C \dots)|d \notin S, i \in P\}. \quad (47)$$

Next, we can analyze the known covert channels mentioned in Section 1 by virtue of Definitions 5 and 6.

Example 3. Detecting the known covert channels: let the attack instance mentioned in Section 1 run in SYS based on our EILCM specifications. Firstly, assume that the ADF implements the information flow rules similar to Asbestos's, rather than rules of EILCM. For secrecy protection, a contaminated process A is a high process and the other uncontaminated processes are low processes.

Let SYS' denote $\text{SYS} \parallel \text{STOP}_{a\text{HIGH}}$. Assume B_i is not contaminated in its initial state. If it is permitted for a contaminated process to change the labels of a less contaminated process implicitly, consider the trace $\langle \text{send}, C, m_1 \rangle, \text{api}_{B_i, \text{LOW}}(\text{recv}, A, \text{msg}), \text{api}_{C, \text{LOW}}(\text{recv}, B_i, \text{msg}) \rangle$, and the trace $\langle \text{api}_{A, \text{HIGH}}(\text{send}, B_i, m_2) \rangle^{\hat{s}}$, where each event is labeled with the sponsor's secrecy level. It is easy to conclude that $s \in \text{traces}(\text{SYS}')$, $t \in \text{traces}(\text{SYS})$, and $s \uparrow a\text{LOW} = t \uparrow a\text{LOW}$. When the event $\text{api}_{C, \text{LOW}}(\text{recv}, B_i, \text{msg})$ is triggered, we have $\hat{s} \langle \text{api}_{C, \text{LOW}}(\text{ok}, \text{msg}) \rangle \in \text{trace}(\text{SYS}'/a\text{HIGH})$ and $\hat{t} \langle \text{api}_{C, \text{LOW}}(\text{ok}, \text{msg}) \rangle \notin \text{trace}(\text{SYS}/a\text{HIGH})$, which fails to satisfy equation (41), that is, the HIGH event $\text{api}_A(\text{send}, B_i, \text{msg})$ interferes with the low process. Thus, we find a covert channel.

Secondly, assume that the ADF implements the rules of EILCM and continues to analyze the attack. When the event $\text{api}_{C, \text{LOW}}(\text{recv}, B_i, \text{msg})$ is triggered, we have $\hat{s} \langle \text{api}_{C, \text{LOW}}(\text{ok}, \text{msg}) \rangle \notin \text{traces}(\text{SYS}'/a\text{HIGH})$ and $\hat{t} \langle \text{api}_{C, \text{LOW}}(\text{ok}, \text{msg}) \rangle \notin \text{trace}(\text{SYS}/a\text{HIGH})$, which means that the trace will not occur in SYS and SYS' . Thus, we do not find a covert channel in EILCM system.

Thirdly, assume ADF denies implicit label change but permits explicit label change. Since the event $u = \text{api}_{B_i, \text{LOW}}(\text{changelabel}, \text{self}, \text{high})$ does not exist in trace s and t , B_i will not be raised to the high level, then both $\text{trace}(\text{SYS})$ and $\text{trace}(\text{SYS}')$ do not involve the trace $\hat{s} \langle \text{api}_{C, \text{LOW}}(\text{ok}, \text{msg}) \rangle$; in addition, if $s' = \langle u \rangle^{\hat{s}}$ and $t' = \langle u \rangle^{\hat{t}}$, we have $s' \uparrow a\text{LOW} = t' \uparrow a\text{LOW} = \langle u \rangle^{\hat{u}} \langle \text{api}_{C, \text{LOW}}(\text{recv}, B_i, \text{msg}) \rangle$, which make C unable to receive

the message from B_i , that is, both refusal $((\text{SYS}'/s')/a\text{HIGH})$ and refusal $((\text{SYS}/t')/a\text{HIGH})$ involve $\{\text{api}_{C, \text{LOW}}(\text{ok}, \text{msg})\}$. Therefore, there is no difference in this case and we do not find a covert channel in the situation of explicit label change.

3.2. Proof of Noninterference Property of EILCM.

Definition 5 strictly prohibits the high events from interfering with the low processes, but this might not reflect the actual secure requirement, e.g., the conditional downgrading. Many information flow models permit conditional information flows from high processes to low processes. For example, these flows are required to be carried out through trusted agents. EILCM permits these flows traversing the appropriate declassifiers. For a secrecy tag d , the declassifiers are the processes owning the ability to delete d from its secrecy level itself. Nevertheless, declassification makes the permitted information flow in EILCM system violate basic noninterference. For instance, consider a process p with $S_p = \{d\}$, $C_p = \{d^+, d^-\}$, a process q with $S_q = \{d\}$, $C_q = \{d^-\}$, and a process r with $S_r = \{d\}$, $C_r = \{d^-\}$. Comparing the case where p has called $\text{api}_{p, \text{HIGH}}(\text{send}, r, m_1)$ with the case where p has not, we find that if it calls $\text{api}_{r, \text{LOW}}(\text{recv}, p, \text{msg})$, the process r will obtain the information m_1 in the former case and will not obtain it in the latter case; similarly, comparing the case where q has triggered the event sequence $\langle \text{api}_{q, \text{HIGH}}(\text{send}, r, m_1), \text{api}_{q, \text{HIGH}}(\text{changelabel}, \text{self}, 0, I) \rangle$ with the case where q has not, we find that if it calls $\text{api}_{r, \text{LOW}}(\text{recv}, q, \text{msg})$, the process r will obtain the information m_1 in the former case and will not obtain it in the latter case. Therefore, we need a reasonable explanation and definition of noninterference for these cases. The new definition should permit the declassifier and low processes to interfere with any other processes and also permit the high processes to interfere with the other high processes but prohibit the high processes interfering with the low processes. This is called intransitive noninterference [33] or noninterference with declassification [18, 29, 30, 34].

Some works analyzed formally Android programs [35], Java programs [36], and microkernel [37] with noninterference theory, while their analyses are not automated. Egger et al. [38] proved that the verification of intransitive noninterference can

be solved in polynomial time. We will use a model checker tool FDR to make an automated analysis.

Before we analyze the $\text{SYS}_{\text{EILCM}}$'s property of non-interference with declassification, we need to specify the declassifying event, also called the mid event.

$$a\text{MID}_d = \{\text{api}_{i,S,I,C} \cdot (\dots) \mid d \in C^-, i \in P\} \cup \{\text{born}_{i^*}(S, I, C \dots) \mid d \in C^-, i \in P\}. \quad (48)$$

Definition 7 permits the intersection between $a\text{MID}$ and $a\text{HIGH}$ (or $a\text{LOW}$) not to be empty. In fact, $a\text{MID}$ includes all the events the declassifiers can trigger whether they are in high state or in low state. The $a\text{MID}$ might include events irrelevant to declassification, e.g., the declassifier sending a message to a high process or receiving a message from a low process. It is easy to conclude that these events neither change the current security state of the system nor interfere with the low process.

Now, we propose our definition of noninterference with declassification based on CSP equivalence.

Definition 8 (noninterference with declassification). Given a system $\text{SYS}_{\text{EILCM}}$, we say that $\text{SYS}_{\text{EILCM}}$ is secure for noninterference with declassification if and only if for each secrecy protection tag d , the following equation holds:

$$\frac{(\text{SYS}_{\text{EILCM}} \parallel \text{STOP})}{a\text{HIGH}_d - a\text{MID}_d} \equiv \frac{\text{SYS}_{\text{EILCM}}}{(a\text{HIGH}_d \cup a\text{MID}_d)}, \quad (49)$$

where $a\text{SYS} = a\text{HIGH}_d \cup a\text{LOW}_d$, $a\text{HIGH}_d \cap a\text{LOW}_d = \varphi$, $a\text{MID}_d \subseteq a\text{SYS}$, and $\equiv$ denotes an appropriate process equivalence.

Our definition of noninterference with declassification demands that the original system permitting all kinds of events to happen and the derived system prohibiting the events in $a\text{HIGH} - a\text{MID}$ to happen are equivalent if they all hide the events in $a\text{MID} \cup a\text{HIGH}$. That is, the set of events $a\text{MID}$ has the same influence on the low processes with $a\text{MID} \cup a\text{HIGH}$; put differently, the low processes cannot interfere with what the high processes do via the declassifiers; likewise, the high processes cannot interfere with the low processes via the declassifiers. Note that this extended definition is equivalent to basic noninterference if $a\text{MID} = \varphi$, where no declassifiers exist.

Next, we can review the declassification example mentioned at the beginning of Section 1 in light of Definitions 7 and 8. The events of $\text{api}_{i,q}(\text{changelabel}, \text{self}, 0, I)$, $\text{api}_p(\text{send}, r, m_1)$, and $\text{api}_q(\text{send}, r, m_1)$ are not only high but also mid. They can occur in $\text{SYS}' = \text{SYS} \parallel \text{STOP}_{a\text{HIGH} - a\text{MID}}$. Therefore, both $\text{SYS}/(a\text{HIGH} \cup a\text{MID})$ and $\text{SYS}/(a\text{HIGH} \cup a\text{MID})$ involve the trace $\langle \text{api}_{r,\text{LOW}}(\text{recv}, p, \text{msg}), \text{api}_{r,\text{LOW}}(\text{okerror}, \text{msg}) \rangle$, permitting p or q to interfere with r . That is, r can deduce the behaviors of the declassifiers p and q , but it cannot interfere with the behaviors of high processes.

The security property of noninterference with declassification requires that there is no explicit and implicit

Definition 7 (mid event). Given a system $\text{SYS}_{\text{EILCM}}$, for each secrecy protection tag d , define the mid alphabets as follows:

information flow from a domain to outside, excluding all the information flows produced by subjects with declassification abilities to write entities outside the domain. The property of noninterference with declassification is weaker than that of basic noninterference, since the latter requires that in any case there is no information flow from a domain to outside. Therefore, there are limitations of the security property of noninterference with declassification because it needs to assume that a subject with declassification ability of a certain domain will not leak the information of this domain. When it is untrusted and sends the information in the domain to the outside of the domain, the secret information is leaked, and the confidentiality is damaged.

Theorem 1. $\text{SYS}_{\text{EILCM}}$ is secure for noninterference with declassification.

Proof. According to Definition 8, we need prove that for each secrecy protection tag d , the claim $(\text{SYS}_{\text{EILCM}} \parallel \text{STOP}_{a\text{HIGH}_d - a\text{MID}_d}) / (a\text{HIGH}_d \cup a\text{MID}_d) \equiv \text{SYS}_{\text{EILCM}} / (a\text{HIGH}_d \cup a\text{MID}_d)$ holds. Since the internal operator is not used in the specifications of $\text{SYS}_{\text{EILCM}}$, the trace model and the stable failures model have the same equivalence semantics; in addition, the secure kernel $\text{KERNEL} = (\text{AEF} \parallel \text{OM}_{q\text{state}}) \parallel \text{ADF}_{q\text{policy}}$ just triggers the related inner event for the user's system call. There are no inner loop processes; moreover, the priority of locking is set to prevent deadlock in terms of conventional methods. Therefore, KERNEL will not consecutively and infinitely execute the inner event, that is, KERNEL will not be divergent. Taking all the factors above into account, we use the trace model to verify process equivalence.

In $\text{SYS}_{\text{EILCM}}$, given a secrecy tag d , firstly consider $d \notin [R]$, that is, d is not mutually exclusive to any other tag. In this case, according to different combinations of a subject's levels and capability, the subjects have eight states with respect to protecting the secrecy domain d . These states are as follows: $(\{d\}, \{\})$, $(\{d\}, \{d^+\})$, $(\{d\}, \{d^-\})$, $(\{d\}, \{d^+, d^-\})$, $(\{\}, \{\})$, $(\{\}, \{d^+\})$, $(\{\}, \{d^-\})$, $(\{\}, \{d^+, d^-\})$, where each state is a 2-tuple (S, C) , with S representing the subject's secrecy level and C representing the subject's capability. During the system state transitions, consider two subjects A and B that have the same state at some moment. We have the following knowledge. Firstly, it is easy to know that A and B have the same influence on the other subjects and objects, that is, A can do anything B does, and vice versa. Secondly,

the influence of other parts of systems on A and B is also the same, that is, whether or not it is permitted that any other part's operation sequence on A is the same as that on B . Thirdly, since both of them are contaminated or uncontaminated subjects simultaneously, their own interaction will not change their level. Finally, the subject's capability remains unchangeable. On the other hand, as for the object's states, since the label of a new subject has a reference to the levels and "capability" attribute of the object from whom it is derived, the objects have eight states in this case. If two objects have the same state, any traces related to them can happen when they are replaced by each other.

In $\text{SYS}_{\text{EILCM}}$, let P be the set of all subject identifiers and let O be the set of all object identifiers. Assume $|P| = 8$ and $|O| = 8$. Assume that in the initial state of the system there is only a subject with its capabilities being $\{d^+, d^-\}$, so it can create any type of object and further any type of subject; therefore, during the system state transitions, it is legal that there are eight subjects with different subject states. Assuming the claim holds on the condition that $|P| = 8$ and $|O| = 8$, we analyze whether the claim holds when $|P| = 9$ and $|O| = 8$. If it does not hold on this condition, the interference must appear at some point after nine subjects start existing simultaneously and be relevant to the behavior of the ninth-born subject because the claim holds on condition that $|P| = 8$.

Meanwhile, the ninth-born subject's state must be the same as one of the top-eight-born subjects because all subjects have at most eight states. Nevertheless, according to the analysis in the previous paragraph, two subjects that have the same states have the same influence on systems, and the influence of other parts on them is also the same, and their own interaction does not change the system state. So, we get the following paradox: given two subjects with the same state, one subject causes the occurrence of interference, while the other does not.

Therefore, the assumption that the claim does not hold at $|P| = 9$ and $|O| = 8$ is wrong. So, if it holds on the condition that $|P| = 8$ and $|O| = 8$, then the claims must hold at $|P| = 9$ and $|O| = 8$. Moreover, we can deduce that if it holds on condition that $|P| = 9$ and $|O| = 8$, then the claim must hold on the condition that $|P| > 9$ and $|O| = 8$, and so on. We can use the same method to analyze the situation of $|P| = 8$, $|O| = 9$. If there are nine objects existing together, it must mean that two subjects among them must have same states, which means that any other part's operation sequence on them has the same result, i.e., permission or rejection. So, if it holds at $|P| = 8$ and $|O| = 8$, then the claim must hold on condition that $|P| = 8$ and $|O| = 9$, and so on.

Therefore, if we can prove that the claim holds on the condition that $|p| = 8$, $|O| = 8$, $R = \varnothing$ and there is a subject with its capabilities $\{d^+, d^-\}$ in the system initial state, inductively, we can prove that the claim holds on the condition that $|p| > 8$, $|O| > 8$, $R = \varnothing$, with the initial state of system being arbitrary. Thus, we prove that it holds on the condition that $|p| \geq 0$, $|O| \geq 0$, $R = \varnothing$, with the initial state of system being arbitrary. Actually, the eight states of the subjects can be divided into five categories, and the states of each category are equivalent: (1) $(\{d\}, \{\}) (\{d\}, \{d^+\})$. (2) $(\{d\}, \{d^-\})$.

(3) $(\{\}, \{d^+, d^-\}) (\{d\}, \{d^+, d^-\})$. (4) $(\{\}, \{d^+\})$. (5) $(\{\}, \{\}) (\{\}, \{d^-\})$. For instance, each subject whose state is in class (3) can freely adjust its own level between $\{d\}$ and its behavior consisting of mid events entirely. So, we only need to analyze whether the claim holds on the condition that $|p| = 5$, $|O| = 5$, $R = \varnothing$ and there is a subject with its capabilities $\{d^+, d^-\}$.

We use FDR2 to verify the claim on the condition that $|p| = 5$, $|O| = 5$, $R = \varnothing$, and there is a subject with the capabilities $\{d^+, d^-\}$ in the system initial state. FDR is a model-checking tool for state machines, with foundations in the theory of concurrency-based CSP. It can establish whether a property holds for a refined transition system. FDR2 improves verification efficiency through compressing the states. It is easy to conclude that the trace collection of $(\text{SYS}_{\text{EILCM}} \parallel \text{STOP}_{a\text{HIGH}_d - a\text{MID}_d}) / (a\text{HIGH}_d \cup a\text{MID}_d)$ is a subset of $\text{SYS}_{\text{EILCM}} / (a\text{HIGH}_d \cup a\text{MID}_d)$.

We need to verify whether or not the latter is a subset of the former. The verification problem can be expressed in FDR2 using the following assertion:

$$\text{assert } \frac{(\text{SYS}_{\text{EILCM}} \parallel \text{STOP}_{a\text{HIGH}_d - a\text{MID}_d})}{(a\text{HIGH}_d \cup a\text{MID}_d)} [T \frac{\text{SYS}_{\text{EILCM}}}{(a\text{HIGH}_d \cup a\text{MID}_d)}]. \quad (50)$$

The previous section considered the case that $d \notin [R]$. As for the case that $d \in [R]$, we know that no subject has the ability to delete tag d from its own level in terms of our EILCM. In this case, the subjects have eight states with respect to protecting the secrecy domain d . Having been compressed, these states are as follows: $(\{\}, \{\})$, $(\{\}, \{d^+\})$, $(\{e^+\}, \{e^+\})$, $(\{\}, \{d^+, e^+\})$, $(\{d\}, \{\})$, $(\{e\}, \{\})$, $(\{d, e\}, \{\})$, where each state is a 2-tuple (S, C) , S represents the subject's secrecy level, C represents the subject's capability, and e represents some tag mutually exclusive to tag d . In this case, we use FDR2 to verify the claim on the condition that $|p| = 10$, $|O| = 1$, $R = \{d, e\}$, and there is a subject with its capabilities $(\{d^+, d^-, e^+, e^-\})$ in the system initial state.

The experiments were performed using FDR2.83 on Ubuntu 9.10 with an Intel P4 2.8 G CPU and 2 GB of memory. Both of the results of verification for the two cases above are true. The runtime verification time is 1680 seconds in the case of $d \notin [R]$ and 2580 seconds in the case of $d \in [R]$.

Taking into account all of the results of the analysis and automatic verification, we conclude that $\text{SYS}_{\text{EILCM}}$ is secure for noninterference with declassification.

Though our noninterference of $\text{SYS}_{\text{EILCM}}$ was deduced from secrecy protection, the same argument holds for integrity protection. For integrity tag d , the high events are the events with integrity level involving the tag d , the low events are the events with its integrity level not involving the tag d , the contaminated subjects are regarded as *high* subjects, and the uncontaminated subjects are regarded as *low* processes. We can use the same methods to prove that the contaminated subjects do not interfere with uncontaminated subjects.

The covert channel mentioned in the paper can also be found by using Definition 8 as a property. Our modeling of the system in the proof process has depicted the possible utilization behavior of the covert channel, and there are

subjects with different labels and capabilities in the initial state of the system, so whether this covert channel can occur has been verified by the FDR tool. The participating entities of this covert channel do not contain declassifiers, so Definitions 8 and 5 are equivalent for detecting this covert channel. In order to explain more clearly, the non-interference analysis of this covert channel is also analyzed by using basic noninterference in Definition 5. $\square$

4. Policy Verification for EILCM System

4.1. Problem Definition and Complexity. The purpose of noninterference analysis is to verify whether the EILCM system implements the information flow control according to the information flow policies and whether there are covert channels in which there are information flows that violate the security policy, including explicit and implicit information flows. If the system is secure without interference, it means that the implementation of the system strictly adheres to the security policy and there is no covert channels.

The purpose of policy verification is to verify the information flow reachability problem under a given tag and capability allocation scheme. The EILCM model is highly flexible, often with thousands of entities whose capabilities and tags are varied. Also, the EILCM model is dynamic, and its information flow accessibility is complicated. Then, the burden on administrators is large, which can easily lead to policy mismatches and mismatches. Automated policy verification can assist administrators in managing policies and thus is required.

In EILCM, although the subject's capabilities for deleting secrecy tags or adding integrity tags from its own levels have good semantics, these capabilities complicate the information flow analysis. We need to verify whether information can flow from one subject/object to another subject/object in the system state transitions. For instance, a subject A with $S_A = \{ds_1\}$, $I_A = \{di_1\}$, and $C_A =$ cannot send messages directly to a subject B with $S_B = \{ds_2\}$, $I_B = \{di_2\}$, and $C_A =$, but A can indirectly achieve its goals through forwarding a subject M with $S_M = \{ds_1\}$, $I_M =$, and $M_C = \{ds_1^-, di_2^+\}$.

In our model, the secrecy submodel is the dual of the integrity submodel. Their problems of policy analysis are equivalent and can be transformed into each other. A conversion method is as follows: given an integrity tag d , for each subject/object x , we use the notation $I_x^d: \langle a_1, a_2, a_3 \rangle$ denoting its integrity level on d , where $a_1, a_2, a_3 \in \{0, 1\}$, a_1 denotes whether x 's level involves d , a_2 denotes whether x has the capability of adding d to its own integrity level, and a_3 denotes whether x has the capability of deleting d from its own integrity level. We can give its dual form of secrecy level on d : $S_x^d: \langle \overline{a_1}, a_3, a_2 \rangle$. Similarly, if we let $I_y^d: \langle b_1, b_2, b_3 \rangle$ be y 's integrity level on d , then we have $S_y^d: \langle \overline{b_1}, b_3, b_2 \rangle$ as y 's secrecy level on d . By enumerating all possible cases, we can determine that the integrity constraint of I_x^d and I_y^d and the secrecy constraint of S_x^d and S_y^d have the same effect on communication. Therefore, in order to simplify the analysis,

we only consider the constraints of secrecy protection on the information flow.

Moreover, there are two aspects that can simplify the analysis. Firstly, it is easy to determine that the remaining part of a legal path after deleting all the object-type nodes is still a legal path because the object's capability is empty. For instance, if a path $p \rightarrow o \rightarrow q$ is legal, where p, q are subjects and o is an object, then $p \rightarrow q$ is legal. So, we need not consider objects being nodes in a legal path. Secondly, in terms of Definition 3, the new-born subject's capabilities are the subset of its parent subject's capabilities, while its secrecy/integrity levels are lower than its parent's. That is, any legal path remains legal if we replace some subject node with the subject's parent. So we need not consider generating subjects.

To facilitate its description, an EILCM system instance Sys is a 9-tuple $Sys = \langle U, O, DS, DI, S, I, C, CO, A \rangle$, where U denotes the set of all the users, O denotes the set of all the objects, DS denotes the set of all the secrecy(import) protection tags, DI denotes the set of integrity(import) protection tags. Let $V = U \cup O$; then, $S: V \rightarrow 2^{DS}$ maps the each subject/object onto secrecy levels, $I: V \rightarrow 2^{DI}$ maps the each subject/object onto integrity levels, $C: S \rightarrow 2^{(DS \cup DI) \times \{+, -\}}$ maps each subject/object onto capabilities, R is the set of mutually exclusive tags constraints, and $A: O \rightarrow 2^{(DS \cup DI) \times \{+, -\}}$ maps each object onto "capability" attributes. According to the analysis of the object's influence, our policy verifications ignore new process generation, any objects occurring in the path, and integrity protection. Therefore, Sys can be simplified as $\langle U, DS, S, C, CO \rangle$. Next, we can define the problem of policy verifications of EILCM systems.

Definition 9 (policy verification problem of EILCM systems). Given an EILCM system instance $Sys = \langle U, DS, S, C, CO \rangle$, for $\forall p, q \in U$, we say p can reach q if and only if there is a path $\langle v_1, v_2, \dots, v_n \rangle$, where $v_1 = p$, $v_n = q$, $v_i \in U$, $1 \leq i \leq n$, and the information can legally flow in terms of requirement of EILCM. Let $S_{initial}(v_i)$ be the initial secrecy level of v_i before the query. Let $S_{in}(v_i)$ be the secrecy levels of v_i just when v_i has received the information from v_{i-1} . Let $S_{out}(v_i)$ be the secrecy levels of v_i when v_i is sending the information to v_{i+1} . That is, the path needs to satisfy two constraints: (1) information flow constraints: $S_{out}(v_{i-1}) \subseteq S_{in}(v_i)$, $1 < i \leq n$; $S_{out}(v_i) \subseteq S_{in}(v_{i+1})$, $1 \leq i < n$; (2) level adjusting constraints: $S_{initial}(v_i) \subseteq S_{in}(v_i) \subseteq S_{initial}(v_i) \cup C^+(v_i)$, $S_{in}(v_i) \text{ sat } CO$, $S_{in}(v_i) - C^-(v_i) \subseteq S_{out}(v_i) \subseteq S_{in}(v_i)$.

The problem of reachability analysis is a decision problem. We use the notation $QEILCM(p, q, Sys)$ to represent the instance judging whether p can reach q in an EILCM system Sys . Next, we use the reduction method to analyze the complexity of this problem. The problem used to reduce from is the 3SAT problem defined below:

Definition 10 (monotone 3SAT). Given a set X of Boolean variables and a collection L of clauses over X , each $l \in L$ having $|l| = 3$ and containing either all positive variables or all negative variables, is there a satisfying truth assignment on X for L ? This problem is notated as $M3SAT(L, X)$. Monotone 3SAT is a special case of 3SAT.

Theorem 2. *The problem of policy verifications of EILCM system is NP-complete.*

Proof. We first reduce monotone 3SAT to the problem in polynomial time, that is, $\text{MSAT} \propto_p \text{QEILCM}$. Given a monotone 3SAT instance $M3\text{SAT}(L, X)$, assume $X = \{x_1, x_2, \dots, x_K\}$ and $L = a_1 \wedge a_2 \wedge \dots \wedge a_M \wedge b_1 \wedge b_2 \wedge \dots \wedge b_N$, where each clause $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ and each clause $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$. We produce the corresponding QEILCM instance $\text{QEILCM}(p, q, \text{Sys})$, $\text{Sys} = \langle U, \text{DS}, S, C, \text{CO} \rangle$ as follows.

Constructing the tags: for each clause a_i , $1 \leq i \leq M$, we introduce a secrecy protection tag d_i into S . For each clause b_j , $1 \leq j \leq N$, we introduce three secrecy protection tags d_j^1, d_j^2, d_j^3 .

Constructing the first part of subject: for each Boolean variable $x_i \in X$, $1 \leq i \leq K$, we introduce a subject u_i into U with its initial secrecy level $S(u_i) = \varnothing$ and its capabilities $C(u_i) = \{d_j^+ | 1 \leq j \in M\} \cup \{(d_j^w)^+ | 1 \leq j \leq N, 1 \leq w \leq 3\}$. Then, for each clause $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$, let $C(u_{i_1}) = C(u_{i_1}) \cup \{d_i^-\}$, $C(u_{i_2}) = C(u_{i_2}) \cup \{d_i^-\}$, $C(u_{i_3}) = C(u_{i_3}) \cup \{d_i^-\}$; for each clause $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$, let $S(u_{j_w}) = S(u_{j_w}) \cup \{d_j^w\}$, where $1 \leq w \leq 3$.

Constructing the second part of subject: for each clause $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$, $1 \leq j \leq N$, we introduce three subjects: u_j^1, u_j^2, u_j^3 into U , let each subject's secrecy level $S(u_j^w) = \varnothing$, and let each subject's capabilities $C(u_j^w) = \{(d_i^z)^+ | 1 \leq i \leq N, 1 \leq z \leq 3, i \neq j\} \cup \{(d_j^z)^+ | 1 \leq z \leq 3, z \neq w\}$.

Constructing p , q , and CO : let $S(p) = \{d_i | 1 \leq i \leq M\}$. Let $S(q) = \varnothing$. Let $\text{CO} = \varnothing$.

Clearly the construction process can be finished in polynomial time. Next, we prove that $M3\text{SAT}(L, X)$ is true if and only if $\text{QEILCM}(p, q, \text{Sys})$ is true.

Equivalence of the two problems: in problem $M3\text{SAT}(L, X)$, if there exist true/false assignments to the variables such that formula L is satisfiable, let set $X' = \{x_{s_1}, x_{s_2}, \dots, x_{s_w}\}$ be the variables assigned to true. Then, in problem $\text{QEILCM}(p, q, \text{Sys})$, we can prove that the path $\langle p, u_{s_1}, u_{s_2}, \dots, u_{s_w}, \text{lin3}(u_1^1, u_1^2, u_1^3), \text{lin3}(u_2^1, u_2^2, u_2^3), \dots, \text{lin3}(u_n^1, u_n^2, u_n^3), q \rangle$ is legal, where $\text{lin3}(u_j^1, u_j^2, u_j^3)$, $1 \leq j \leq n$, denotes choosing one subject as the current node from subjects u_j^1, u_j^2, u_j^3 that can legally forward the information from the previous node. Here, we can reasonably assume that each node x in the path will delete every tag it has the right to delete from its current level when it receives the information from previous node, that is, $S_{\text{out}}(x) = S_{\text{in}}(x) - C^-(x)$. Now, since each clause $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ is true, $1 \leq i \leq m$, X' includes at least one variant of $\{x_{i_1}, x_{i_2}, x_{i_3}\}$. Therefore, corresponding to each tag d_i in p 's levels, $1 \leq i \leq m$, there is at least one subject of $\{u_{s_1}, u_{s_2}, \dots, u_{s_w}\}$ that can delete d_i from its own

contaminated level when the information flow from p passes the path $\langle p, u_{s_1}, u_{s_2}, \dots, u_{s_w} \rangle$. And since $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$ is true, X' includes at most two variables of $x_{j_1}, x_{j_2}, x_{j_3}$. Correspondingly, for each group of tags d_j^1, d_j^2, d_j^3 , $1 \leq j \leq n$, after the information flow passes the path $\langle p, u_{s_1}, u_{s_2}, \dots, u_{s_w} \rangle$, $S_{\text{out}}(u_{s_w})$ includes at most two tags of this group. This means that the path $\langle \text{lin3}(u_1^1, u_1^2, u_1^3), \text{lin3}(u_2^1, u_2^2, u_2^3), \dots, \text{lin3}(u_n^1, u_n^2, u_n^3) \rangle$ can delete all the tags that are bound with the information level and contaminated by $u_{s_1}, u_{s_2}, \dots, u_{s_w}$. Therefore, if the formula L is satisfiable, p can reach q .

For the "only if" part, assume that there exists a legal path $\langle p, v_1, v_2, \dots, v_w, q \rangle$, where $v_i \in \{u_y | 1 \leq y \leq K\} \cup \{u_z^1 | 1 \leq z \leq N\}$, $1 \leq i \leq w$. We first simplify the path. Let the node v_r be the first one whose $S_{\text{out}}(v_r)$ does not include the tags d_i , $1 \leq i \leq M$. According to our transformation, for each v_j where $j \leq r$, we have $v_i \in \{u_y | 1 \leq y \leq K\}$, that is, another form of the path is $\langle p, u_{\lambda_1}, u_{\lambda_2}, \dots, u_{\lambda_r}, v_{r+1}, v_{r+2}, \dots, v_w, q \rangle$. Moreover, it is easy to determine that the path remains legal after deleting arbitrary node v_j where $j > r$ and $v_j \in \{u_z^1 | 1 \leq z \leq N\}$, and we use $\langle p, u_{\lambda_1}, u_{\lambda_2}, \dots, u_{\lambda_r}, u_{\gamma_{r+1}}^1, u_{\gamma_{r+2}}^1, \dots, u_{\gamma_w}^1, q \rangle$ to denote the remains of the path. From the path, we pick up the subjects that belong to $\{u_i | 1 \leq i \leq K\}$ to constitute set U' . Each variable in X that corresponds to each user in U' constitutes the set X' . If the variables in X' are assigned to true and the variables in $X - X'$ are assigned to false, we can prove L is true. In fact, for each tag d_i involved in p 's levels, there must be at least a subject v_z that has the ability to delete d_i from its own level. Correspondingly, for each $a_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$, X' involves at least one of $x_{i_1}, x_{i_2}, x_{i_3}$. Meanwhile, for each $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$, it is impossible that X' simultaneously involves $x_{j_1}, x_{j_2}, x_{j_3}$; otherwise, before the information that comes from p and flows down the path arrived $u_{\gamma_{r+1}}^1$, the information has been contaminated by d_j^1, d_j^2, d_j^3 simultaneously, that is, $d_j^1, d_j^2, d_j^3 \in S_{\text{out}}(u_{\lambda_y})$. However, according to our transformation, no subject among u_j^1, u_j^2, u_j^3 has the abilities to delete all the three tags simultaneously, and the path does not exist, which is contradictory to our assumption, so each $b_j = \overline{x_{j_1}} \vee \overline{x_{j_2}} \vee \overline{x_{j_3}}$ holds. Therefore, if p can reach q , then formula L is satisfiable. Now we can prove $\text{MSAT} \propto_p \text{QEILCM}$, that is, QEILCM is NP-hard.

Next, we show the QEILCM in NP. Given an instance $\text{QEILCM}(p, q, \text{Sys})$ with a solution in form of a path, the verification algorithm can get the answers in linear time $O(n)$, where n is the length of path. Now we prove that the problem of policy verifications of EILCM system is NP-complete.

Theorem 2 shows that in the worst case, the policy verification problem in EILCM system cannot be solved in polynomial time. It reflects that the EILCM policy with the rich expressive ability can provide management solutions for complex fine-grained information flow control. The high complexity of EILCM is attributed to the mechanisms of decentralized declassification and taint-tracking. According to this conclusion, the complexity of the verification algorithm is exponential, and a good verification algorithm should solve fewer

instances over long time periods while solving most EILCM instances in short time periods. With some search optimization strategies, the backtracking algorithm can guarantee a low computational overhead in most cases. $\square$

4.2. Algorithms for QEILCM. The backtracking algorithm searches the solution in the tree of candidate solutions, which starts from the root node and does not search the subtree that must not include the solution. The process of searching is systematic and jumping. In our problem, if p is a source node and q is a destination node, then p is regarded as the root node, the leaf nodes are the subjects who can send information to q , and the middle nodes are the subjects that can receive information from their parent nodes. Therefore, the path from root node to leaf node is an information flow path from p to q . The tree is an arrangement of subjects. In our algorithm, the variable *curpath* records the current searched path, and *pathlen* records the length of current searched path. The backtracking function searches all the possible paths' depth recursively.

We adopt two policies to decrease the search space. We firstly introduce a pruning function *prune*(*curpath*[*pathlen*]) to prune the subtree where there must not be solutions, and its root node is *curpath*[*pathlen*]. That is, if the node *curpath*[*pathlen*] has a tag in its levels that no subject on its extending subtree has the ability to delete, the function returns 0. The pruning function avoids the ineffective search. In order to decrease the computation amount, we introduce the integer array *allsub*, with each item in it corresponding to a tag and recording the sum of subjects that do not exist in current path and have abilities to delete the tag. Each recursion only does the incremental computation rather than the overall computation again. Secondly, we adjust the search order and search preferentially the subjects that have greater benefits and smaller expense, which can efficiently select the next node to extend. We define a subject i 's priority $PRI_i = taint_i / subtract_i$, where $taint_i$ is the sum of tags it has no ability to delete from its levels, that is, $taint_i = |S_i - C_i^-|$, and $subtract_i$ is the size of C_i^- . PRI considers the influence of the subject i 's contaminated situation (expenses) and its abilities of deleting tags (benefits). In the preprocess stage, function *SortbyPRI* arranges the search priority from the subjects having small priority to subjects having great priority. Algorithm 1 is as follows.

In our algorithm, function *canflow*(*subject1*, *subject2*) judges whether *subject1* can send information to *subject2*. Function *flowsuccess*(*subject1*, *subject2*, *curpath*[*pathlen* + 1]) judges whether *subject2*'s levels change violates the constraints *CO* if *subject1* can send information to *subject2*; moreover it will store the newly adjusted *subject1*'s levels into *curpath*[*pathlen* + 1] if subjects have sent information to *subject2*. Functions *addcapability*(*allsub*, *curpath*[*pathlen*]) and *removecapability*(*allsub*, *curpath*[*pathlen*]), respectively, add/subtract 1 to the value of each *allsub* item if the node *curpath*[*pathlen*] has the ability to delete the tag corresponding to the item. The algorithm's computational

complexity is $O(n!)$. Nevertheless, because of the requirement of partial relation between the sender's level and the receiver's level, in general, the algorithm can get the answer in a shorter time, which can be demonstrated by the following experiments.

In order to validate QEILCM algorithm, we first create a test data generator. It performs as follows: given n subjects and m tags, for each tag d and each subject p , whether p 's levels include d depends on the random choice based on $\{0, 1\}$, and p 's capabilities concerning d depend on the random choice based on $\{0, 1, \dots, 5\}$, in which 0, 1, 2, and others denote, respectively, the following capabilities: adding d , subtracting d , both adding and subtracting d , and no capabilities. We implement the QEILCM algorithm using Visual C++. Our experiments are run on Windows2003 with an Intel P4 2.8 G CPU and 2 GB of memory.

Firstly, we present our results about the algorithm's performance under different parameter values. These parameters are as follows: subject sum (n), tag sum (m), and the size of mutually exclusive set (k). Each group with constant parameter values repeats the test 20 times. The evaluation index includes average runtime (st) and average path length (l) in case that there exists a path, average runtime (ut) in case that there does not exist a path, successful times (sc), unsuccessful times (uc), and times (N/A) in case that all the tests do not work out the result in 30 minutes. The results are shown in Table 2.

Table 2 shows that although its complexity is exponential time, we can get the answers in most cases. If m and k are constants, the increase of runtime the algorithm takes will not be proportional to the increase of n at all, and even st has no distinct change, possibly because the scale of the arranging tree is not increased in proportion to the increase of subject sum under the constraints of partial relation of levels between the subjects. If the parameter k is constant, with the increase of m , the runtime will decrease remarkably because the subject's levels are generated randomly and the probability that a pair of subject's levels satisfying partial relations will decrease when m is bigger. If n and m are constants, with the increase of k , the success rate will appreciably decrease with the increase in k , while the runtime has no distinct change.

Secondly, we present our results about the functions *prune* and *SortbyPRI* influencing algorithm performance. Letting $k = 2$, for different combinations of n/m , we test eight times to compare the runtime in the case of using them with the runtime in the case of not using them. As shown in Figure 3, functions *prune* and *SortbyPRI* improve the search efficiency remarkably.

5. Related Work

Information flow models can effectively control information propagation to protect the secrecy and integrity of the operating system. The traditional one-directional information flow model is too rigid to use. Much work adopts the technique of taint-tracking to capture the more precise characterization of information flow. IX [39], an experimental MLS variant of Unix, has a "drop-on-exec" feature, in

Input: array $u[0..n-1]$: each item records a subject's levels and capabilities. n : sum of the subjects. co : set of mutually exclusive tags constraints.

Output: if u_0 can reach u_{n-1} , then print the path; else print "unreachable."

QEILCM (u)

```
{  $u = \text{SortbyPRI}(u)$ ;
   $\text{curpath}[0] = u[0]$ ;
  ifBacktrack ( $1, u, \text{curpath}, \text{pathlen}$ )
  then  $\text{output}(\text{curpath})$ ;
else  $\text{output}(\text{"cannot find path"})$ ; }
Backtrack ( $\text{depth}, u, \text{curpath}, \text{pathlen}$ )
{ if  $\text{depth} = n - 1$  return 0;
  if not  $\text{canflow}(\text{curpath}[\text{pathlen}], u[n - 1])$  return 1;
  ifprune ( $\text{curpath}[\text{pathlen}], \text{allsubcapability}$ ) return 0;
for  $i = \text{depth}, \dots, n - 2$ 
{ ifcanflow ( $\text{curpath}[\text{pathlen}], u[i]$ ) continue;
if not  $\text{flowsuccess}(\text{curpath}[\text{pathlen}], u[i], \text{curpath}[\text{pathlen} + 1])$ 
continue;
   $\text{pathlen}++$ ;  $\text{swap}(u[\text{depth}], u[i])$ 
   $\text{remove capability}(\text{all sub}, \text{curpath}[\text{pathlen}])$ ;
  ifBacktrack ( $\text{depth} + 1, u, \text{curpath}, \text{pathlen}$ ) = 1 return 1;
   $\text{add capability}(\text{allsub}, \text{curpath}[\text{pathlen}])$ ;
   $\text{pathlen}++$ ;  $\text{swap}(u[\text{depth}], u[i])$ 
return 0; }
```

ALGORITHM 1: QEILCM ($u[0..n-1]co[0, k-1]$).

TABLE 2: Algorithm performance under different parameter values.

n	m	k	l	sc	uc	N/A	st (ms)	ut (ms)
30	15	2	3	10	10	0	<1	3
60	15	2	5.5	12	8	1	1	4
120	15	2	8	15	5	0	1	4
120	15	4	7.5	14	6	0	1	3
120	20	4	10	8	11	1	2	7
120	30	4	7	2	18	0	4	6
200	30	4	8.2	5	14	1	10	274
400	30	4	20	9	8	3	2500	7300
2000	50	8	35	3	15	2	2200	11300
5000	50	8	82	4	6	10	17150	$5.2 * 10^5$

FIGURE 3: Influence of function prune and SortbyPRI on performance.

which a process will rise to cover the labels of code that the process executes and data that it reads. For integrity protection, the Low-Water Mark Policy of the Biba model

permits subjects to "read down" objects, but adjusts the current level of a subject to the lowest level of the object it has read. LOMAC [40] is a dynamically loadable extension for

COTS Linux kernels based on the Low-Water Mark model. UMIP [41] uses only high and low integrity levels to preserve system integrity in the face of network-based attacks, where a high-integrity process may drop its integrity to low when it receives remote network traffic, and a low-integrity subject cannot perform key operations. IFEDAC [42] improves UMIP to provide strong user separation, where the integrity labels of a subject are the ID set of users who have contaminated the subject, and a process integrity level will be contaminated by the object it executes or reads. In IFEDAC, each object has read/write/manage protect classes, which are the sets of users who are authorized to read/write/manage it and originated from system DAC policy. Each subject can read/write/manage it when its level is no less than the object's read/write/manage protect class. Mao et al. [42] developed several techniques to automate the generation of low-level policies of the taint-tracking model from data contained in software package managers and logs that capture normal usage of these systems. However, these studies do not involve the problem of the covert channel in implicit label change model and some of them might not uphold noninterference.

Meanwhile, specifying and managing information flow model policies is a challenging problem. Much work develops verification techniques for access control policies. Such analysis on traditional access control models has been done well. In their landmark paper, Harrison et al. [43] first showed that in general, the safety analysis problem in the access control matrix model is undecidable. Jha et al. [44] proved that the problems of security analysis in ARBAC97 model are PSPACE-complete; further, a number of special cases of the problems are NP-complete. When it comes to policy verifications of information flow models, Chaudhuri et al. [23] presented a logic-programming language and tool, called EON, to model and analyze dynamic access control models of the Asbestos and Windows Vista systems. Query evaluation in EON can be reduced to decidable query satisfiability in a fragment of Datalog. Harris et al. [24] presented a method of abstraction that allows a model checker to verify policy soundly for DIFC programs that manipulate potentially unbounded sets of processes, principals, and communication channels. Harris et al. [20] also presented a technique that automatically turns a DIFC-unaware application with information flow requirements into a Flume application. Guttman et al. [45] presented a systematic way based on model checking to determine the information flow security properties of SELinux systems. Hicks et al. [46] modeled the SELinux MLS model, using a logical specification and implemented that specification in the Prolog language. Nevertheless, few of these studies in information flow policy analyze the computational complexity of policy verification.

6. Conclusions and Future Work

Aiming at the vulnerability and potential security risks of mobile operating systems, this paper proposes a flexible dynamic fine-grained security model. It can effectively protect the security of private data in untrusted operating system environments. In particular, our model can close the known covert channels because EILCM regards potential

information flow situation as a possible information flow and expands the semantics of information propagation. It also solves the contradictions between users' sharing of common software and authorization constraints for separation of duty or least privilege through mutually exclusive tags constraints.

We use two ways to formally analyze its safety. Firstly, in terms of Ryan and Schneider's definition of noninterference [26], we verify that a known covert channel in the traditional implicit label change models does not exist in EILCM. Secondly, we used an automated checker to prove that abstract EILCM systems have the security property of noninterference with declassification due to our definition of noninterference with declassification based on standard CSP-oriented specification.

We prove that the problem of EILCM policy verification is an NP-complete problem through reducing the 3SAT problem to it. We also present a backtrack-based search algorithm to solve the problem. Experiments show that the algorithm is effective. This conclusion can reflect to some extent the computational complexity of the problem of Flume policy verification due to their partial similarity.

In the future, we plan to use our analysis method to verify the safety of other relevant models.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the National Natural Science Foundation of China under grant nos. 61972040 and 62176265, the Premium Funding Project for Academic Human Resources Development in Beijing Union University under grant no. BPHR2020AZ03, and the National Key R&D Program of China under grant no. 2018YFB0803603.

References

- [1] K. Chen, Y. Zhang, and P. Liu, "Leveraging information asymmetry to transform android apps into self-defending code against repackaging attacks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1879–1893, 2018.
- [2] J. Zhang, A. R. Beresford, and I. Sheret, "SensorID: sensor calibration fingerprinting for smartphones," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, pp. 638–655, San Francisco, CA, USA, May 2019.
- [3] S. Maruyama, S. Wakabayashi, and T. Mori, "Tap'n ghost: a compilation of novel attack techniques against smartphone touchscreens," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, pp. 620–637, San Francisco, CA, USA, May 2019.
- [4] J. Gamba, M. Rashed, A. Razaghpanah, and J. Tapiador, "An analysis of pre-installed android software," in *Proceedings of*

- the 2020 IEEE Symposium on Security and Privacy, pp. 1039–1055, San Francisco, CA, USA, May 2020.
- [5] W. Qi, W. Ding, and X. Wang, “Construction and mitigation of user-behavior-based covert channels on smartphones,” *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 44–57, 2018.
  - [6] D. Shen, Z. Li, X. Su, J. Ma, and R. Deng, “TinyVisor: an extensible secure framework on android platforms,” *Computers & Security*, vol. 72, pp. 145–162, 2018.
  - [7] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, “Ryoan: a distributed sandbox for untrusted computation on secret data,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI’16)*, pp. 533–549, USENIX Association, Savannah, GA, USA, November 2016.
  - [8] A. C. Myers and B. Liskov, “Protecting privacy using the decentralized label model,” *ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 4, pp. 410–442, 2000.
  - [9] C. K. Ginny, C. Kumar, and K. Naik, “Smartphone processor architecture, operations, and functions: current state-of-the-art and future outlook: energy performance trade-off,” *The Journal of Supercomputing*, vol. 77, no. 2, pp. 1377–1454, 2021.
  - [10] Y. Zhao, D. Sanán, F. Zhang, and Y. Liu, “Refinement-based specification and security analysis of separation kernels,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 127–141, 2019.
  - [11] Y. Liu, T. Song, and L. Liao, “TPII: tracking personally identifiable information via user behaviors in HTTP traffic,” *Frontiers of Computer Science*, vol. 14, no. 3, Article ID 143801, 2020.
  - [12] G. Barthe, G. Betarte, J. D. Campo, and C. Luna, “System-level non-interference of constant-time cryptography. Part I: model,” *Journal of Automated Reasoning*, vol. 63, no. 1, pp. 1–51, 2019.
  - [13] N. B. Said and I. Cristescu, “End-to-end information flow security for web services orchestration,” *Science of Computer Programming*, vol. 187, Article ID 102376, 2020.
  - [14] Z. Wu, X. Chen, Z. Yang, and X. Du, “Reducing security risks of suspicious data and codes through a novel dynamic defense model,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2427–2440, 2019.
  - [15] A. Nadkarni, B. Andow, W. Enck, and S. Jha, “Practical DIFC enforcement on android,” in *Proceedings of the 25th USENIX Security Symposium*, pp. 1119–1136, USENIX Association, Austin, TX, August 2016.
  - [16] P. Efstathopoulos, M. Krohn, and S. Frey, “Labels and event processes in the asbestos operating system,” *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, 2005.
  - [17] N. Zeldovich, S. B. Wickizer, E. Kohler, and D. Mazieres, “Making information flow explicit in HiStar,” in *Proceedings of Usenix Association 7th Usenix Symposium on Operating Systems Design and Implementation (OSDI’2006)*, pp. 263–278, USENIX Association, Seattle, WA, November 2006.
  - [18] M. Krohn and E. Tromer, “Noninterference for a practical DIFC-based operating system,” in *Proceedings of the 2009 IEEE Symposium on Security and Privacy*, pp. 61–76, IEEE, Berkeley, CA, USA, May 2009.
  - [19] D. Schultz and B. Liskov, “IFDB: decentralized information flow control for databases,” in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 43–56, ACM, Prague, Czech Republic, April 2013.
  - [20] W. R. Harris, S. Jha, and T. Reps, “DIFC programs by automatic instrumentation,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS’2010)*, pp. 284–296, ACM, Chicago IL, USA, October 2010.
  - [21] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel, “Laminar: practical fine-grained decentralized information flow control,” in *Proceedings of 2009 ACM Sigplan Conference on Programming Language Design & Implementation*, pp. 63–74, ACM, Dublin, Ireland, June 2009.
  - [22] J. Burket, P. Mutchler, M. Weaver, M. Zaveri, and D. Evans, “GuardRails: a data-centric web application security framework,” in *Proceedings of the 2nd USENIX Conference on Web Application Development*, pp. 1–12, USENIX, Portland, OR, USA, June 2011.
  - [23] A. Chaudhuri, P. Naldurg, S. K. Rajamani, G. Ramalingam, and L. Velaga, “EON: modeling and analyzing dynamic access control systems with logic programs,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS’2008)*, pp. 381–390, ACM, New York, USA, October 2008.
  - [24] W. R. Harris, N. A. Kidd, S. Chaki, S. Jha, and T. Reps, “Verifying information flow control over unbounded processes,” in *Proceedings of 16th International Symposium on Formal Methods (FM 2009)*, pp. 773–789, Springer, Eindhoven, Netherlands, October 2009.
  - [25] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall International, Englewood Cliffs, NJ, USA, 1985.
  - [26] P. A. Ryan and S. A. Schneider, “Process algebra and non-interference,” *Journal of Computer Security*, vol. 9, no. 1-2, pp. 75–103, 2001.
  - [27] J. A. Goguen and J. Meseguer, “Security policies and security models,” in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pp. 11–20, IEEE, Oakland, CA, USA, April 1982.
  - [28] A. G. Ramirez, J. Schmaltz, F. Verbeek, B. Langenstein, and H. Blasum, “On two models of noninterference: rushby and greve, wilding, and vanfleet,” in *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security*, pp. 246–261, IEEE, Florence, Italy, September 2014.
  - [29] P. Baldan and A. Beggato, “Multilevel transitive and intransitive non-interference, causally,” in *Proceedings Of the 18th International Conference On Coordination Languages And Models*, pp. 1–17, IFIP, Heraklion, Greece, June 2016.
  - [30] W. Jamroga and M. Tabatabaei, “Strategic noninterference,” in *Proceedings of the 2015 International Conference On ICT Systems Security And Privacy Protection*, pp. 67–81, IFIP, Hamburg, Germany, May 2015.
  - [31] Formal Systems (Europe) Ltd., *Failures Divergences Refinement: FDR2 Manual*, <https://www.fsel.com>, 2005.
  - [32] A. Bossi, C. Piazza, and S. Rossi, “Modelling downgrading in information flow security,” in *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW’2004)*, pp. 187–201, IEEE, Pacific Grove, CA, USA, June 2004.
  - [33] J. Rushby, *Noninterference, Transitivity, and Channel-Control Security Policies*, CSL-92-02, Stanford Research Institute, Menlo Park, CA, USA, 1992.
  - [34] A. Nanevski, A. Banerjee, and D. Garg, “Verification of information flow and access control policies with dependent types,” in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pp. 165–179, IEEE, Berkeley, CA, USA, May 2011.
  - [35] P. Aldous and A. Might, “Static analysis of non-interference in expressive low-level languages,” *Static Analysis*, vol. 9291, pp. 1–17, 2015.
  - [36] A. J. Vaughan and S. Chong, “Inference of expressive declassification policies,” in *Proceedings of the 2011 IEEE*

- Symposium on Security and Privacy*, pp. 180–195, IEEE, Berkeley, CA, USA, May 2011.
- [37] T. Murray, D. Matichuk, M. Brassil et al., “Sel4: from general purpose to a proof of information flow enforcement,” in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pp. 415–429, IEEE, Berkeley, CA, USA, May 2013.
  - [38] S. Eggert, R. V. D. Meyden, H. Schnoor, and T. Wilke, “The complexity of intransitive noninterference,” in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pp. 196–211, IEEE, Berkeley, CA, USA, May 2011.
  - [39] M. D. McIlroy and J. A. Reeds, “Multilevel security in the UNIX tradition,” *Software: Practice and Experience*, vol. 22, no. 8, pp. 673–694, 1992.
  - [40] F. Timothy, “LOMAC: low water-mark integrity protection for COTS environments,” in *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pp. 230–245, IEEE, Berkeley, CA, USA, May 2000.
  - [41] N. Li, Z. Mao, and H. Chen, “Usable mandatory integrity protection for operating systems,” in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pp. 164–178, IEEE, Berkeley, CA, USA, May 2007.
  - [42] Z. Mao, N. Li, H. Chen, and X. Jiang, “Trojan horse resistant discretionary access control,” in *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT’2009)*, pp. 237–248, ACM, Stresa, Italy, June 2009.
  - [43] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, “Protection in operating system,” *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.
  - [44] S. Jha, N. H. Ninghui Li, M. W. TripunitaraQihua, and W. Winsborough, “Towards formal verification of role-based access control policies,” *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 242–255, 2008.
  - [45] J. D. Guttman, A. L. Herzog, J. D. Ramsdell, and C. W. Skorupka, “Verifying information flow goals in security-enhanced Linux,” *Journal of Computer Security*, vol. 13, no. 3, pp. 115–134, 2005.
  - [46] B. Hicks, S. Rueda, L. S. Clair, T. Jaeger, and P. McDaniel, “A logical specification and analysis for SELinux MLS policy,” *ACM Transactions on Information and System Security*, vol. 13, no. 3, pp. 26–31, 2010.