Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 6, June 2019, pp. 1631-1641
DOI: 10.23940/ijpe.19.06.p14.16311641

Role Behavior Detection Method of Privilege Escalation Attacks for Android Applications

Hui Li[a,b,*], Limin Shen[a], Chuan Ma[a], and Mingyuan Liu[a]

[a]*School of Information Science and Engineering, Yanshan University, Qinhuangdao, 066004, China*
[b]*School of Business Administration, Hebei Normal University of Science and Technology, Qinhuangdao, 066004, China*

Abstract

For privilege escalation attacks in the Android system, the detection method of role behavior was proposed based on component features and process algebra. The classification of roles was constructed from the analysis of the privilege escalation attack model. Feature extraction from components includes component permissions, component communication, API calls, and sensitive data flow. Process algebra was used to construct modes of role behavior, and roles of applications were identified through equivalence relation. Finally, the dangerous path was detected in multi-applications, and then applications constituting to privilege escalation attacks were ascertained. The experiment showed that the proposed method can effectively detect privilege escalation attacks, the potential safe hazards in applications were pointed out, and the role of applications was identified.

Keywords: Android; privilege escalation attack; role behavior; detection method; process algebra

(Submitted on March 20, 2019; Revised on April 7, 2019; Accepted on June 9, 2019)

1. Introduction

The Android smartphone operating system, due to its design of system performance, usability, security, and development convenience, is favored by many users [1], but its security problems are increasing. The Threat Intelligence Report [2-3] released by Nokia Threat Intelligence Laboratories points out that the number of Android malware samples increased by 31% in 2018 compared with 2017. According to CNNIC's "The 42nd China Statistical Report on Internet Development" [4], 360 Internet Security Center's "2017 Q2 China Mobile Security Report" [5], and "2018 Q3 China Mobile Security Report" [6], 54% of Internet users encountered information security incidents in the first half of 2018; 62.1% of malicious programs stole privacy on the Android platform; and compared with 2016, unauthorized access to permission of "location information" increased from 2.0% to 4.5% in live apps of the Android system. Android apps can obtain software and hardware information in devices, steal user privacy data, turn on phone traffic and telephone charges deliberately, and act as terminal zombies through escalation of permissions, multi-application conspiracy, or implant malicious applications.

Zhongyang et al. [7] designed the DroidAlarm to analyze potential leaks and identify leak paths. Heuser et al. [8] proposed a forensic analysis method of Android application-layer privilege escalation attack based on ASM access control architecture. Zhou et al. [9] introduced a privilege escalation attack framework for kernel mandatory access control based on dynamic stain tracking. Bhandari et al. [10] proposed a method for simultaneously analyzing multiple applications to check for collusion among applications. Bugiel et al. [11] adopted a security framework for extending Android middleware and deploying mandatory access control in the kernel to detect and prevent application-level privilege escalation attacks. Lee et al. [12] put forward a method to protect the Android platform by monitoring important system calls. Niazi et al. [13] designed SAndroid to monitor and detect malicious applications by tracking system logs and malicious process signatures. Felt et al. [14] proposed IPC Inspection to prevent privilege escalation attacks; when an application receives messages from other applications, it must reduce the set of permissions and delete the permissions that other applications do not have. Wang et al. [15] had put forward detection methods based on component, application layer and stacking defect. Yu [16] proposed a method for monitoring the parameter list in the function code to detect the leakage of sensitive information.

* Corresponding author.
E-mail address: lh_23@163.com

Dasgupta et al. [17] put forward a multi-user permission strategy and formulated a methodology for shared-trustworthy access. Wu et al. [18] proposed a method for detecting Android malware based on the static feature mechanism.

In summary, although the dynamic and static detection methods provided above have achieved some results for the issue of privilege escalation attacks, there are two limitations:

(1) Applications that constitute privilege escalation attack play different roles, and there is no clear definition of roles.

(2) For applications that may constitute a threat of attack, they cannot to point out the potential defects.

In view of the above limitations, we proposed a method to detect the role behavior of privilege escalation attacks. The contributions are as follows:

(1) Role classification. We analyzed the model of privilege escalation attacks and classified the application roles that constitute the attacks.

(2) Role behavior modeling and detection. The role behavior was modeled and detected based on extracting features of the role behavior and process algebra.

(3) The application of safety defects was pointed out. In the process of role behavior detection, it was shown that the application has security risks and does not constitute a privilege escalation attack.

(4) Test results demonstrated that the method can effectively detect attacks, identify roles, and point out potential safety hazards in the application.

2. Role Classification of Privilege Escalation Attacks

2.1. Model of Privilege Escalation Attacks

Privilege escalation attacks occur in the case of application collusion. Android's security architecture cannot guarantee that the caller and the callee have the same privilege, meaning an application with less (or no) privilege can access other components with more privilege applications [19]. Currently, Android applications (abbreviated as apps) are based on four components: Activity, Service, ContentProvider, and BroadcastReceiver. The model of privilege attacks is given in Figure 1.

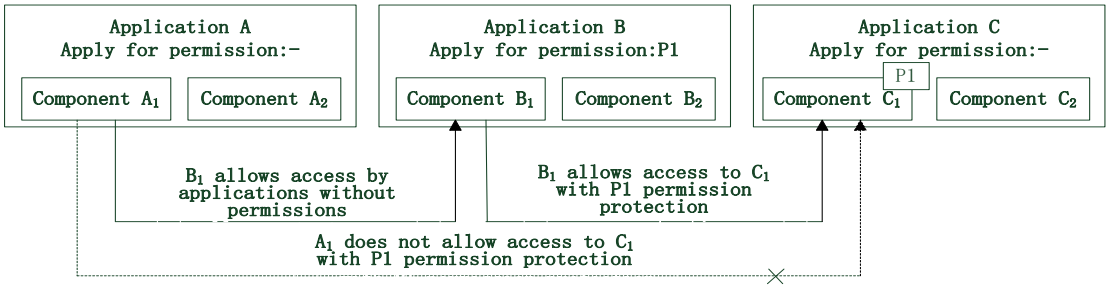


Figure 1. Model of privilege escalation attacks

Figure 1 shows that the applications A, B, and C can constitute privilege escalation attacks for the permission P1. Applications A, B, and C can run on the Android platform independently. Application A has no permission P1. Application B has components B_1, B_2, and permission P1, and components B_1 and B_2 are protected by P1. Component C_1 of application C has permission P_1, and C_1 is protected by P1. Component A_1 can access component B_1 without permission, and B_1 can access component C_1 after having P1 permission. Component A_1 has permission P1 without applying for P1, which constitutes a privilege escalation attack.

2.2. Privilege Escalation Attack Case

In order to illustrate our approach, a privilege escalation attack case consisting of ExampleA, ExampleB, and ExampleC, which run independently, was given. The key code is shown in Figure 2.

2.3. Role Classification of Privilege Escalation Attacks

According to 2.1 and 2.2, apps can be classified according to the tasks of the application's responsibilities in the privilege escalation attack.

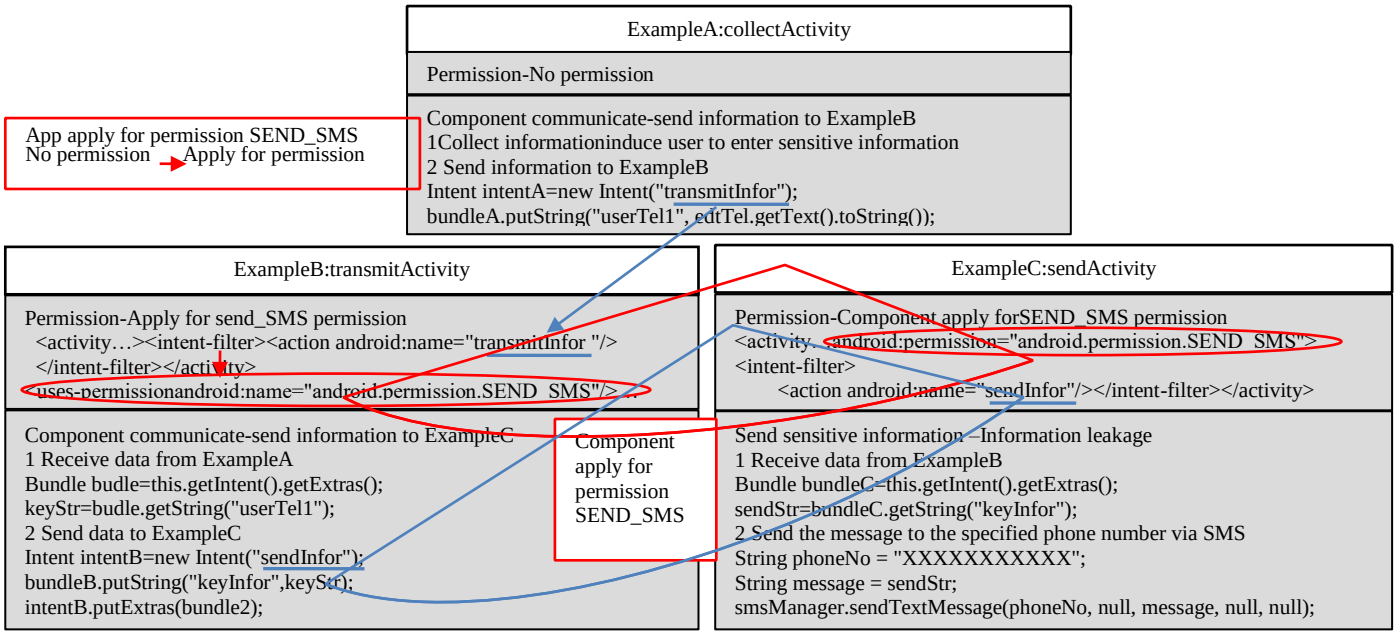


Figure 2. Privilege escalation attack case

Definition 1: Collecting role. This role utilizes the UI interface to induce users to input sensitive information, meaning it collects users' sensitive and available information.

Definition 2: Transmission role. This role has a dangerous permission that is required for this application. It provides an escalation of permission and transmission sensitive information to other applications' components.

Definition 3: Sending role. This role has a dangerous permission that sends sensitive information locally.

Definition 4: Safety hazard application. The application cannot constitute a privilege escalation attack, but there are security hazards such as abuse of permission.

Definition 5: Safety application. There are no security hazards.

3. Construct Role Behavior Detection Model

3.1. Extract Role Behavior Features

This method performed static and dynamic feature extraction on components to build a BFT (behavior feature table). Static features include permissions, components, components' communication, sensitive APIs, and pairs of sensitive data flow. The dynamic features are sensitive API calls.

3.1.1. Extract Permissions and Components

The information of permissions and components in AndroidManifest.XML are extracted.

- **Step 1** Extract the permission of the application according to the tag < uses-permission >;
- **Step 2** Extract registered components according to < activity ></activity > and other tags;
- **Step 3** Extract Intent actions of components based on Step 2;
- **Step 4** Extract permissions applied for components;
- **Step 5** Store the extracted permissions and components into BFT.

3.1.2. Extract Component Communication

The unencrypted APK file was converted into Smali file by apktool. Smali file was analyzed and extracted to obtain the information of component communication.

- **Step 1** Induce each direct method/virtual method;
- **Step 2** Extract the call relationship between functions based on Step 1;
- **Step 3** According to Intent communication, extract the special function calls (getExtras, putExtras, etc.);
- **Step 4** Store information of component communication into BFT.

3.1.3. Extract Sensitive API Calls

Apps will call sensitive APIs while using dangerous permissions. The official Google document provides a list of dangerous permissions. PScout [20] analyzed and summarized the permissions and API correspondence of multiple versions of the Android system. A sensitive API set could be constructed by combining dangerous permissions and the reference [20]. Strace, which comes with Android SDK, extracted the sequence of system calls to obtain sensitive API calls.

- **Step 1** Obtain the list of dangerous permissions;
- **Step 2** Build a mapping between dangerous permissions and APIs, and extract the sensitive API set;
- **Step 3** The system call sequence was obtained by Strace, and the sensitive API call was obtained by combining with the sensitive API set;
- **Step 4** Save sensitive API calls into BFT.

3.1.4. Extract Pairs of Sensitive Data Flow

FlowDroid extracted pairs of sensitive data flow by analyzing content, flow, field, and object-sensitive, that is, <source,sink> [21]. The source is the sensitive information acquisition source, and the sink is the sensitive information transmission source. The extracted pairs of sensitive data flow were added to BFT.

3.2. Model of Role Behavior based on Process Algebra

3.2.1. Extended Process Algebra

Process algebra [22-23] can effectively describe Android architecture and message communication features. We used process algebra as a modeling language to create a component behavior model based on the BFT of the components. The syntax and semantic specifications of the extended process algebra are given below:

$$P = A(y_1, y_2, \cdots, y_n) | \sum_{i \epsilon I} a_i . P_i \left| \left(\underline{X} < y_1, y_2, \cdots, y_n > | X(y_1, y_2, \cdots, y_n) | \tau \right) . P \right| (\upsilon \chi) P \quad (1)$$

In the formula:

(1) $A(y_1, y_2, \cdots, y_n)$ represents that each process P has a unique process identifier, where y_i represents the free name in P.

(2) $\sum_{i \in I} a_i . P_i = a_1 P_1 + a_2 P_2 + \cdots + a_n P_n$. It is a summation, where I is any finite indexing set. If $I =$, then $\sum_{i \in I} a_i . P_i$ is the empty summation, written as 0, and it indicates that the process is successfully terminated. P_i is guarded by a_i, because P_i must start activities after the action represented by a_i occurs.

(3) $\left(\underline{X} < y_1, y_2, \cdots, y_n > | X(y_1, y_2, \cdots, y_n) | \tau \right) . P$, where $\underline{X} < y_1, y_2, \cdots, y_n >$ represents the action of message sending; $X(y_1, y_2, \cdots, y_n)$ represents the action of message receiving; and τ represents the internal action.

(4) $(\upsilon \chi) P$, and the scope of χ is limited to the subsequent process P, where υ is a qualified identifier.

3.2.2. Role Behavior Modeling

Definition 6: Application behavior. In the Android architecture, applications are composed of components. The behavior of an application is obtained by the behavior of the components. Different components provide different services and

interact with users, which results in a series of internal or external reactions of application. The application behavior is composed of a series of component's actions.

Definition 7: Component behavior. It refers to the conversion and operation of data receiving and sending data by components. The components' actions present service or functionality to the inside and outside of the application. The Android platform has a complete access mechanism, and sensitive behavior of components are protected by permissions. Therefore, component behavior is a set of information sending, receiving, and executing operations under the protection of the permissions. According to Equation (1), the definition of component behavior is defined as follows:

$$\begin{aligned} &ComAction\left((x_1, x_2, \cdots, x_n), \left(\underline{x_1}, \underline{x_2}, \cdots, \underline{x_m}\right), (p_1, p_2, \cdots, p_w)\right) \\ &= x_1(inputContent_1).x_2(inputContent_2).\cdots.x_n(inputContent_n).operation_1.operation_2.\cdots.operation_k.\underline{x_1} \\ &< outputContent_1 >.\underline{x_2} < outputContent_2 >.\cdots.\underline{x_w} < outputContent_w > (p_1.p_2.\cdots.p_w) \end{aligned} \tag{2}$$

Where *n* represents the component with *n* inputs, *m* represents the component with *m* outputs, and *w* represents *w* permissions. $operation_i$ represents the actions of the components, and it is an integral part of the application behavior.

Definition 8: Behavior of collecting role component. According to Definition 1, 6, and 7 (Equations (1) and (2)), it includes at least the operations of calling other application components and sending information.

$$\begin{aligned} &collectComAction\left((u_1, u_2, \cdots, u_n), \left(\underline{x_1}, \underline{x_2}, \cdots, \underline{x_m}\right)\right) \\ &= u_1(dangerContent_1).u_2(dangerContent_2), \cdots, u_n(dangerContent_n).operation_1.operation_2.\cdots. \\ &\quad operation_k.\underline{x_1} < dangerContent_1 >.\underline{x_2} < dangerContent_2 >.\cdots.\underline{x_m} < dangerContent_m > \end{aligned} \tag{3}$$

Where u_i represents that the user has *n* sensitive information inputs or acquires *n* sensitive information of the device, and *m* represents the component's *m* outputs.

Definition 9: Behavior of transmission role component. According to Definitions 2, 6, and 7 (Equations (1) and (2)), it includes sensitive information or operations passed by the component of the collecting role, calling to other application components, sending actions, and requesting for dangerous permissions.

$$\begin{aligned} &transmitComAction\left((x_1, x_2, \cdots, x_n), \left(\underline{x_1}, \underline{x_2}, \cdots, \underline{x_n}\right)\right) \\ &= (\nu P_1)((x_1(dangerContent_1).x_2(dangerContent_2), \cdots, x_n(dangerContent_n).operation_1.operation_2. \\ &\quad \cdots.operation_k.\underline{p_1} < dangerContent_1 >.\underline{p_1} < dangerContent_2 >.\cdots.\underline{p_1} < dangerContent_n > | \\ &\quad (p_1(dangerContent_1).p_1(dangerContent_2).\cdots.p_1(dangerContent_n).operation_1.operation_2. \\ &\quad \cdots.operation_k.\underline{x_1} < dangerContent_1 >.\underline{x_2} < dangerContent_2 >.\cdots.\underline{x_n} < dangerContent_n >) \end{aligned} \tag{4}$$

Definition 10: Behavior of sending role component. According to Definitions 3, 6, and 7 (Equations (1) and (2)), it includes sensitive information passed by the component of transmission role and requests for dangerous permission. SourceSink pairs under sensitive API calls.

$$\begin{aligned} &sendComAction\left((x_1, x_2, \cdots, x_n)\right) \\ &= (\nu P_1)(\nu sourceSink_1)x_1(dangerContent_1).x_2(dangerContent_2).\cdots.x_n(dangerContent_n).operation_1. \\ &operation_2.\cdots.operation_n.\underline{x_1} < dangerContent_1 >.\underline{x_2} < dangerContent_2 >.\cdots.\underline{x_n} < dangerContent_n > \end{aligned} \tag{5}$$

4. Role Behavior Detection Method

According to Definition 1-5 and the equivalence relation of state in the process algebra, three kinds of behavioral equivalence relations are defined: behavior strong equivalence, behavior weak equivalence, and behavior non-equivalent.

Labeled transition system. A labeled transition system (LTS) over Act is a pair (*Q*, *T*), where there are a set *Q* of states and a ternary relation $T \subseteq (Q \times Act \times Q)$, known as a transition relation. If $\forall q \in Q$, $\forall q' \in Q$, $\exists a \in Act$, and $(q, a, q') \in T$, we write $q\ a \rightarrow\ q'$.

Strong simulation. Let (*Q*, *T*) be an LTS, and let *S* be a binary relation over *Q*. If $\forall p$ and $p\ a \rightarrow\ p'$, then there exists q' such that $q\ a \rightarrow\ q'$ and $p'Sq'$, and *S* is called a strong simulation over (*Q*, *T*).

Definition 11: Behavior Strong equivalence. A binary relation *S* over *Q* is said to be a strong bisimulation over the LTS (*Q*, *T*) if both *S* and its converse are simulations. We say that *p* and *q* are strongly equivalent, written as $p \sim q$, if there exists a strong bisimulation *S* such that *pSq*. This means that the internal and external behavior of components is equivalent to one of the three role models.

Weak simulation. *S* is a weak simulation if and only if, whenever *PSQ*, $P \rightarrow P'$, and then there exists Q' such that QQ' and $P'SQ'$. If $P\ \lambda \rightarrow\ P'$, then there exists Q' such that $Q\ \lambda \Rightarrow\ Q'$ and $P'SQ'$.

Definition 12: Behavior weak equivalence. A binary relation *S* over *P* is said to be a weak bisimulation if both *S* and its converse are weak simulations. We say that *P* and *Q* are weakly bisimilar, written as $P \approx Q$, if there exists a weak bisimulation *S* such that *PSQ*. This means that the internal behavior is not equivalent, while the external representation is equivalent, and applications have security risks according to Definition 4.

Definition 13: Behavior non-equivalence. Based on Definitions 11 and 12, *P* and *Q* are *LTS*; whenever *P* chooses any transition, *Q* cannot find an appropriate transition to match. That is, there is no strong (weak) simulation *R*, and then *P* and *Q* are not equivalent, written as $P \neq Q$. This means that the internal and external representations of component behavior are not equivalent to the three types of role models, and they are called security applications according to Definition 5.

Definition 14: Dangerous path. There is an information transmission path among the three types of roles, and there is a dangerous permission.

Algorithm 1: Component behavior detection algorithm based on process algebra. Assume a component $Com_i(0 < i \leq w)$ of the Apk was tested, where *w* is the number of Apk components. $Action_j(0 < j \leq v)$, where *v* is the number of actions of the corresponding component. P_1 is a dangerous permission.

Algorithm 1: Component behavior detection algorithm based on process algebra

Input: $Com_i(1,2,\cdots,w)$, $Action_j(1,2\cdots,v)$, $collectComAction\left((u_1,u_2,\cdots,u_n),\left(\underline{x_1},\underline{x_2},\cdots,\underline{x_m}\right)\right)$, $transmitComAction\left((x_1,x_2,\cdots,x_n),\left(\underline{x_1},\underline{x_2},\cdots,\underline{x_m}\right)\right)$, $sendComAction((x_1,x_2,\cdots,x_n))$, P_1

Output: String collectApk, transmitApk, sendApk, hazardApk, safetyApk

1. Assumption: use *PA* point to $A(\forall i,j=1,2,\cdots,m)=Com_i\sum_{j=1}^{m}Action_j$, use *PC* point to collectComAction, use *PT* point to transmitComAction, and use *PS* point to sendComAction
2. Initialization: $PA \rightarrow A_i, PC \rightarrow colllectComAction_k, PT \rightarrow transmitComAction_l, PS \rightarrow sendComAction_h$
3. For each component in $A(\forall i,j=1,2,\cdots,m)$
4. Construction $A_i, collectComAction_i, transmitComAction_i, sendComAction_i$
5. If $PA{\sim}PC$ Then print $A.A_i$ to collectApk, $PA=PA\rightarrow next, PC=PC\rightarrow next$// $A.A_i$ represents a component of App *A*
6. ElseIf $PA{\sim}PT$ Then print $A.A_i$ to transmitApk, $PA=PA\rightarrow next,\ PT=PT\rightarrow next$
7. ElseIf $PA{\sim}PS$ Then print $A.A_i$ to sendApk, $PA=PA\rightarrow next, PS=PS\rightarrow next$
8. ElseIf $PA\approx PC$ or $PA\approx PT$ or $PA\approx PS$ Then print $A.A_i$ to hazardApk, $PA=PA\rightarrow next, PC=PC\rightarrow nent,\ PT=PT\rightarrow next, PS=PS\rightarrow next$
9. ElseIf $PA\neq PC$ and $PA\neq PT$ and $PA\neq PS$ Then print *A* to safetyApk

Algorithm 2: Dangerous path detection algorithm. The feature representation of the component was obtained by the BFT of each role component: application name, component name (permission, receiving interface, sending interface, sensitive API call, sensitive data flow pair). Take the transmission role as the starting point, search the sending role backward, output the matching content, and store it as an array object string. Then, use the output array object string as input to search the collecting role forward. If there is a complete path among the role's components, then they constitute a privilege escalation attack.

Algorithm 2: Dangerous path detection algorithm

Input: String collectApkCom, transmitApkCom, sendApkCom// collectApkCom represents components' features expression of the collecting role

Output: dangerousPath

1. $arrayObj_0$=getStartPoint(transmitApkCom)
2. for each $arrayObject_i$ in transmitApkCom
3. for s=0 to sendApkCom.length- transmitApkCom.length
4. if $arrayObject_i$ [1⋯m] == sendApkCom [s+1,s+m]
5. print $arrayObject_i$ to Temp1[s] //Temp1[s] searched array object

// Represent transmitApk and sendApk have path

```
6.  searchArrayObj0= Temp1[0]
7.  k= Temp1.length
8.  for j=0 to Temp1[k-1]
9.    if arrayObjecti [1…m] == collectApkCom [s+1,s+m]
10.     print arrayObjecti to Temp2//transmitApk and collectApk have path
11. Combine Temp1 and Temp2 then add to dangerousPath
12. print dangerousPath
```

5. Detection Method Experiment

5.1. Extract Roles Component Features

According to 3.1, the BFT of the components of ExampleA, ExampleB, and ExampleC are shown in Tables 1, 2, and 3, respectively.

Table 1. BFT of ExampleA/collectActivity

Feature	Dangerous permission	ActionName	Receiving interface	Sending interface	Call sensitive API	Sensitive data flow	Sensitive data
ExampleA/ collectActivity	----	----	----	Intent.ActionName= transmitInfor; putString()/putExtras()	----	----	telNum

Table 2. BFT of ExampleB/transmitActivity

Feature	Dangerous permission	ActionName	Receiving interface	Sending interface	Call sensitive API	Sensitive data flow	Sensitive data
ExampleB/ transmitActivity	Android.permission.SEND_SMS	ActionName ="transmitInfor"	getInent().getExtras/getString()	Intent.ActionName="sendInfor"; putString()/putExtras()	void enforceReceiveAndSend(Java.lang.String)	----	telNum

Table 3. BFT of ExampleC/sendActivity

Feature	Dangerous permission	ActionName	Receiving interface	Sending interface	Call sensitive API	Sensitive data flow	Sensitive data
ExampleC/ sendActivity	Android.permission.SEND_SMS	AnctionName= "transmitInfor"	getInent().getExtras getString()/getExtras (KeyInfor)	----	void enforceReceive AndSend(Java.lang.String)	(Bundle.getString() /sendActivity.onCreate(), Log/sendMessage. sendSMSMessage)	telNum

5.2. Modeling and Detection of Role Component Behavior

5.2.1. Modeling and Detection of Role Behavior of the Case

For the BFT of the component of the privilege escalation attack case given in 5.1, Definition 6 and Equations (1) and (2) were used to model the following components:

(1) ExampleA's collectActivity component was modeled as follows:

$$A: collectActivityComAction\left(x_1, \underline{x_1}\right) = x_1(telNum).\underline{x_1} < telNum >$$

(2) ExampleB's transmitActivity component was modeled as follows:

$$B: transmitActivityComAction\left(x_1, \underline{x_1}\right) = ((x_1(telNum).\underline{P_1} < telNum >)|(P_1(telNum).\underline{x_1} < telNum >)|\tau).P_1$$

(3) ExampleC's sendActivity component was modeled as follows:

$$C: sendActivityComAction = \left(x_1(telNum).\underline{x_1} < telNum >\right).P_1.sourceSink_1$$

(4) Definitions 8-10 (Equations (3)-(5)) were used to model the role behavior of the case, as follows:

1) Modeling the behavior of collecting role components:

$$M_1: collectComAction\left(u_1, \underline{u_1}\right) = u_1(telNum).\underline{u_1} < telNum >$$

2) Modeling the behavior of transmission role components:

$$M_2: transmitComAction\left(x_1, \underline{x_1}\right) = (\nu P_1)((x_1(telNum).\underline{P_1} < telNum >)|(P_1(telNum).\underline{x_1} < telNum >))$$

3) Modeling the behavior of sending role components:

$$M_3: sendComAction = (\nu P_1)(\nu sourceSink_1).x_1(telNum).\underline{x_1} < telNum >$$

(5) According to the syntax of MWB, the above components' modeling was converted into MWB language. According to Definition 11, the strong equivalence verification between the collectActivity component (represented by A1) and the collecting role component model (represented by M1) is shown in Figure 3.

```
The Mobility Workbench
 (MWB'99, version 4.135, built Thu Jan 08 13:13:41 2004)

MWB>agent A1(x,y)=x(telNum).'y<telNum>.A1(x,y)
MWB>agent M1(x,y)=x(telNum).'y<telNum>.M1(x,y)
MWB>eq A1 M1
The two agents are equal.
Bisimulation relation size = 2.
```

Figure 3. Behavior strong equivalence verification

According to Definition 14 and Algorithms 1 and 2, we can see the following:

(1) ExampleA belonged to the collecting role. Similarly, it could be verified that ExampleB belonged to the transmission role, and ExampleC belonged to the sending role.

(2) There was a dangerous path: (ExampleA.collectActivity) -> (ExampleB.transmitActivity) -> (ExampleC.sendActivity).

(3) ExampleA, ExampleB, and ExampleC constituted a privilege escalation attack.

5.2.2. Modeling and Detection to Extend the Role Behavior of the Case

The features of the transmitActivity and sendActivity components were adjusted and verified weak equivalence and non-equivalence behavior.

(1) Assuming that the data of the transmitActivity component was sent internally, the transmitActivity component of ExampleB was modeled as follows:

$$B_1: transmitActivityComAction_1\left(x_1, \underline{x_1}\right) = x_1(telNum).\underline{x_1} < telNum >$$

$$B_2: transmitActivityComAction_2\left(x_1, \underline{x_1}\right) = (\nu P_1)((x_1(telNum).\underline{P_1} < telNum >)|(P_1(telNum).\underline{x_1} < telNum >))$$

$$B_{20}: transmitActivityComAction_{20}\left(x_1, \underline{x_1}\right) = x_1(telNum).transmitActivityComAction_{21}(x_1, \underline{x_1}, telNum)$$

$$\begin{aligned} B_{21}: & transmitActivityComAction_{21}\left(x_1, \underline{x_1}, telNum\right) \\ & = x_1(sourceSink_1).transmitActivityComAction_{22}\left(x_1\underline{x_1}, telNum, sourceSink_1\right).\underline{x_1} < telNum \\ & >.transmitActivityComAction_{20}(x_1, \underline{x_1}) \end{aligned}$$

$$\begin{aligned} B_{22}: & transmitActivityComAction_{22}\left(x_1, \underline{x_1}, telNum, sourceSink_1\right) = \underline{x_1} < telNum \\ & >.transmitActivityComAction_{21}(x_1, \underline{x_1}, sourceSink_1) \end{aligned}$$

(2) Assuming that the sendActivity component was not protected by P_1 permission and there is no sensitive path, the sendActivity component of ExampleC was modeled as follows:

$$C_1: sendActivityComAction_1\left(x_1, \underline{x_1}\right) = x_1(telNum).\underline{x_1} < telNum >$$

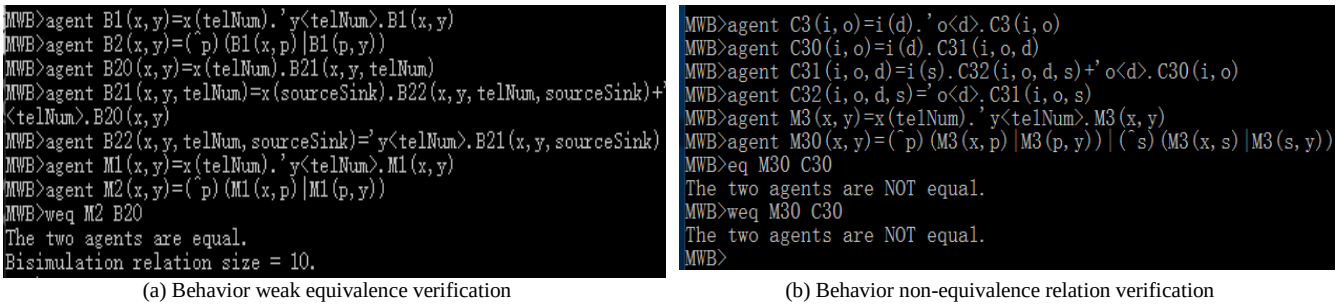
$$C_{20}: sendActivityComAction_{20}\left(x_1, \underline{x_1}\right) = x_1(telNum).sendActivityComAction_{21}(x_1, \underline{x_1}, telNum)$$

$$C_{21}: sendActivityComAction_{21}\left(x_1, \underline{x_1}, telNum\right) = x_1(sourceSink_1).sendActivityComAction_{22}\left(x_1, \underline{x_1}, telNum, sourceSink_1\right) + \underline{x_1} < telNum >.sendActivityComAction_{20}\left(x_1, \underline{x_1}\right)$$

$$C_{22}: sendActivityComAction_{22}\left(x_1, \underline{x_1}, telNum, sourceSink_1\right) = \underline{x_1} < telNum > .sendActivityComAction_{21}\left(x_1, \underline{x_1}, sourceSink_1\right)$$

According to Definitions 12 and 13, the equivalence verification of the transmitActivity component, the sendActivity component, and the roles component model is shown in Figure 4.

Verification showed that the transmitActivity component of ExampleB, which changed the sending interface, was weakly equivalent to the transmitComAction model. The sendActivity component of ExampleC without the permission of P_1 was not equivalent to the sendComAction model. This proved that the changed ExampleB is an application with security hazards, and ExampleC without the permission of P_1 is a secure application.



(a) Behavior weak equivalence verification (b) Behavior non-equivalence relation verification

Figure 4. Behavior weak equivalence and non-equivalence relation verification

6. Evaluation and Effectiveness Analysis

6.1. Experimental Evaluation

The key steps of this detection method are role modeling, sensitive data flow detection, and dangerous path detection. The time complexity of the algorithm is O(m). 56 Android Apks samples were tested, and the time and space cost analysis data are shown in Table 4.

Table 4. Time cost and space cost

	Minimum	Maximum	Average value
Time cost	380.497 S	93.163 S	130.175 S
Space cost	856.5534 M	922.0956 M	888.3262 M

6.2. Method Comparison

According to the component-based detection method in reference [15], 56 sample Apks were tested, and the test data comparison results are shown in Table 5. It can be seen that the role behavior detection method can detect privilege escalation attacks and role classification effectively.

Table 5. Test data comparison

	Risk of privilege escalation attack /%	Composition privilege escalation attack /%	The role that constitutes the attack /%	Safety application /%	Failed application /%
Component-based detection	57.1%	0%	0%	42.9%	16.1%
Our approach	78.6%	5.4%	37.5%	21.4%	1.8%

6.3. Experimental Validity

53 Apks were extracted from the Android application market such as Google Play, and the research team developed three Apks to constitute a sample set. The validity of this method was verified by testing the samples. The results showed that the application of the collecting role is 1.8%, the application of the transmission role is 30.4%, and the application of the sending role is 5.4%. However, the encryption of the Apk or the confusion of the code may cause obstacles to the detection of the method, so 1.8% of Apks failed, as shown in Table 6.

Table 6. Role distribution

Classification	Collecting role	Transmission role	Sending role	Application with safety hazard	Safety application	Privilege escalation attack	Failed
Number	1	17	3	22	12	3	1
Percentage	1.8%	30.4%	5.4%	39.3%	21.4%	5.4%	1.8%

There were 1,902 components in 56 Apks. They were detected, and there were 293 components with Intent communication, 111 dangerous components, and 58 registered but unused components, as shown in Table 7.

Table 7. Threat distribution for component communication

	Component with Intent communication	Dangerous component	Registered but unused component	Safety component
Number	293	111	58	1440
Percentage	15.4%	5.8%	3.1%	75.7%

Using the detection of 56 Apks, the defects of some applications with safety hazards are shown in Table 8.

Table 8. Defect factors that constitute privilege escalation attacks

Defects	Repeat applied permission	Apply for non-existent permissions	Permission has been disabled (before 5.0)	Apply for dangers permission	Registered but unused component	Dangerous component
Number	21	10	28	39	13	25
Percentage	37.5%	17.9%	50.0%	69.6%	23.2%	44.6%

7. Conclusions

On the basis of defining the role classification and feature extraction of privilege escalation attacks, role behavior modeling and detection were proposed and realized by using process algebra combined with strong equivalence, weak equivalence, and non-equivalence. This method enhanced the detection of app behavior and implemented the role location of apps in privilege escalation attacks and the identification of potential security risks. Compared with the method in reference [15], it can be seen that this method makes up for the blind area that cannot distinguish roles and improves the accuracy of detection. Through the detection of 56 Apks, 37.5% of Apks belong to the privilege attack roles, and 39.3% of Apks have security hazards. Security hazard factors such as abuse of permission and communication of dangerous components are also indicated.

Acknowledgments

This work was partly financially supported through grants from the National Natural Science Foundation of China (No. 61772450), Natural Science Foundation of Hebei Province (No. F2017203307, F2016203290), Science and Technology Project of Hebei Province (No.17210701D), and Youth Fund for Science and Technology Research of Institution of Higher Education in Hebei Province (No. QN2016073).

References

1. S. H. Qing, “Research Progress on Android Security,” *Journal of Software*, Vol. 27, No. 1, pp. 45-71, 2016
2. “Nokia Threat Intelligence Report-2017,” (https://networks.nokia.com/solutions/threat-intelligence, Last accessed on January 30, 2019)
3. “Nokia Threat Intelligence Report-2019,” (https://networks.nokia.com/solutions/threat-intelligence, Last accessed on January 30, 2019)
4. “The 42nd China Statistical Report on Internet Development,” (http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201808/t20180820_70488.htm, Last accessed on February 1, 2019)
5. “China Mobile Security Report in Q2 2017,” (http://zt.360.cn/1101061855.php?dtid=1101061451&did=490633771, Last accessed on February 1, 2019)
6. “China Mobile Security Report in Q32018,” (http://zt.360.cn/1101061855.php?dtid=1101061451&did=210801941, Last accessed on February 1, 2019)

7. Y. B. Zhongyang, Z. Xin, B. Mao, and L. Xie, "DroidAlarm: An All-Sided Static Analysis Tool for Android Privilege-Escalation Malware," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pp. 353-358, 2013
8. S. Heuser, M. Negro, P. K. Pendyala, and A. R. Sadeghi, "DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android," in *Proceedings of International Conference on Financial Cryptography and Data Security*, pp. 260-268, 2017
9. W. M. Zhou, Y. Q. Zhang, and X. F. Liu, "POSTER: A New Framework Against Privilege Escalation Attacks on Android," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 1411-1413, 2013
10. S. Bhandari, F. Herbreteau, V. Laxmi, A. Zemmari, P. S. Roop, and M. S. Gaur, "Detecting Inter-App Information Leakage Paths," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 908-910, 2017
11. S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. R. Sadeghi, and B. Shastry, "Poster: The Quest for Security Against Privilege Escalation Attacks on Android," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 741-744, 2011
12. H. T. Lee, D. Kim, M. Park, and S. J. Cho, "Protecting Data on Android Platform Against Privilege Escalation Attack," *International Journal of Computer Mathematics*, Vol. 93, No. 2, pp. 401-414, 2016
13. R. H. Niazi, J. A. Shamsi, T. Waseem, and M. M. Khan, "Signature-based Detection of Privilege-Escalation Attacks on Android," in *Proceedings of 2015 Conference on Information Assurance and Cyber Security* (*CIACS*), pp. 44-49, 2016
14. "The Effectiveness of Install-Time Permission Systems for Third-Party Applications," (http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-143.pdf, Last accessed on March12, 2019)
15. C. Wang, R. B. Zhang, and G. Li, "Technology of Detection for Privilege Escalation Attack on Android," *Transducer and Microsystem Technologies*, Vol. 36, No. 1, pp. 146-148, 2017
16. D. Yu, "Research and Implementation of a Detection Method for Privilege Escalation Attack of Android System," Peking University, Beijing, 2013
17. D. Dasgupta, A. Roy, and D. Ghosh, "Multi-User Permission Strategy to Access Sensitive Information," *Information Sciences*, Vol. 423, pp. 24-49, 2018
18. D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "DroidMat: Android Malware Detection Through Manifest and API Calls Tracing," in *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security*, pp. 62-69, 2012
19. L. Davi, A. Dmitrienko, A. R. Sadeghi, and M. Winandy, "Privilege Escalation Attacks on Android," in *Proceedings of the 13th International Conference on Information Security*, pp. 346-360, 2011
20. K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pp. 217-228, 2012
21. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, et al., "FlowDroid: PreciseContext, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 259-269, 2014
22. C. A. R. Hoare, "Communicating Sequential Processes," Springer, New York, 1978
23. R. Milner, "A Calculus of Communicating Systems," Springer-Verlag, 1980

Hui Li is a doctoral student in the School of Computer Science and Technology at Yanshan University. Her research interests include information security, mobile application security, and mobile information systems.

Limin Shen graduated from the School of Computer Science and Technology at Yanshan University with a Ph.D. He visited Illinois Institute of Technology in the U.S.A. for collaborative research from 2005 to 2007. He is currently a professor and Ph.D. supervisor at Yanshan University. He is also a senior member of CCF and a member of the China Software Engineering Professional Committee. His current research interests include information security, collaborative computing, and system integration.

Chuan Ma received his doctoral degree in computer science and technology from Yanshan University. His research interests include information security and mobile security.

Mingyuan Liu is a master's student in the School of Computer Science and Technology at Yanshan University. Her research interests include information security and network security.