Article

Risk Measurement Method for Privilege Escalation Attacks on Android Apps Based on Process Algebra

Limin Shen ¹, Hui Li ^{1,2,*} , Hongyi Wang ³, Yihuan Wang ¹, Jiayin Feng ^{1,4} and Yuqing Jian ^{1,4}

¹ School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China; shenllmm@ysu.edu.cn (L.S.); wangyihuan@stumail.ysu.edu.cn (Y.W.); feng_ada2000@stumail.ysu.edu.cn (J.F.); jianpurple@stumail.ysu.edu.cn (Y.J.)

² School of Business Administration, Hebei Normal University of Science & Technology, Qinhuangdao 066004, China

³ Department of Commerce and Trade, Qinhuangdao Vocational and Technical College, Qinhuangdao 066100, China; wanghongyi0515@gmail.com

⁴ School of Mathematics and Information Science & Technology, Hebei Normal University of Science & Technology, Qinhuangdao 066004, China

* Correspondence: lh_23@stumail.ysu.edu.cn

Received: 11 April 2020; Accepted: 28 May 2020; Published: 30 May 2020

Abstract: On the Android platform, information leakage can use an application-layer privilege escalation attack composed of multi-app collusion. However, the detection effect of a single app that can construct privilege escalation attacks is not good. Furthermore, the existing software and app measurement methods are not applicable to the measurement of collusion privilege escalation attacks. We propose a method for measuring the risk of a single app by using process algebra to model and determine the attack behavior, and we construct a measurement function based on sensitive data transitions and the feature set of attack behavior. Through the analysis of the privilege escalation attack model, the feature set of attack behavior is obtained. Then, based on the extracted behavior feature set, process algebra is used to model the dangerous behavior of an app. The dangerous behavior of the app is determined by weak equivalence and non-equivalence, and finally the risk of the app is measured based on the measurement function. Three known applications are used to verify the attack, and the risk measurement values are above 0.98. Based on the classification of applications on the market, we select typical apps in each category to build the test set. Benchmark tests and test set experiments show that the risk measurement results are consistent with the actual detection results, verifying the feasibility and effectiveness of this method.

Keywords: process algebra; privilege escalation attack; risk measurement; behavior feature set

1. Introduction

With the rapid development of the mobile internet and the Internet of Things(IoT), the Android system, which accounts for 40.39% of operation system (OS) market share, has become an important application platform [1]. Therefore, Android apps are widely used in smart homes, smart healthcare, online education, transportation, tourism, and other aspects closely related to people's lives. From the perspective of trends in mobile threats, security problems related to the mobile internet and the IoT are highlighted in the 5G era [2–4]. There are also more and more threats to Android apps that carry a large amount of users' privacy data [5,6]. The Nokia Thread Intelligence Report2019 points out that, in 2018, the average monthly infection rate in mobile networks was 0.31%, and Android devices were responsible for 47.15% of the observed malware infections [7]. Based on the report in [8], it was found that 98.7% of the devices in the security analysis results of the system vulnerability. Compared with

89.7% in 2018, the percentage of privilege escalation vulnerabilities has increased, and the threat by privilege escalation attacks has increased. However, application-layer collusion privilege escalation attacks composed of multi-app collusion is based on multiple independent and secure app collusions. Such attacks can complete the theft of user's privacy data and destructive operation through the collusion of multiple apps. The attack behavior of collusion privilege escalation attacks is more covert and more threatening. Therefore, more and more researchers are focusing on collusion privilege escalation attacks on Android apps.

At present, researchers have made remarkable detection and prevention methods for collusion privilege escalation attacks on Android apps. However, for the current permission granting mechanism, most existing detection and prevention methods become costly or rigid in the recent Android dynamic permission environment [9]. Risk measurement method can avoid the detection of communication between applications and reduce the cost of the method, so it is a feasible solution to measure the risk of the app which can constitute the collusion privilege escalation attacks. At the same time, researchers have put forward many classical methods for the security measurement of software and apps. Therefore, it has a solid research foundation and technical feasibility.

However, due to the strong concealment of collusion privilege escalation attacks, the use of existing methods to measure a single app is not effective. The main problems are as follows: (1) the measurement is mainly based on the overall behavior, performance, or development process of the software, without considering the attack behavior features of collusion privilege escalation attacks; (2) the risk measurement of the app is only based on one feature, such as permission or application programming interface (API) calls, without considering the multi-feature nature of collusion privilege escalation attacks. There is a lack of research on the security measurement of collusion privilege escalation attacks on a single app. Therefore, this paper proposes a method to measure collusion privilege escalation attacks on Android apps based on process algebra. The specific method is described as follows:

- (1) Extraction of the attack behavior feature set and the number of transitions. Based on an analysis of the model of collusion privilege escalation attacks, six features of the app, namely dangerous permissions of the app, dangerous permissions of the components, component intent communication, sensitive API calls, sensitive data flow acquisition, and dissemination of sensitive data, are obtained as the feature set of attack behavior, and the number of transitions is extracted by using static technology.
- (2) Attack behavior modeling and determination. Based on the extraction of the attack behavior features set, process algebra is used to model app behavior and attack behavior, and weak equivalence is used to determine app behavior.
- (3) Risk measurement. According to the result of the behavior determination, the number of behavior features and the number of transitions, a measurement function is constructed to measure the risk of the single app.

The contributions of this paper are as follows:

- (1) Construction and extraction of the feature set of collusion privilege escalation attacks. After obtaining the attack behavior feature set, the static extraction method is used to extract the behavior feature set and the number of transitions of sensitive data. This is done in order to make up for the lack of collusion privilege escalation attack features in the existing app measurement methods.
- (2) Modeling and determination of app behavior using process algebra. The behavior of the app is modeled and determined based on semantics and the equivalence concept of process algebra, and any app that is weakly equivalent to attack behavior is measured. In view of the particularity of collusion attacks, this makes up for the inaccurate measurement results caused by the existing measurement methods that do not distinguish the equivalence relationship of test objects.

- (3) Construction of the measurement function and test set experiment. Based on the number of features in the feature set and the number of transitions, the measurement function is constructed. The case, benchmarks test and test set experiments are completed by using the measurement function. The experimental results show that the method is feasible and effective.

2. Related Work

For the detection of and protection against malware in Android systems, researchers have put forward many classical solutions, from the initial signature-based detection method to the behavior-based dynamic and static detection methods, and then to the data mining and machine learning-based detection methods [10–13]. However, due to their lack of analysis of the features of collusion privilege escalation attacks, these detection methods have a poor detection effect on a single app that can construct collusion privilege escalation attacks. Based on these previous achievements, researchers have carried out in-depth analysis on privilege escalation attacks on Android apps, and produced remarkable research results. Kernel-level privilege escalation attacks are detected by stain tracking and monitoring permission information [14,15], while for application-level privilege escalation attacks, references [16,17] respectively detect and protect against application-level privilege escalation attacks that function by Android security modules (ASM) access control and monitoring of important system calls. However, application-layer collusion privilege escalation attacks occur on the basis of the collusion of multiple independent and secure apps, so it is particularly important to detect the dangerous path between apps [9,16]. However, the existing detection methods have shortcomings in the following two aspects: (1) their lack of detection of inter-application communication that can construct privilege escalation attacks; (2) their lack of analysis and detection of the features of collusion privilege escalation attacks. With the continuous improvement of Android security mechanisms, the cost of inter-application communication detection and multi-feature detection is increasing. Therefore, the measurement of application-layer collusion privilege escalation attacks has attracted the attention of researchers.

Meanwhile, researchers have made many achievements in the field of software and app security trust measurement. Zhao Qian et al. [18] used pi-calculus, an important branch of process algebra, to measure software dependability by comparing real running results with expected results, so as to conduct quantitative research on software credibility. Although this method uses runtime in behavior modeling, it can reflect the credibility of real-time software changes. However, for collusion privilege escalation attacks, it can disguise the attack result as the expected target and rationalize the attack behavior. Based on the trust theory in the field of human sociology, a measurable software trust framework (social to software, S2S) is proposed in reference [19]. From the point of view of software capability, it gives an overall measurement solution for software trustworthiness. However, as for a single app that constitutes collusion privilege escalation attacks, the app is secure and non-threatening when it alone running, which will make the measurement method invalid. In reference [20], evidence is considered as the basic element for establishing software trustworthiness and supporting a trustworthiness evaluation, and a software process trustworthiness model is proposed based on process assurance. Although the establishment of this method runs through the whole development process, and its considerations are more comprehensive, it is difficult to measure collusion privilege escalation attack by this method in cases where developers have designed an app as part of a collusion privilege escalation attack from the beginning. Reference [21] introduces a risk assessment method based on an analysis of the relationship between permission and function, and it is found that standardizing permission application by Android developers can provide reasonable permission configuration to users in real time and reduce the risk of permission abuse. However, this method only considers the risk of using permissions in the implementation of functions, but fails to consider other attack features. Xu Junfeng et al. [22] proposed a credit index measurement method for Android app security based on analytic hierarchy process (AHP) combined with the Android software certification strength and the violation records in the third-party application market. This method considers the

violation records of third-parties. Therefore, the credibility evaluation is more objective. However, the concealment of the attack associated with collusion privilege escalation attacks makes it difficult for the third-party's violation record to mark the app's violation in time. Li Danjun et al. [23] proposed an application risk assessment method based on behavioral analysis by using a sandbox approach to dynamically monitor and record the behavior of applications on Android. However, this method only considers the system's API calls, not other attack features. Reference [24] proposed a criterion to measure the security risks of the apps based on analyzing requested permissions of large numbers of malwares and benign apps. This method used the concepts of entropy given that more informative permissions have higher impacts on the computed risk values. Although this method considers the scale of information in permissions, the permissions for collusion attack may not have more information.

Therefore, previous research results on software and app credibility measurement have been achieved. However, due to the strong concealment and camouflage of application-layer collusion privilege escalation attacks, the detection effect of existing software measurement methods for collusion privilege escalation attacks is not good. The main problems are as follows:

- (1) Up to now, there has been no method to measure the credibility of a single app that constructs collusion privilege escalation attacks.
- (2) The existing methods do not consider the feature set of attack behavior when measuring the credibility of a single app, and they only focus on one feature.

Therefore, it is very important to measure the risk of a single app that constructs collusion privilege escalation attacks. In view of this, the present paper proposes a method for app risk measurement using process algebra.

3. Background Information

In order to better describe and understand this measurement method, some background concepts are explained.

- (1) Definition and classification of privilege escalation attacks on android app.

Privilege escalation attacks means that an application with lower (less) permissions can access components with higher (more) permissions without being restricted by tasks. That is, a malicious program without any permission can obtain the required permission through a third-party app.

This attack can be divided into two categories: kernel-level and application-level. There are two kinds of application-layer privilege escalation attacks: confused deputy attacks and collusion privilege escalation attacks. Our study is focused on the application-layer collusion privilege escalation attack.

- (2) Definition of application-layer privilege escalation.

The privilege escalation attack occurs in the application-layer of Android architecture. Android apps access components with more permissions than themselves through third-party apps and intent communication.

- (3) Definition of confused deputy attacks.

This is a kind of application-layer privilege escalation attack. It exploits privileged app interface vulnerability and exploits the interface vulnerability. One component of the app has applied for dangerous permissions that it does not need, and there are interfaces that allow other apps to access through intent. By scanning the interface, malware can use its implementation to confuse agent attacks.

- (4) Definition of collusion privilege escalation attacks.

This is a kind of application-layer privilege escalation attack. It takes place on the basis of collusion of multiple apps, and completes the attack by gradually upgrading permission. App A obtains sensitive information but does not apply for permission P, but component comA of A has the ability to access the component comB of App B. At this time, B applies for P. ComB has the ability to access the component comC of App C, and comC has applies for P. Therefore, from comA does not apply for P to comA completes access to comC with using comB and P, which constitutes collusion privilege escalation attacks, and completing sensitive information leaks and other attacks.

(5) Definition of privilege escalation vulnerabilities.

It has the threat proposed in privilege escalation attacks. One component of the app accesses a component of another app with permissions through intent communication. At this time, Intent communication is a vulnerability that may constitute a privilege escalation attacks.

4. Construction and Extraction of the Attack Behavior Feature Set

4.1. Construction of the Attack Behavior Feature Set

Figure 1 describes a privilege escalation attack based on multi-app collusion [25]. AppA and AppC do not apply for permission P1, while AppB applies for P1, and component ComC₁ applies for P1; therefore, component ComA₁ is not protected by P1, while components ComB₁ and ComC₁ are protected by P1. As can be seen in Figure 1:

- (1) ComA₁ can access ComB₁ that with the protection of P1.
- (2) ComB₁ and ComC₁ have the same permission P1, and ComB₁ can access ComC₁.
- (3) ComA₁ accesses ComC₁ through ComB₁ by (1) and (2), and AppA, AppB, and AppC escalate permissions on P1.

Figure 1. Diagram of a privilege escalation attack.

Therefore, based on the analysis of Figure 1, we can obtain the behavior feature set F of the privilege escalation attack as

$$F = \{F1, F2, F3, F4, F5, F6\} \quad (1)$$

where:

- (1) F1 indicates the dangerous permissions of the application.
- (2) F2 indicates the dangerous permissions of the components.
- (3) F3 indicates the component intent communication.
- (4) F4 indicates the sensitive API calls.
- (5) F5 indicates the sensitive data flow acquisition.
- (6) F6 indicates the dissemination of sensitive data.

4.2. Extraction of the Attack Behavior Feature Set

For Android application package (APK), APKTool is used to convert APK into a Smali file for analysis. The AndroidManifest.xml file is the information description file of the application, which defines the permissions, components, intent filter and other information contained in the application. The behavior feature set is extracted from Smali and AndroidManifest.xml.

The latest dangerous permissions list is obtained from Google official documents, and permissions disabled after version 5.0. PScout [26] summarizes the relationship between the permissions and API calls of multiple versions of Android, and builds sensitive API sets based on the dangerous permissions list.

1. Permissions of the application

The application permissions are extracted according to the <uses-permission> tag in the AndroidManifest.xml file.

2. Permissions of components

- (1) The component information is extracted according to the <activity></activity> tag in the AndroidManifest.xml file.
- (2) The permissions of components are extracted on the basis of (1).

3. Component Intent communication

Based on the extraction of the component information, the special function calls (getStringExtra, putExtra, etc.) in the Smali file are used to extract the information on Intent communication.

4. Sensitive API calls

The system call sequence is extracted by the Android Software Development Kit's (SDK's) strace, and the sensitive API calls are obtained by combining the sensitive API set.

5. Sensitive data flow acquisition

FlowDroid [27] is used to extract sensitive data flow pairs, that is, <source, sink>, where source is the information's source and sink is the leakage point.

6. Dissemination of sensitive data

FlowDroid [27] is used to detect the data dissemination path from source to sink.

7. Applying for the dangerous permissions of the application and components

On the basis of 1 and 2, the dangerous permissions applied by the app are obtained according to the dangerous permissions list.

4.3. Attack Case

According to the attack model in Section 4.1, we designed three apps to constitute a privilege escalation attack [28], and the key code is shown in Figure 2.


```

<application..... >
    <activity
        android:name=".inputInfromShowActivity"
        android:label="show input">
    </activity>
</application>

Intent intentAppA=new Intent("sensitiveInfo");
Bundle bundleAppA =new Bundle();
bundleAppA.putString("userName",editName.getText().toString());
bundleAppA.putString("userTel",editTel.getText().toString());
intentAppA.putExtras(bundleAppA);
startActivity(intentAppA);

```

(a)

```

<application..... >
    <activity
        android:name="com.example.newsshow.sendNewsToFriend".....>
        <intent-filter>
            <action android:name="sensitiveInfo"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </activity>
</application>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.SEND_SMS"/>...

Bundle bundleB=this.getIntent().getExtras();
keyStr=bundleB.getString("userName")+","+budle.getString("userTel");

Intent intentAppB=new Intent("sensitiveInfoSend");
Bundle bundleAppB=new Bundle();
bundleAppB.putString("keyInfo",keyStr);
intentAppB.putExtras(bundleAppB);
startActivity(intentAppB);

```

(b)

```

<application.....>
    <activity
        android:name="com.example.healthmonitorlh.sendMessage".....
        android:permission="android.permission.SEND_SMS">
        <intent-filter>
            <action android:name="sensitiveInfoSend"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </activity>
</application>
<uses-permission android:name="android.permission.INTERNET"/>

Bundle bundleC=this.getIntent().getExtras();
sendStr=bundleC.getString("keyInfo");

String phoneNumber = "XXXXXXXXXXXX";
String message = sendStr;
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNumber, null, message, null, null);

```

(c)

Figure 2. (a) comAppA key code; (b) comAppB key code; (c) comAppC key code.

According to the above code:

- (1) comAppA does not apply for any dangerous permission. Its component *inputInfromShowActivity* obtains the user's private information (username and password) through edittext, and sends it to the component whose <intent-filter> is **sensitiveInfo** through the intentAppA.
- (2) comAppB has applied for the dangerous permission android.permission.SEND_SMS, and its component *sendNewsToFriend* has <intent-filter> as **sensitiveInfo**. The user's private information (username and password) is obtained by bundleB and is sent to the component whose <intent-filter> is **sensitiveInfoSend** through intentAppB.
- (3) comAppC has not applied for any dangerous permission, but its component *sendMessage* has applied for android.permission.SEND_SMS dangerous permission and has <intent-filter> as **sensitiveInfoSend**. Using bundleC, the username and password are obtained and sent to the mobile phone number by SMS.

5. Attack Behavior Modeling Based on Process Algebra

5.1. Process Algebra Based on Behavior Features

Process algebra [29] can effectively describe the Android architecture and message communication characteristics, and it can be used in the creation of an application attack behavior model according to the attack behavior feature set. The following is the syntactic and semantic specification of process algebra based on behavior features.

$$\begin{aligned} \text{appComBehavior} ::= \\ = \sum_{\substack{i \in I \\ j \in K}} P_i \cdot a_j | \text{Feature}_1 | \text{Feature}_2 | \cdots | \text{Feature}_w | (\bar{X}(y_1, y_2, \dots, y_n) | X(y_1, y_2, \dots, y_n)) | (\text{action})P \quad (2) \end{aligned}$$

where:

- (1) $\sum_{j \in K} P_i \cdot a_j$ is a summation, where I and K are any finite indexing set. a_j is protected by P_i , because a_j must start activities after the action represented by P_i occurs. It represents a collection of behaviors of a component of an app under permission.
- (2) $\text{Feature}_1 | \text{Feature}_2 | \cdots | \text{Feature}_w$ represents that the component has w features at the same time.
- (3) $(\bar{X}(y_1, y_2, \dots, y_n) | X(y_1, y_2, \dots, y_n))$, where $\bar{X}(y_1, y_2, \dots, y_n)$ represents the action of message transmission; $X(y_1, y_2, \dots, y_n)$ represents the action of message receipt; $y_1, y_2, \dots, y_n$ represents n sensitive data.
- (4) $(\text{action})P$ represents that the action is under the protection of permission P .

5.2. Attack Behavior Modeling

In the Android architecture, apps are made up of several services and components. A component is the basic unit of an app. In the case of a privilege escalation attack, a component with dangerous behavior must complete the collection, transmission and sending of sensitive information under the protection of dangerous permission. The four components of Android are as follows:

- (1) Activity, which is used to express the function.
- (2) Service, which runs in the background and does not provide interface presentation.
- (3) BroadcastReceiver, which is used to receive broadcast information.
- (4) ContentProvider, which supports data storage and reading across multiple applications and is equivalent to a database.

Therefore, the dangerous behavior of a privilege escalation attack must be carried out by a component, and the behavior of the application components constitutes the behavior of the application. According to Formulas (1) and (2), the definition of component attack behavior under P_1 can be obtained, as shown in Formula (3).

$$\text{attackAction} ::= \sum_{i \in 6} P_1 \cdot a_i | F_1 | F_2 | \cdots | F_6 | (\bar{X}(y_1, y_2, \dots, y_n) | X(y_1, y_2, \dots, y_w)) | (X)P \quad (3)$$

where:

- (1) $\sum_{i \in 6} P_1 \cdot a_i$ represents the attack behavior set under P_1 .
- (2) $F_1 | F_2 | \cdots | F_6$ represents the six features of a component.
- (3) $(\bar{X}(y_1, y_2, \dots, y_n) | X(y_1, y_2, \dots, y_w))$, where $\bar{X}(y_1, y_2, \dots, y_n)$ represents the action of message transmission and $X(y_1, y_2, \dots, y_w)$ represents the action of message receipt. $y_1, y_2, \dots, y_n$ represents n sensitive data.

- (4) $(\chi)P$ represents the behavior χ of an application is protected by permission P .

6. Behavior Determination and Risk Measurement Based on Process Algebra

According to the state relations of strong equivalence, weak equivalence and non-equivalence defined in process algebra, we use the concept of weak equivalence to determine the dangerous behavior and then combine the risk measurement function to measure the risk.

Definition 1. Labelled transition system (LTS). Suppose the action of application is under the protection of permission P , and the set of app actions $Act = \{a_1, a_2, \dots, a_t, a'_1, a'_2, \dots, a'_t\}$ is a pair (Q, T) which is the LTS, where $Q = \{(a_1, P), (a_2, P), \dots, (a_t, P), (a'_1, P), (a'_2, P), \dots, (a'_t, P)\}$ is a set of states; $T \subseteq (Q \times Act \times Q)$ is a ternary relation, known as a transition relation. If $\forall (a_i, P) \in Q, \forall (a'_i, P) \in Q, \exists a_i \in Act$ (or $\exists a'_i \in Act$) and $((a_i, P), a_i, (a'_i, P)) \in T$ (or $((a_i, P), a'_i, (a'_i, P)) \in T$), we written $(a_i, P) \xrightarrow{a_i} (a'_i, P)$ (or $(a_i, P) \xrightarrow{a'_i} (a'_i, P)$).

Definition 2. Weak simulation. Suppose $S = \{((a_1, P), (a'_1, P)), ((a_2, P), (a'_2, P)), \dots, ((a_t, P), (a'_t, P))\}$ is a binary relation in the state space $\mathcal{P}$ of a component, and $P, P', Q, Q' \in \mathcal{P}$, $\xrightarrow{\lambda}$ represents any action by a feature of a component. S is a weak simulation if and only if, whenever PSQ , if $P \xrightarrow{\lambda} P'$, then there exists Q' such that QQ' and $P'SQ'$; if $P \xrightarrow{\lambda} P'$ then there exists Q' such that $Q \xrightarrow{\lambda} Q'$ and $P'SQ'$. This means that state P has passed a finite number of state transitions to state Q . An interaction action of P can be corresponding to a series of interaction actions of Q , or even no action.

Definition 3. Behavior weak equivalence. Based on Definitions 1 and 2, 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 there is a certain equivalence relationship between the actions of the component and the dangerous behavior model, which needs to be measured by the measurement function.

Definition 4. Behavior non-equivalence. Based on Definitions 1, 2, and 3, P and Q are the LTS. Whenever P chooses any transition, Q cannot find an appropriate transition to match. That is, if there is no strong (weak) simulation R , then P and Q are not equivalent, which is written as $P \neq Q$. This means that there is no equivalent relationship between the actions of this component and the dangerous behavior model, and the risk measure is 0.

Definition 5. Sensitive data transition. There exists a set $Act = \{a_0, a_1, \dots\}$ of actions, a state set $Q = \{q_0, q_1, \dots\}$ of states, and a subset T of $Q \times Act \times Q$ called the transitions. That is, $q, q' \in Q, a \in Act$, then $q \times a \times q'$, called q transition q' . We use FlowDroid to obtain the path of stain dissemination, so as to obtain the number of sensitive data transitions.

Definition 6. Risk measurement function. In the case of extracting the behavior feature set and sensitive data transitions, we construct the measurement function $f(x, y)$.

$$f(x, y) = \begin{cases} 1 - \frac{1}{e^{x+y}}, & P \approx Q \\ 0, & P \neq Q \end{cases} \quad (4)$$

where x represents the number of dangerous features in $F(0 \leq X \leq 6)$, and y represents the number of transitions $(0 \leq y \leq \infty)$.

- (1) When x tends to be 6 and Y tends to be infinite, then $\lim_{x \rightarrow 6, y \rightarrow \infty} \frac{1}{e^{x+y}} = 0$. That is, when the more dangerous features and transition times; $f(x, y)$ are closer to 1, the risk is greater.
- (2) When x tends to be 0 and Y tends to be 0, then $\lim_{x \rightarrow 0, y \rightarrow 0} \frac{1}{e^{x+y}} = 1$. That is, when the less dangerous features and transition times; $f(x, y)$ are closer to 0, the risk is smaller.

Algorithm 1. Here, riskcom is a set of components with dangerous behavior features.

Algorithm 1. Risk measurement algorithm based on process algebra.

```

1: Input: riskCom, F, attackAction, X, Y, f(x,y)
2: Output: measureValue
3: Assumption: use PR point to riskCom = {Com0, Com1, ...}
4: Initialization: PR → riskCom0,
5: For each component in riskCom
6:   Construction attackActionF according to F
7: If attackActionF ≈ attackAction Then
8:   call f(x,y)
9:   print measureValue
10: PR = PR → next
11: ElseIf PA ≠ PC Then
12:   print measureValue=0
13:   PR = PR → next
14: EndIf

```

7. Experiment of an Attack Case

7.1. Feature Set Extraction of Attack Behavior

Using Equation (1), the behavior feature sets of comAppA, comAppB, and comAppC can be obtained respectively, as shown in Table 1. Because the code is tedious, in order to save space, the behavior feature is not represented directly by code, but it is represented by T and F, where T means that the feature exists, and F means that the feature does not exist.

Table 1. Feature sets of comAppA, comAppB and comAppC.

	F1	F2	F3	F4	F5	F6
comAppA	F	F	T	F	T	T
comAppB	T	T	T	T	T	T
comAppC	F	T	T	T	T	T

From Table 1, it is shown that the number of Fs for comAppA, comAppB, and comAppC is 3, 6, and 5, respectively.

7.2. Attack Behavior Modeling

For the case of the privilege escalation attack given in Section 4.3, the attack behavior models of comAppA, comAppB, and comAppC can be obtained according to Table 1 and Equation (3).

- (1) The attack behavior of comAppA is modeled as shown in Equation (5).

$$\text{comAppAAttack} ::= (\bar{X}\langle y_1 \rangle | X(y_1)) \Big|_{F6} \quad (5)$$

- (2) The attack behavior of comAppB is modeled as shown in Equation (6).

$$\begin{aligned} \text{comAppBAttack} &::= \sum_{i=2}^6 P_1 \cdot F_i \Big|_{(\bar{X}\langle y_1, y_2 \rangle | X(y_1, y_2))} \Big|_{(vX)P_1} \\ &::= P_1 \cdot F_2 + P_1 \cdot F_3 + P_1 \cdot F_4 + P_1 \cdot F_5 + P_1 \cdot F_6 \Big|_{(x(y_1) | \bar{X}\langle y_1 \rangle)} \cdot P_1 \Big|_{P_1} \end{aligned} \quad (6)$$

- (3) The attack behavior of comAppC is modeled as shown in Equation (7).

$$\begin{aligned} \text{comAppCAttack} &::= \sum_{j=2}^6 P_1 \cdot F_j \left(\left(\overline{X} \langle y_1, y_2 \rangle \right) \left| X \langle y_1, y_2 \rangle \right| \right) (\nu X) P_1 \\ &::= P_1 \cdot F_2 + P_1 \cdot F_3 + P_1 \cdot F_4 + P_1 \cdot F_5 + P_1 \cdot F_6 \left(\left(x(y_1) \right) \left| \overline{X} \langle y_1 \rangle \right| \right) \cdot P_1 \end{aligned} \quad (7)$$

7.3. AppBehavior Modeling

The behavior models of comAppA, comAppB, and comAppC can be obtained according to Table 1 and Equation (2).

- (1) The behavior of comAppA is modeled as shown in Equation (8).

$$\text{comAppABehavior} ::= \text{filterIntent} \left(\left(X(\text{sensitiveInfo}) \right) \left| \overline{X} \langle \text{sensitiveInfo} \rangle \right| \right) \text{sourceSink} \quad (8)$$

- (2) The behavior of comAppB is modeled as shown in Equation (9).

$$\begin{aligned} \text{comAppBBehavior} &:: \\ &= P_1 \cdot \text{comPermission} + P_1 \cdot \text{Intent} + P_1 \cdot \text{sensitiveAPI} + P_1 \cdot \text{sourceSink} \\ &\quad + P_1 \cdot \text{filterIntent} \left(\left(X(\text{sensitiveInfo}) \right) \left| \overline{X} \langle \text{sensitiveInfo} \rangle \right| \right) \cdot P_1 \end{aligned} \quad (9)$$

- (3) The behavior of comAppC is modeled as shown in Equation (10).

$$\begin{aligned} \text{comAppCBehavior} &:: \\ &= P_1 \cdot \text{comPermission} + P_1 \cdot \text{Intent} + P_1 \cdot \text{sensitiveAPI} + P_1 \cdot \text{sourceSink} \\ &\quad + P_1 \cdot \text{filterIntent} \left(\left(X(\text{sensitiveInfo}) \right) \left| \overline{X} \langle \text{sensitiveInfo} \rangle \right| \right) \cdot P_1 \end{aligned} \quad (10)$$

- (4) To expand the case, assuming that the behavior component of comAppA has no Intent communication, the behavior model of comAppA is shown in Equation (11).

$$\text{comAppAChange} ::= X(\text{sensitiveInfo1}) \left| \overline{X} \langle \text{sensitiveInfo1} \rangle \right| \quad (11)$$

7.4. BehaviorDetermination

App has a feature then it has a state when the feature exists. For example, App has permission P, and then its component is protected by P. The privilege escalation attacks must be a part of the action set of the application itself. Therefore, according to the Definition (1), the status and behavior of app can build LTS, and the status of Equations (6) and (9) must exist in the LTS. For the convenience of description, Tables 2 and 3 are used to simplify the states represented by related features.

Table 2. Symbol simplification of Equation (6).

	Formula Content	Simplified Symbols
dangerous permissions of the components	$P_1(F_2)$	p_0
sensitive API calls	F_4	p_1
component intent communication	F_3	p_2
sensitive data flow acquisition	F_5	p_3
dissemination of sensitive data	F_6	p_4

Table 3. Symbol simplification of Equation (9).

	Formula Content	Simplified Symbols
dangerous permissions of the application	P_1	q_0
dangerous permissions of the components	comPermission	q_1
sensitive API calls	sensitiveAPI	q'_1
component intent communication	Intent	q_2
sensitive data flow acquisition	sourceSink	q_3
dissemination of sensitive data	filterIntent	q_4

Construct the state transition diagram of Equations (6) and (9), as shown in Figure 3a,b.

Figure 3. (a) State transition diagram of Equation (6); (b) State transition diagram of Equation (9).

According to the Definition (2), suppose that $S = \{(q_0, p_0), (q_1, p_1), (q'_1, p_1), (q_2, p_2), (q_3, p_3), (q_4, p_4)\}$.

- (1) p_0 weak simulation q_0 verification. $\forall (q, p) \in S$, each migration of the first element q , $q_i \xrightarrow{a} q_j$, $0 \leq i, j \leq 4$ can be migrated $p_i \xrightarrow{a} p_j$, $0 \leq i, j \leq 4$ by a second element p matching (or a series of migration, or even no migration). For example, for (q'_1, p_1) , q'_1 has $q'_1 \xrightarrow{c} q_3$, so $\exists (q_3, p_3) \in S$, using $p_1 \xrightarrow{c} p_3$ matching. Therefore, s is a weak simulation, that is, p_0 weak simulation q_0 .
- (2) In the same manner, the weak simulation relationship of each state in two state graphs can be verified.
- (3) According to the concept of weak equivalence Definition 3, S^{-1} can be verified as a weak simulation in the way of (1).
- (4) Therefore, the weak equivalence between the states in Figure 3a,b can be verified. Then, the weak equivalence between the Equations (6) and (9) is verified.

To make the verification more automated, the mobility workbench (MWB) was chosen as the verification tool. Standardize Equations (6) and (9) according to MWB syntax [30]. For example, the output $\bar{a}$ in process algebra should be converted to 'a. Therefore, $\bar{x}\langle y_1 \rangle$ in Equation (6) is convert to MWB syntax is ' $x\langle y_1 \rangle$ '. In order to better distinguish the input and output, we use ' $y\langle y_1 \rangle$ ' to represent the output. The comAppBBehavior in Equation (9) is represented by agent BB when it is encoded by MWB, and the comAppBAttack in Equation (6) is represented by agent BA when it is encoded by MWB. Agent BB1X and BA1X represent the components of BB and BA respectively. Weq and eq are used to verify the weak equivalence and strong equivalence of the two models. Therefore, the weak equivalence relationship between the component behavior model (BB) and the component attack behavior model (BA) can be verified, as shown in Figure 4a. At the same time, the comAppAChange in Equation (11) is represented by AC when it is encoded by MWB, and the comAppAAttack in Equation (5) is represented by AA when it is encoded by MWB. According to the syntax rules of MWB and the Definition 4, the non-equivalence relationship between the component model (AC) with changed features and the component attack model (AA) can be obtained, as shown in Figure 4b.

```

MWB>agent BB10(x,y)=x(sensitiveInfo).y<sensitiveInfo>.BB10(x,y)
MWB>agent BA10(x,y)=x(y1).y<y1>.BA10(x,y)
MWB>weq BB10 BA10
The two agents are equal.
Bisimulation relation size = 2.
MWB>agent BB11(x,y)=(p1) (BB10(x,p1) | BB10(p1,y))
MWB>agent BA11(x,y)=(p1) (BA10(x,p1) | BA10(p1,y))
MWB>weq BB11 BA11
The two agents are equal.
Bisimulation relation size = 13.
MWB>agent BB12(x,y)=(comper) (BB10(x,comper) | BB10(comper,y))
MWB>agent BA12(x,y)=(f2) (BA10(x,f2) | BA10(f2,y))
MWB>weq BB12 BA12
The two agents are equal.
Bisimulation relation size = 13.
MWB>agent BB13(x,y)=(intent) (BB10(x,intent) | BB10(intent,y))
MWB>agent BA13(x,y)=(f3) (BA10(x,f3) | BA10(f3,y))
MWB>weq BB13 BA13
The two agents are equal.
Bisimulation relation size = 13.
MWB>agent BB14(x,y)=(sensitiveapi) (BB10(x,sensitiveapi) | BB10(sensitiveapi,y))
MWB>agent BA14(x,y)=(f4) (BA10(x,f4) | BA10(f4,y))
MWB>weq BB14 BA14
The two agents are equal.
Bisimulation relation size = 13.
MWB>agent BB15(x,y)=(sourcesink) (BB10(x,sourcesink) | BB10(sourcesink,y))
MWB>agent BA15(x,y)=(f5) (BA10(x,f5) | BA10(f5,y))
MWB>weq BB15 BA15
The two agents are equal.
Bisimulation relation size = 13.
MWB>agent BB16(x,y)=(filterintent) (BB10(x,filterintent) | BB10(filterintent,y))
MWB>agent BA16(x,y)=(f6) (BA10(x,f6) | BA10(f6,y))
MWB>weq BB16 BA16
The two agents are equal.
Bisimulation relation size = 13.

```

(a)

```

MWB>agent AC10(x,y)=x(sensitiveInfo).y<sensitiveInfo>.AC10(x,y)
MWB>agent AC11(x,y)=(f6) (AC10(x,f6) | AC10(f6,y))
MWB>agent AA(x,y)=x(sensitiveInfo).y<sensitiveInfo>.AA(x,y)
MWB>eq AC11 AA
The two agents are NOT equal.
MWB>weq AC11 AA
The two agents are NOT equal.

```

(b)

Figure 4. (a) Weak equivalence relationship verified; (b) Non-equivalence relationship verified.

As can be seen from Figure 4, MWB can be used to verify the weak equivalence relations of Equations (5) and (8), as well as Equations (7) and (10) respectively, that is, it can be verified that comAppA, comAppB, and comAppC are weakly equivalent to the attack model.

7.5. Risk Measurement

According to Definition 5, the number of sensitive data transitions of comAppA, comAppB, and comAppC can be obtained, as shown in Table 4.

Table 4. Number of transitions of comAppA, comAppB and comAppC.

	comAppA	comAppB	comAppC
number of transitions	1	2	1

Using Equation (4) and the risk measurement algorithm 1 based on process algebra, measurement values for the three apps can be obtained, as shown in Table 5.

Table 5. Measurement values.

	X	Y	Measurement Value
comAppA	3	1	0.9816843611
comAppB	6	2	0.9996645373
comAppC	5	1	0.9975212478

From the measurement results in Table 5, we can see that the measurement values of comAppA, comAppB and comAppC are close to 1. The measurement values of comAppAChange, for which the relevant features have been changed, is 0. These results are consistent with the actual app's dangerous behaviors, so this shows the effectiveness of our method.

8. Method Evaluation and Effectiveness Analysis

8.1. Method Evaluation

In this paper, APKTool and FlowDroid are used to extract the attack behavior feature set. MWB is used to verify the equivalence of behavior. Java language is used to implement the risk measurement algorithm. All experiments were carried out on a machine with 8G memory and Intel (R) core (TM) i5-2520M CPU.

The key steps in this method are attack behavior feature set extraction and transition detection, and the time complexity of the dangerous behavior determination and measurement algorithm is $O(n)$. In the experiment on the test set, due to the different of feature set and the number and path of transition, the costs of time and space are discretely distributed, as shown in Figure 5.

Figure 5. (a) Time cost; (b) Space cost.

This method is discussed in two cases: weak equivalence and non-equivalence. According to measurement algorithms and measurement function, weak equivalence APKs must be risk measured, but non-equivalence APKs risk measurement value is 0. The size of the APK, the number of components, the number of sensitive data transitions and the number of features in the attack behavior feature set affect the time cost and space cost of this method. There are three situations as follows:

- (1) Some of the APKs have some features as small size, fewer components, or fewer sensitive data transitions and attack features. In general, those APKs are non-equivalent to attacks behavior. Those APKs have less time cost and space cost.
- (2) Another part of APKs have the following features as larger size, more components, or a greater number of sensitive data transitions and attack features. In general, those APKs will be weak equivalent to the attack behavior. Those APKs has more time cost and space cost for this method.
- (3) There will be some special APKs. For example, two APKs have similar size, but because the number of components, transitions and features of first APK is far greater than second APK, the time cost and space cost of first APK are also far greater than second APK. However, it will follow the principle that with more components, or the greater number of transitions and features there, the more this method will cost in terms of time and space.

Therefore, according to the analysis of (1) and (2) on the above, the time cost and space cost of this method are prone to the phenomenon of two groups separated, as shown in Figure 5b. According to (3) on the above, due to special circumstances, those two groups are not necessarily completely separated, as shown in Figure 5a.

8.2. Benchmarks Test and Analysis

DroidBench [31] is a set of open source real-life Android applications to be used as a testing ground for static and dynamic security and measurement methods. FieldAndObjectSensitivity, InterAppCommunication, and InterComponentCommunication test sets in DroidBench are measured by the method in this paper. Three test sets cover sensitive data acquisition and propagation, inter-application communication and component communication, which have the attack features of collusion privilege escalation attacks. The methods, shown in Section 4, Section 5, and Section 6, and Equation (4), are used to verify the behavior equivalence and measure the risk. Table 6 shows the F sets, equivalence relations, and measurement results of each test object, where T represents the existence of feature and F represents the no-existence of feature.

Table 6. Benchmarks results.

Test Sets and Test Objects		F						Number of Transition	Equivalence	Measurement Results
		F1	F2	F3	F4	F5	F6			
FieldAndObjectSensitivity	FieldSensitivity1	T	T	F	T	T	T	1	Weak	0.997521248
	FieldSensitivity2	T	T	F	T	T	T	1	Weak	0.997521248
	FieldSensitivity3	T	T	F	T	T	T	1	Weak	0.997521248
	FieldSensitivity4	T	T	F	T	T	T	1	Weak	0.997521248
	ObjectSensitivity1	T	T	F	T	T	T	1	Weak	0.997521248
	ObjectSensitivity2	T	T	F	T	T	T	2	Weak	0.999088118
InterAppCommunication	SendSMS	T	T	T	T	T	T	3	Weak	0.99987659
	StartActivityForResult1	T	T	T	T	T	T	4	Weak	0.9999546
InterComponentCommunication	ActivityCommunication1	T	T	F	T	T	T	1	Weak	0.997521248
	ActivityCommunication2	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication3	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication4	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication5	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication6	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication7	T	T	T	T	T	T	2	Weak	0.999664537
	ActivityCommunication8	T	T	T	T	T	T	2	Weak	0.999664537
	BroadcastTaintAndLeak1	T	T	T	T	T	T	1	Weak	0.999088118
	IntentSink1	T	T	T	T	T	T	1	Weak	0.999088118
	IntentSink2	T	T	T	T	T	T	1	Weak	0.999088118
	IntentSource1	T	T	T	T	F	T	1	Weak	0.997521248
	ServiceCommunication1	T	T	F	T	T	T	1	Weak	0.997521248
	SharedPreferences1	T	T	F	T	T	T	1	Weak	0.997521248

The relationship between the sum of the number of behavior features and transition of the test object and the measurement results is shown in the Figure 6.

Figure 6. Relationship between the number of features and transition and the measurement results.

From the measurement results for 22 objects in Benchmarks, the following two conclusions are obtained:

- (1) The measurement value in Table 6 is between 0.9975–0.9999, which verifies the effectiveness of this method. The three test sets have information leak of sensitive data, communication between apps and communication between components respectively. From the measurement results, it can be seen that app with the risk of such an information leak has a higher risk of privilege escalation attacks, which is consistent with the actual situation.
- (2) Figure 6 shows that the sum of the number of behavior features and transition is directly proportional to the measurement results. The necessity of building F and extracting migration times is verified.

8.3. Test Set Test and Analysis

8.3.1. Composition of the Test Set

The test set is composed of 51 apps from the Android application market such as Google Play and 3 apps developed by the research team [32]. In order to enrich the test set, 22 categories are selected; and one to three typical apps are selected for each category. The specific classification and number of apps are shown in Table 7.

Table 7. Categories of the app test set.

Category	Working	Daily Life	Shopping	Home Control	Medical Treatment	Finance	Examination	Browser	Tourism	Beauty	Social Networks
number	3	3	3	2	3	3	3	1	2	1	3
category	Picture browsing	Reading	System tools	News	Home-based elderly care	Study	Exercise	Wallpaper	Plug-in unit	Entertainment	Research group
number	3	1	3	3	3	3	3	1	2	2	3

The selection of each typical application is as follows: (1) among the same type of app, it should be ranked as high as possible; (2) the APK should be as small as possible. Following the secondary principle, the size of APKs in the test set is shown in Figure 7. The smallest APK's size is 0.4 M, the largest APK's size is 22.5 M, and the average size is 5.33 M.

Figure 7. The size composition of the app in the test set.

Meanwhile, apps from the same developer have the convenience of having collusion privilege escalation attacks deliberately created for them, and the same developer is prone to generate the same software vulnerabilities in the process of app development. Therefore, in the selection of samples, six developers and 16 apps are selected, as shown in Table 8.

Table 8. Percentage of app with the same developer.

	Same Developer	Different Developer
Number	16	38
Percentage	29.6%	70.4%

For the test set, the detection method in reference [33] by our research group is used for detection, which is divided into three categories: constituent attack apps, hidden danger apps, and non-dangerous apps. The specific detection results are shown in Table 9.

Table 9. App attack classification.

	Constituent Attack App	Hidden Dangerous App	Non-Dangerous App
Number	21	19	14
Percentage	38.9%	35.2%	25.9%

According to Table 9, unsafe apps account for 74.1% of the test set.

8.3.2. Measurement Results and Analysis

We use the method in Section 4 to extract the feature set F of the test set, and part of the extracted data is shown in Table 10.

Table 10. Part of the feature set extraction.

Package Name	F1	F2	F3	F4	F5	F6	Transition Number
gjhs.kaoshi.namespace	T	T	T	T	T	T	110
com.hzxh.likerunning	F	F	T	F	T	T	9
com.example.healthmonitor	T	F	F	T	T	T	3
com.vpubao.zhiyue	T	F	T	T	F	T	0
com.gmail.barry2015.android.easysearch_news_cn	F	F	F	F	F	F	0
...	...	...	...	...	...	...	...

Using the methods in Sections 5 and 6, the test set can be divided into weak equivalent app and non-equivalent app, as shown in Table 11.

Table 11. Decision result of the equivalence relation.

	Weak Equivalence App	Non-Equivalence App
Number	40	14
Percentage	74.1%	25.9%

According to the data in Tables 9 and 11, the attack behavior determination result by our method is consistent with the detection result by reference [33].

To measure the test set, the measurement algorithm is used, and the app determination results and measurement values from Table 10 are shown in Table 12.

Table 12. Part of the measurement values.

Package Name	Number of Features in the Feature Set	Transition Number	Decision Result	Measurement Value
gjhs.kaoshi.namespace	6	110	weak equivalence	1
com.hzxh.likerunning	3	9	weak equivalence	0.9999938558
com.example.healthmonitor	4	3	weak equivalence	0.9990881180
com.vpubao.zhiyue	4	0	weak equivalence	0.9816843611
com.gmail.barry2015.android.easysearch_news_cn	0	0	Non-equivalence	0
...	...	...	...	...

The measurement values of test set are shown in Figure 8a, and the curve values of 40 weakly equivalent apps are shown in Figure 8b.

Figure 8. (a) Test set measurement results; (b) Weak equivalence app measurement results.

In Table 8, 13 apps developed by the same developer were verified as a weak equivalent with the attack model. The measurement results are shown in Figure 9.

Figure 9. Measurement results for app developed by the same developer.

From the measurement results in Figures 8 and 9, the following three conclusions are obtained:

- (1) The measurement results clearly show the risk degree of the attack. From Figure 8b, it can be seen that the measurement values of weakly equivalent apps with the collusion privilege escalation attack model are all above 0.86, which is consistent with the actual danger level of the app.

- (2) The measurement value is proportional to the number of features in the feature set and the number of transitions of sensitive data. As can be seen from Figures 8b and 9, the more features and sensitive data transition times, the greater the risk measurement. However, in Figure 8a, there are some particularities, so we need to further strengthen the analysis and refine the feature set and the measurement function.
- (3) The risk of collusion privilege escalation attacks is higher in an app developed by the same developer. From Figure 9, it can be seen that the measurement values of weakly equivalent apps with the attack model are all above 0.95, which proves that the risk of collusion privilege escalation attacks is high.

9. Conclusions

Process algebra was used to model a component's behavior and attack behavior based on the extracted behavior feature set and the number of transitions. Then, weak equivalence was used for decision making. Finally, a risk measurement was carried out based on the measurement function. Through the case, benchmarks and test set experiments, it is verified that the measurement results of this method can clearly show the risk degree of collusion privilege escalation attacks for an app. From the research in this paper, it can be seen that if an app has more attack features and sensitive data transition times, then the app has a higher risk measurement value. Furthermore, the risk of collusion privilege escalation attacks for apps that were developed by the same developer is significantly higher than that of other apps. Future work will be carried out in relation to the following two aspects:

- (1) Because there are many apps with measurement values in the range of 0.86–0.99, it is necessary to refine the measurement function and privilege escalation attacks behavior feature set to ensure that the measurement value is more reasonable.
- (2) We used the same weight of features in the feature set, and different weights can be calculated using a correlation method to improve the accuracy of the measurement.

Author Contributions: Conceptualization, H.L.; formal analysis, H.L.; methodology, H.L.; project administration, L.S.; project administration, H.W.; investigation H.W.; software, Y.W., J.F. and Y.J.; writing—original draft, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: 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), and Science and Technology Project of Hebei Province (No. 17210701D).

Acknowledgments: The authors would like to thank the reviewers for their many insightful comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Operating System Market Share WorldWide. Available online: <https://statcounter.com> (accessed on 30 August 2019).
2. 360 Internet Security Center. 2018 Android Malware Special Report. Available online: <https://research.360.cn/2015/reportlist.html?list=1> (accessed on 3 January 2020).
3. Nokia Threat Intelligence Lab. The Coming of Age of IoT Botnets. Available online: <https://onestore.nokia.com/asset/205166> (accessed on 10 November 2019).
4. Xu, Y.; Ren, J.; Wang, G.; Zhang, C.; Yang, J.; Zhang, Y. A Blockchain-based Nonrepudiation Network Computing Service Scheme for Industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *15*, 3632–3641. [CrossRef]
5. Xu, Y.; Zeng, Q.; Wang, G.; Zhang, C.; Ren, J.; Zhang, Y. An Efficient Privacy-Enhanced Attribute-Based Access Control Mechanism. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5556. [CrossRef]
6. Jiang, X.; Mu, D.; Zhang, H. Unix Domain Sockets Applied in Android Malware Should Not Be Ignored. *Information* **2018**, *9*, 54. [CrossRef]
7. Nokia Threat Intelligence Lab. Available online: <https://pages.nokia.com/T003B6-Threat-Intelligence-Report-2019.html> (accessed on 10 November 2019).

8. 360 Internet Security Center. Available online: <https://zt.360.cn/1101061855.php?dtid=1101061451&did=210942656> (accessed on 30 November 2019).
9. Xu, Y.; Wang, G.; Ren, J.; Zhang, Y. An Adaptive and Configurable Protection Framework against Android Privilege Escalation Threats. *Future Gener. Comput. Syst.* **2019**, *92*, 210–224. [\[CrossRef\]](#)
10. Androguard. Available online: <https://androguard.readthedocs.io/en/latest/> (accessed on 30 May 2020).
11. Lee, S.; Ju, D.Y. Assessment of malicious applications using permissions and enhanced user interfaces on Android. In Proceedings of the 11th IEEE International Conference on Intelligence and Security Informatics (IEEE ISI), Seattle, WA, USA, 4–7 January 2013.
12. Zegzhda, P.; Zegzhda, D.; Pavlenko, E.; Dremov, A. Detecting Android application malicious behaviors based on the analysis of control flows and data flows. In Proceedings of the 10th International Conference on Security of Information and Networks (SIN), Jaipur, India, 13–15 October 2017.
13. Amin, A.; Eldessouki, A.; Magdy, M.T.; Abdeen, N.; Hindy, H.; Hegazy, I. AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach. *Information* **2019**, *10*, 326. [\[CrossRef\]](#)
14. Zhou, W.M.; Zhang, Y.Q.; Liu, X.F. POSTER: A new framework against privilege escalation attacks on android. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, Berlin, Germany, 4–8 November 2013; pp. 1411–1413.
15. Yamauchi, T.; Akao, Y.; Yoshitani, R.; Nakamura, Y.; Hashimoto, M. Additional Kernel Observer to Prevent Privilege Escalation Attacks by Focusing on System Call Privilege Changes. In Proceedings of the IEEE Conference on Dependable and Secure Computing (DSC), Kaohsiung, Taiwan, 10–13 December 2018.
16. Heuser, S.; Negro, M.; Pendyala, P.K.; Sadeghi, A.R. DroidAuditor: Forensic Analysis of Application-Layer Privilege Escalation Attacks on Android. In Proceedings of the International Conference on Financial Cryptography and Data Security, Sliema, Malta, 3–7 April 2017; pp. 260–268.
17. Lee, H.T.; Kim, D.; Park, M.; Cho, S.J. Protecting Data on Android Platform against Privilege Escalation Attack. *Int. J. Comput. Math.* **2016**, *93*, 401–414. [\[CrossRef\]](#)
18. Zhao, Q.; Wang, H.Q.; Feng, G.S.; Zhao, J. Measuring method of software dependability based on Pi calculus. *J. Jilin Univ.* **2011**, *41*, 6.
19. Yang, X.; Jabeen, G.; Luo, P.; Zhu, X.L.; Liu, M.H. A Unified Measurement Solution of Software Trustworthiness Based on Social-to-Software Framework. *J. Comput. Sci. Technol.* **2018**, *33*, 603–620. [\[CrossRef\]](#)
20. Wang, D.X.; Wang, Q. Trustworthiness evidence supporting evaluation of software process trustworthiness. *J. Softw.* **2018**, *29*, 3412–3434. (In Chinese)
21. Han, J.J. Risk Evaluation Based on Relationship between Function and Permission for Android App. *Tianjin Univ.* **2016**, 15–30.
22. Xu, J.F.; Wang, J.J.; Zhu, K.L.; Zhang, P.H.; Ma, Y.F. Credit index measurement method for Android application security based on AHP. *J. Tsinghua Univ.* **2018**, *58*, 2. [\[CrossRef\]](#)
23. Li, Z.J.; Wu, C.M.; Wang, X. Assessment of Android applications risk behavior based on a sand box system. *J. Tsinghua Univ.* **2016**, *56*, 5. [\[CrossRef\]](#)
24. Deypir, M. Entropy-based security risk measurement for Android mobile applications. *Soft Comput.* **2019**, *23*, 7303–7319. [\[CrossRef\]](#)
25. Qing, S.H. Research progress on Android security. *J. Softw.* **2016**, *27*, 45–71. (In Chinese)
26. Au, K.W.Y.; Zhou, Y.F.; Huang, Z.; Lie, D. *PScout: Analyzing the Android Permission Specification*. Proceedings of the 2012 ACM Conference on Computer and Communications Security; ACM: New York, NY, USA, 2012; pp. 217–228.
27. Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Traon, Y.L.; Octeau, D.; McDaniel, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Not.* **2014**, *49*, 259–269. [\[CrossRef\]](#)
28. Attack Case. Available online: <https://pan.baidu.com/s/1haAdhXTDpHvJyHywisbOjQ> (accessed on 5 January 2020).
29. Milner, R. *Communicating and Mobile Systems the Pi-Calculus*; United Kingdom at the University Press: Cambridge, UK, 1999.
30. The Mobility Workbench User’s Guide. Available online: <http://www.it.uu.se/profundis/mwb-dist/guide-3.122.pdf> (accessed on 25 April 2020).

31. DroidBench-Benchmarks. Available online: <https://blogs.uni-paderborn.de/sse/tools/droidbench/> (accessed on 5 April 2020).
32. APK Test Set. Available online: https://pan.baidu.com/s/1m8wC4v_ugbYH_iPPK4hgBA (accessed on 5 January 2020).
33. Li, H.; Shen, L.M.; Ma, C.; Liu, M.Y. Role Behavior Detection Method of Privilege Escalation Attacks for Android Applications. *Int. J. Perform. Eng.* **2019**, *15*, 1631–1641.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).