Compositional Information Flow Verification for Inter Application Communications in Android System

Xue Rao(✉), Ning Xi(✉), Jing Lv(✉), and Pengbin Feng(✉)

Xidian University, Xi'an, China
{xrao,lvj}@stu.xidian.edu.cn, nxi@xidian.edu.cn, pbfeng@outlook.com

Abstract. Inter-component communication (ICC) is commonly used in Android for information exchange among different components/apps. However, it also brings severe challenges to information flow security. When data is transferred and processed, the diversity of different security mechanisms in various apps make data more vulnerable to leakage. Although there are several analysis approaches on security verification on inter-component information flow, repetitive verification on the same component during complex interactions increases the overhead, which would affect task execution efficiency and consume more energy. Therefore, we propose a compositional information flow security verification approach, which improves efficiency by separating the intra-app and inter-app analysis and verification process. The experiment and analysis show that our method is more effective than traditional global approaches.

Keywords: Android system · Information flow model · Inter-Component Communication · Compositional verification

1 Introduction

The current android operating system allows users to run many applications developed by third-party independent developers, which are available in android app markets. In addition, multiple applications can communicate and exchange data by inter-component communication (ICC). ICC is the key mechanism of communication between applications in android, which enriches the functions of android applications, such as WeChat, which can access health data for ranking. Unfortunately, while ICC enhances user functionality, it can be exploited by malicious software to threaten user privacy. Indeed, researchers have shown that android apps frequently collect and use users' private data without their prior consent [17].

When applications communicate with each other, it is more prone to data leakage [1,3]. Existing information flow analysis methods mainly include static analysis, dynamic analysis and machine learning analysis. In addition, there are

X. Chen et al. (Eds.): ML4CS 2019, LNCS 11806, pp. 239–252, 2019.
https://doi.org/10.1007/978-3-030-30619-9_17

some methods that combine the previous methods, such as hybrid (combining dynamic and static) analysis methods.

The static analysis method decompile the .APK file of each app, and then perform static taint analysis on the decompiled code to find out the data leakage path, such as flowDroid [4], IIFDroid [8], DroidSate [10], DroidGuard [14] and so on. They can analyze all the application's resources or codes to achieve high coverage on code. However, they lack an actual execution path and face critical challenges in the presence of code obfuscation [22], loading dynamic code [15], reflection calls [23], native code [24], and multithreading issues. The dynamic analysis method detects the privacy leakage within an application by executing the application in a real or virtual device, such as Mobile-sandbox [16], TaintDroid [6] and ScanDroid [12]. They can observe actual execution trace and tackle code obfuscation and dynamic code loading. However, code coverage is limited by dynamic analysis methods because it cannot execute all possible traces in one time. As a result, the private data leakage vulnerabilities which exist in the uncovered codes will be missed. Moreover, current malware can recognize dynamic monitors as the analyzed app executes, causing the app to pose as a benign program in these situations [25]. In order to solve the challenges of static analysis and dynamic analysis, some hybrid solutions are proposed. For example, HybriDroid [9] present a novel hybrid approach aims to automatically find privacy leakages in a given app set.

In addition, in recent years, with the development of artificial intelligence algorithms, it has become a trend to combine information flow analysis with artificial intelligence. Machine learning is a branch of artificial intelligence mainly treating information flow analysis as a classification problem. By analyzing the differences in features between benign applications and malicious applications, the features with statistical differences are selected, and then trained to classify [26–31]. In the feature extraction stage, static analysis method is generally used to extract features. [32] makes use of the similarity analysis of android application features of multiple dimensions to obtain the relevant rules of multiple dimensions of android application. [33] automatically learns security/privacy-related behaviors by analyzing user comments based on machine learning. In order to extract API data dependencies, [34] conducts context-sensitive, flow-sensitive and inter-process data flow analysis. [35] proposes a semantic-based feature extraction and detection method for malicious code, which extracts the key behaviors of malicious code and the dependencies between behaviors. The advantages of machine learning analysis are low implementation overhead and simple operation. The disadvantage is that it is influenced by the difference of training applications and the selection of characteristics. Besides, the current machine learning approach does not support analysis of inter-apps.

Most above analysis methods can be used for the analysis on the information flow within a single component or application. In addition, many researches are proposed for information flow analysis on the inter-communication between different components [2,7,11,18,19]. IccTA [2] combines multiple applications into one and performs intra-app analysis on the combined one. Covert [7] is

a tool for analyzing vulnerabilities across applications that allows incremental analysis of applications while they are being installed, updated, or deleted. MR-Droid [19] empirically assesses ICC risks and tests for high-risk pairs. However, most of these approaches works in a global way, in which they analyze the flows across multiple components by modeling them as one combined entity. In other words, the whole model must be remodeled even if there is a little change on one component. And it would cost many efforts but with little benefits. Besides, flows in the unchanged component or application will be reverified during the analysis, which causes the additional verification load on mobile devices.

In order to reduce the overhead of verifying compositional information flow security, this paper presents a compositional approach to automatically verify security of information flow across multiple components or applications. This paper presents the following original contributions:

(1) We define a formal model on individual application and sequential composite applications combined by inter-component communication for information flow analysis.
(2) We make the formal security constraints on information flow for each participant across multiple applications.
(3) We propose a compositional information flow verification approaches for secure inter-app communication among applications in android.

The rest of the paper is structured as follows. Section 2 presents the motivation examples for this study. Sections 3 and 4 defines the formal models and propose a security theorem for compositional information flow which verify with the verification framework in Sect. 5. Section 6 evaluates our methodology and Sect. 7 is conclusion.

2 Motivating Example

To illustrate our approach, we provide a concrete example of information transmission among different android's apps through Inter-Component Communication (ICC). Android provides a flexible application level message known as Intent for communications between components. The example includes three apps. App_1 contains $GetDataActivity$ which obtains the user's sensitive data such as phone number, e-health record and so on, which is shown in LIST1. App_2 contains $ForwardActivity$, which receives sensitive data (user's movement steps) from intent message $MSteps$ and forwarding it to App_3 by intent message $Fsteps1$. App_2 is shown in LIST2. App_3 contains $ReceiveDataActivity$ which is responsible for receiving intent message and sends the data to a remote server which can exploit it at will.

LIST 1 : App_1: send an intent to transmit data

```
1  public class GetDataActivity extends AppCompatActivity {
2      protected void onCreate(Bundle savedInstanceState) {
3          ...
```

```
4         sp = getSharedPreferences("User", Context.MODE_WORLD_READABLE);
5             SharedPreferences.Editor edit = sp.edit();
6             edit.putString("Value",meditText1.getText().toString().trim());
7                 edit.commit();
8                 String value = sp.getString("Value","Null");
9                  ...
10                 Intent MSteps = new Intent();
11                  Bundle bundle = new Bundle();
12                   MSteps.setAction("com.example.second");
13                   bundle.putString("params3", value);
14                  MSteps.putExtra("bundle", bundle);
15                  startActivity(MSteps);
16        }
17     }
```

LIST 2:App_2: receive an intent and send an intent to transmit data

```
1  public class ForwardActivity extends AppCompatActivity {
2   protected void onCreate(Bundle savedInstanceState) {
3      ...
4     final  Intent Rsteps1 = getIntent();//source
5
6      button.setOnClickListener(new View.OnClickListener() {
7         public void onClick(View v) {
8                ...
9                 Intent Fsteps1 = new Intent();
10                Bundle bundle1 = new Bundle();
11                Fsteps1.setAction("com.example.three");
12                String value1 = text.getText().toString();
13                bundle1.putString("para",value1);
14               Fsteps1.putExtra("bundle",bundle1);
15                Fsteps1.putExtra("para",value1);
16               startActivity(Fsteps1);//sink
17       });
18     }
```

Listing 3: App_3: receives an intent

```
1   public class ThreeActivity extends AppCompatActivity {
2   protected void onCreate(Bundle savedInstanceState) {
3       ...
4          final Intent Rsteps2 =getIntent(); //source
5          Bundle bundle = intent.getBundleExtra("bundle");
6          final String value = bundle.getString("para");
7          text.setText(value);
8             ...
9     }
10  }
```

More specifically, from line 4 to line 9 in LIST 1, *GetDataActivity* edits a e-health record and stores it in SharedPreferences which is a lightweight storage class on the Android platform. From line 10 to line 15, *GetDataActivity* sets the action of Intent, then gets the stored e-health data and subsequently sends it to *ForwardActivity* through an Intent message. The Intent filter which is defined

in the Manifest file of *App*2 is responsible for receiving this Intent message and handle it. Likewise, *ForwardActivity* gets an Intent and receives users movement step data of the message from *GetDataActivity* (On line 4 in *App*2). From line 9 to 16, *ForwardActivity* sets the type of Intent and sends an Intent message to *ReceiveDataActivity*. In *ReceiveDataActivity*, it gets the Intent message and receives the data.

In this example, if App_3 is malicious software or is monitored by an attacker, the sensitive data in App_1 may be leaked to the attacker even though the information flow is secure in App_1 and App_2.

3 Android Application Model

In Android system, an application is composed by components which are described in a special file called Manifest. There are four kinds of components, i.e., activity, service, content provider and broadcast receiver. Activities construct user interface of an app. Each app may have multiple activities representing different screens of the application to the user. Services do not have user interface but perform time-consuming tasks in the background. Content providers act analogous to a database and provide access to a constructed set of data. Broadcast receivers listen to global events.

Referring to the android application model described in [7], an app model can be formally defined as follows.

Definition 1. *A model for an android app is a tuple* $A_i = <C_i, I_i, IF_i, Sec_i> 1 \leq i \leq n$, *where*

C_i *is a set of components represent as* $C_i = \{ c_1, c_2, \ldots, c_m \}$, *and each* c_j $(j < m)$ *is a component of* A_i. *Each component contains a series of methods for executing the required functions. We use* M_i *to represent the set of methods that used in application* A_i.

I_i *is a set of event messages called intents that can be used for both intra-and inter-app communications. Here we use* $In_{i,j}$ *to represent an intent message set from Application* A_i *to Application* A_j *where* $In_{i,j} \subset I_i$.

IF_i *is a set of Intent filters. Each intent filter is attached to a component and responsible for filtering implicit intents.*

Sec_i *is the set of security properties of all methods in* A_i.

Different applications can cooperate with each other to fulfill different user's requirements through ICC. This paper studies a simplified type of inter-application communications, i.e., sequential inter-application communication. And we call these applications as sequential composite applications. Sequential composite applications is composed by a set of applications $A_1, A_2, \ldots, A_n$ which communicates with each other in a sequential way. According to the characteristics of sequential composite application, its model can be defined as follow.

Definition 2. *The composite apps* A_c *can be represented as a tuple* $A_c = <AC, CI, CIF>$. *where*

AC is a sequential group of apps in which each app has only one predecessor and one successor.
CI is a collection of all intents using for communication between A_i and A_{i+1}.
CIF is the set of all Intent filters from all the applications used as the entry point for the adjacent application communications.

Based on the above definition of the composite application model, the model of the example in Sect. 2 can be extracted as A_c. $AC = \{App_1, App_2, App_3\}$, where, $App_1 = <\{GetDataActivity\}, \{Msteps\}, \{Interfilters\}>$. Each intent is shown in bold in the code in Sect. 2. $Interfilters$ are defined in the corresponding application Manifest file.

4 Secure Information Model for Composite Application in Android

4.1 Security Label Model

As a device for storing and processing data, android phone contains a lot of sensitive information, such as e-health, contacts, and so on. According to the different sensitivity of information, we use multi-level security model to describe the security properties of data.

By referring to [20], security label model can be defined as a lattice $(SL, \geq)$, where SL is a finite set of security levels that is orderly by $\geq$.

We define a function $g : M_i \rightarrow sl$ to represent the security level of each method in application A_i. Based on the security label model, security property in A_i can be represented by the security level on the methods.

4.2 Information Flow in Intra-app

For intra-app information flows, we use static analysis technique [4] to analyze them. In one application's component, there are source methods that are responsible for accessing sensitive data such as phone numbers and sink methods that are responsible for outputting data [5]. During the execution of the application, data are received by different sources, processed by methods in component, and finally outputted by different sinks, which constructs different data prorogation paths. And we can define the information flow within an intra-app as follows.

Definition 3. *The information flow of android's app A_i can be represented as a tuple $flow = <source, sink>$ where, $source, sink \in M_i$.*

Based on the above description, we define $Flow_i = \{<source, sink> | source, sink \in M_i, i \in N\}$ as the set of all flows in A_i.

Combining with the multi-level security model, $sl(source)$ and $sl(sink)$ are used to represent the security levels of sources and sinks. According to the definition of non-interference [13], the security of information flow in a intra-app can be formally defined as follows.

Definition 4. *The information flows in application A_i are secure if it satisfies that for $\forall source, sink \in M_i$, $sl(source) > sl(sink)$, there is no existence of information flow from source to sink, namely $<source, sink> \notin Flow_i$.*

4.3 Secure Information Flow in Composite Applications

Considering sequential group apps $A_1, A_2, \ldots, A_n$, when applications communicates with each other by intents, data are passed from the *sink* method of one component in A_i to the *source* method of component in A_{i+1}, which forms an inter-app information flow across multiple applications. And the following apps $A_j (j > i)$ may leak data despite the information flow in A_i is secure. The Fig. 1 shows the intra and inter flows in our example in Sect. 2.

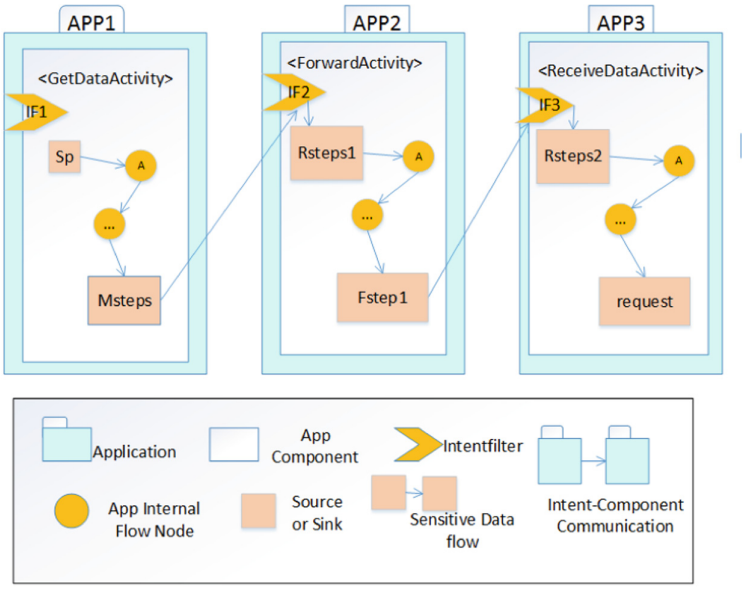

Fig. 1. Composite information flow

For sequential composite application A_C, the definitions of inter-app information flow can be given as follows.

Definition 5. *For* $\forall source_i \in M_i$, $\forall sink_j \in M_j$, *there is a inter flow* $flow_{i,j} = <source_i, sink_j>$ *from* A_i *to* A_j, *if they satisfy one of the following conditions.*

(1) For $i = j - 1$, *there* $\exists sink_i \in M_i$, $source_j \in M_j$ *that satisfy* $<sink_i, source_j> \in In_{i,j}$.

(2) For $i \neq j - 1$, $\exists A_k, 1 < k < j; \exists source_k \in M_k, \exists sink_{k+1} \in M_{k+1}$, *and they satisfy that* $\exists flow_{i,k} = <source_i, sink_k>, \exists flow_{k+1,j} = <source_{k+1}, sink_j>$ *and* $\exists <sink_k, source_{k+1}> \in In_{k,k+1}$.

According to the above description of the inter-app information flow, we use $Flow_{inter}$ to represent the set of all inter-app information flows in the sequential composite applications where $Flow_{inter} = \bigcup_{0 \leq i,j \leq n} flow_{i,j}$.

Then we can obtain the following definition on secure information flow in composite application.

Definition 6. *The information flows in sequential composite application A_C are considered secure iff it satisfies that for each $A_i \in A_C$, for $\forall source_i \in M_i, \forall sink_j \in M_j (i \leq j)$, there is no existence of information flow $source_i$ to $sink_j$, namely $<source_i, sink_j> \notin Flow_i \cup Flow_{inter}$.*

Based on the composite information flow security definition above, we can obtain the following theorem.

Theorem 1. *In the security sequential combinatorial application $A_C s$, the information flow must satisfies following conditions.*

(1) For $\forall <source, sink> \in Flow_i$ in A_i, there is $sl(source) \leq sl(sink)$.
(2) For $\forall <sink_i, source_{i+1}> \in In_{i,j}$, there is $sl(sink_i) \leq sl(source_{i+1})$.

We can use the mathematical induction to prove the theorem by referring to [20].

5 Compositional Information Flow Verification for Composite Application Android System

Android inter-app communication is a basic behavior that usually occurs during system running. For example, wechat accesses health data and makes statistical ranking. The famous social software Weibo adds friends through visiting contacts. In order to ensure the data security across multiple applications, we propose an compositional information flow security Theorem 1. According to the Theorem 1, we can infer that for a sequential composite application, the information flow security verification procedure includes two different phases, i.e., the intra-flow verification and inter-flow verification.

5.1 Intra Flow Verification in Single Application

In the process of intra-flow verification, we first use the flowdroid tool [4] to obtain the application's sensitive information flow. Then each flow is verified according to condition (1) in Theorem 1. If the flows in the application is valid, the certificate is generated which can be used for the inter flow verification to avoid the repeated verification. The certificate includes all essential information for inter flow verification, e.g., the security level on each source and sink method and so on. The procedure for flow validation and certificate generation is shown in Algorithm 1.

After successful verification, the generated certificate will be stored in the database. This procedure can be executed during the application is going to be installed on the system at the first time. And only secure ones can obtain the certificates while the others are not allowed to be installed. The counterexample will also be return to users.

Algorithm 1. Intra-app verification & certificate set-up

Input: $A_i =< C_i, I_i, IF_i, Sec_i >$Apk file
Output: $Ce =< i, It, sl >$, true, false

use flowdroid anaysis apk file to get the set of intra flow $FlowResult$
for each flow $f \in FlowResult$ **do**
  get security level $sl(source)$ and $sl(sink)$ from Sec_i
  if $(sl(source) > sl(sink))$ **then**
    break
  else
    n++
  end if
end for
if $(n < |FlowResult|)$ **then**
  return false
else
  generate the certificate Ce base on I_i, Sec_i
  signature(Ce, CA)
  return true
end if

5.2 Compositional Flow Verification for Inter-application Communications

According to the condition (2) in Theorem 1, the security on the inter-app flows can be ensured by verification on the inter flows between the adjacent applications. The compositional verification algorithm is as follows.

In the compositional verification process, the certificate is obtained first and then the adjacent application's inter-flows are verified. If the validation is successful, the result is returned to the user. Otherwise, return counterexample of an insecure flow.

6 Implementation and Evaluation

This paper mainly studies the compositional application's information flow security verification approach. In this section, we experimentally compare the verification time overhead of our approach with the global verification approach. Our approach has been described in Sect. 5. The global approach first uses ApkCombiner [21] to combine multiple applications into one, and then use flowdroid [4] for information flow analysis and verification. The basic experiment configuration is shown in Table 1 and verification results are shown in Figs. 2 and 3. In Table 1, the Applications number refers to the number of applications tested, and Combined applications number refers to the number of compositional applications formed.

Algorithm 2. Compositional verification approach

Input: A_i, A_{i+1}
Output: true, false, leakagepath

```
1: get Ce_i and Ce_{i+1}
2: for each intent in ∈ Ce_i · It do
3:     get security level of sink_i and source_{i+1} from Ce_i and Ce_{i+1}
4:     if (sl(sink_i) > sl(source_{i+1})) then
5:         leakagepath =< sink_i, source_{i+1} >
6:         break
7:     else
8:         n++
9:     end if
10: end for
11: if (n < |Ce_i · It|) then
12:     output leakagepath
13:     return false
14: else
15:     return true
16: end if
```

Table 1. Basic configuration

General	
Testing tools	ApkCombiner, Flowdroid, Our approach
Application	
Applications form	apk files
Applications number	3, 6, 9, 12, 15, 18
Combined applications number	1, 8, 27, 64, 125, 216
Inter-app communications number	1, 2, 3, 4, 5, 6, 7

Figures 2 and 3 show that the verification time of the global method is much higher than that of our approach. The reason is that when the communication among different applications changes, the global approach needs to reverify all the flows in the whole composite application. On the contrary, our approach only needs to verify the relevant inter flows between the applications according to certificate, which saves lots of costs on the reverification on intra flows in the same application. Besides, our compositional verification algorithm is easy to extend. With the increasing number of steps n, we only need to verify the additional flows between A_n and A_{n+1} to ensure the security of the composite application.

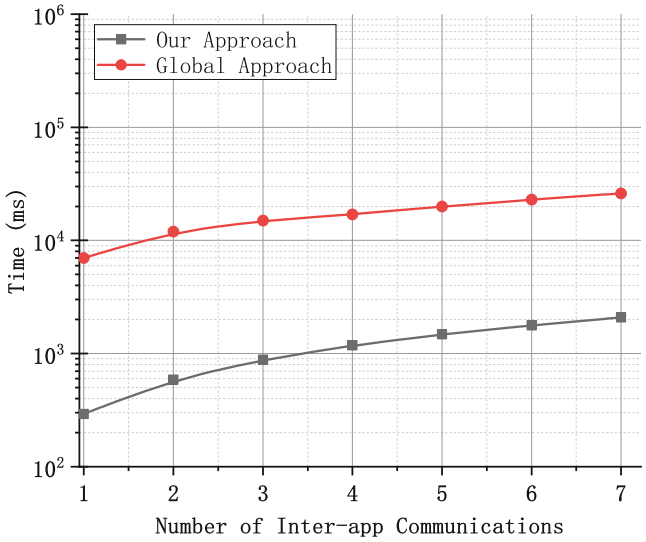

Fig. 2. The verification time increases with the inter-app communication number

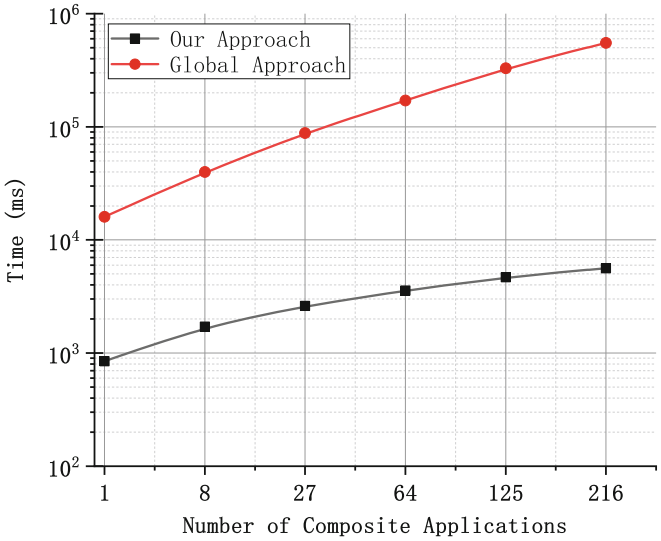

Fig. 3. Verification time increases with the number of composite applications

7 Conclusion

ICC is used to communicate among multiple applications, which may cause leakage on users' sensitive data. In this paper, we design the formal model of android application and sequential composite applications by ICC. Then, through the

analysis of the intra and inter information flows, we get the security theorem on sequential composite application. Based on the theorem, we design a compositional information flow security verification algorithm. The process of information flow verification in intra and inter app is separated to avoid repeated verification in application when communication changes. Finally, we compared our approach with the global verification through the experimental evaluation. And the results show that our approach can reduce the overhead of information flow verification effectively. Since our approach relies on the accuracy of the information flow recognized by flowdroid [4], in the future, we are going to improve the precision of information flow validation by using machine learning to identify source and sink more accurately.

References

1. Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A., Shastry, B.: Towards taming privilege-escalation attacks on Android. In: NDSS 2012 (2012)
2. Li, L., et al.: Detecting inter-component privacy leaks in Android apps. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 280–291 (2015)
3. Marforio, C., Ritzdorf, H., Francillon, A., Capkun, S.: Analysis of the communication between colluding applications on modern smartphones. In: ACSAC 2012 (2012)
4. Arzt, S., et al.: FlowDroid: precise context, flow, field, object sensitive and lifecycle-aware taint analysis for Android apps. ACM SIGPLAN Not. **49**(6), 259–269 (2014)
5. Rasthofer, S., et al.: A machine-learning approach for classifying and categorizing Android sources and sinks. In: Proceedings of 14th Network and Distributed System Securit (NDSS) (2014)
6. Enck, W., et al.: TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones. Commun. ACM (2014)
7. Bagheri, H., Sadeghi, A., Garcia, J., Malek, S.: Covert: compositional analysis of Android inter-app permission leakage. IEEE TSE **41**(9), 866–886 (2015)
8. Bohluli, Z., Shahriari, H.R.: Detecting privacy leaks in Android apps using inter-component information flow control analysis. In: Proceedings of 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC), pp. 1–6 (2018)
9. Chen, H., Leung, H.-F., Han, B., Su, J.: Automatic privacy leakage detection for massive Android apps via a novel hybrid approach. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–7 (2017)
10. Gordon, M.I., Kim, D., Perkins, J.H., Gilham, L., Nguyen, N., Rinard, M.C.: Information flow analysis of Android applications in DroidSafe. In: NDSS (2015)
11. Bosu, A., Liu, F., Yao, D., Wang, G.: Collusive data leak and more: large-scale threat analysis of inter-app communications. In: ASIACCS (2017)
12. Fuchs, A.P., Chaudhuri, A., Foster, J.S.: ScanDroid: automated security certification of Android applications. Technical report, Department of Computer Science, University of Maryland, College Park (2009)
13. Goguen, J.A., Meseguer, J.: Security policies and security models. In: 1982 IEEE Symposium on Security and Privacy, pp. 11–20. IEEE (1982)
14. Bagheri, H., Sadeghi, A., Jabbarvand, R., Malek, S.: Automated dynamic enforcement of synthesized security policies in Android. Technical report (2015)

15. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute this! Analyzing unsafe and malicious dynamic code loading in Android applications. In: NDSS 2014, no. February, pp. 23–26 (2014)
16. Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., Hoffmann, J.: Mobile-sandbox: having a deeper look into android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1808–1815. ACM, Coimbra (2013)
17. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 95–109. IEEE (2012)
18. Jing, Y., Ahn, G.-J., Doupe, A., Yi, J.H.: Checking intent-based communication in Android with intent space analysis. In: ASIACCS (2016)
19. Liu, F., Cai, H., Wang, G., Yao, D., Elish, K.O., Ryder, B.G.: MR-Droid: a scalable and prioritized analysis of inter-app communication risks. In: 2017 IEEE Security and Privacy Workshops (SPW), pp. 189–198 (2017). 10.11999JEIT140902
20. Xi, N., Ma, J., Sun, C., Shen, Y., Zhang, T.: Distributed information flow verification framework for the composition of service chain in wireless sensor network. Int. J. Distrib. Sens. Netw. **2013**, 10 (2013)
21. Li, L., Bartel, A., Bissyandé, T.F., Klein, J., Traon, Y.L.: ApkCombiner: combining multiple Android apps to support inter-app analysis. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 513–527. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_34
22. Harrison, R.: Investigating the effectiveness of obfuscation against Android application reverse engineering. Royal Holloway University of London, RHUL-ISG-2015-7 (2015)
23. Ghosh, S., Tandan, S.R., Lahre, K.: Shielding Android application against reverse engineering. Int. J. Eng. Res. Technol. **2**(6), 2635–2643 (2013)
24. Protsenko, M., Mller, T.: Protecting Android apps against reverse engineering by the use of the native code. In: 12th International Conference on Trust and Privacy in Digital Business, Valencia, Spain, pp. 99–110 (2015)
25. Strazzere, T.: DEX education 201: anti-emulation. In: HITCON 2013 (2013)
26. Wolfe, B., Elish, K.O., Yao, D.D.: Comprehensive behavior profiling for proactive Android malware detection. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 328–344. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_19
27. Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P.: DroidMat: Android malware detection through manifest and API calls tracing. In: Proceedings of the Asia Joint Conference on Information Security (Asia JCIS), pp. 62–69 (2012). https://doi.org/10.1109/AsiaJCIS.2012.18
28. Gascon, H., Yamaguchi, F., Arp, D., Rieck, K.: Structural detection of Android malware using embedded call graphs. In: Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISEC), pp. 45–54 (2013). https://doi.org/10.1145/2517312.2517315
29. Chakradeo, S., Reaves, B., Traynor, P., Enck, W.: MAST: triage for market-scale mobile malware analysis. In: Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC), pp. 13–24 (2013). https://doi.org/10.1145/2462096.2462100
30. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in Android. In: Zia, T., Zomaya, A., Varadharajan, V., Mao, M. (eds.) SecureComm 2013. LNICST, vol. 127, pp. 86–103. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-04283-1_6

31. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: DREBIN: effective and explainable detection of Android malware in your pocket. In: Proceedings of the 21th Annual Symposium on Network and Distributed System Security (NDSS 2014) (2014). https://doi.org/10.14722/ndss.2014.23247
32. Zhang, X.Y., Zhang, G., Shen, L.W., Peng, X., Zhao, W.Y.: Similarity analysis of multi-dimension features of Android application. Comput. Sci. **43**(3), 199–205, 219 (2016). (in Chinese with English abstract). https://doi.org/10.11896/j.issn.1002-137X.2016.03.037
33. Kong, D.G., Cen, L., Jin, H.X.: AUTOREB: automatically understanding the review-to-behavior fidelity in Android applications. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS 2015), pp. 530–541 (2015). https://doi.org/10.1145/2810103.2813689
34. Zhang, M., Duan, Y., Feng, Q., Yin, H.: Towards automatic generation of security-centric descriptions for Android apps. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS 2015), pp. 518–529 (2015). https://doi.org/10.1145/2810103.2813669
35. Wang, R., Feng, D.G., Yang, Y., Su, P.R.: Semantics-based malware behavior signature extraction and detection method. Ruanjian Xuebao/J. Softw. **23**(2), 378–393 (2012). https://doi.org/10.3724/SP.J.1001.2012.03953. (in Chinese with English abstract), http://www.jos.org.cn/1000-9825/3953.htm