DroidTrack: Tracking and Visualizing Information Diffusion for Preventing Information Leakage on Android

Shunya Sakamoto[1], Kenji Okuda[2], Ryo Nakatsuka[1], and Toshihiro Yamauchi[1]*
[1]Graduate School of Natural Science and Technology, Okayama University, Japan
[2]Faculty of Engineering, Okayama University, Japan

Abstract

An Android app can collaborate with other apps by using an intent. It can also control personal information or use permissions granted by a user. However, users cannot detect when their apps communicates with other apps. Therefore, they might not be aware of any information leakage if an app happens to be malware. In this paper, we propose a method for tracking and visualizing the diffusion of sensitive information and preventing its leakage on an Android device. This method, which we call DroidTrack, alerts a user that there is the possibility of information leakage when an app uses APIs (Application Program Interfaces) to communicate externally. These alerts are triggered only if the app has already called APIs to collect sensitive information. Users are given the option to refuse the execution of the API if it is not appropriate. Furthermore, by illustrating how their personal data is shared, users are provided with the necessary information to help them decide whether an API call is appropriate.

Keywords: android, information leakage prevention, security, visualization

1 Introduction

Recent years have witnessed the rapid proliferation of smartphones, and Android [4] has emerged as a popular smartphone operating system (OS). App developers can easily develop an Android app and make it available through a Web site such as Google Play Store [6]. However, an app [11] can hijack administrative privileges by exploiting vulnerabilities in the Android OS, and thus send out illegally collected sensitive information.

In particular, a major issue is the widespread emergence of malware, which performs unwanted or unexpected processing. Furthermore, apps that are not sufficiently malignant to be blacklisted as malware call application program interfaces (APIs) that collect sensitive information inappropriately. This makes it difficult to ensure transparency when the app handles user information. Malware that targets the Android OS is usually intended to illegally collect sensitive information. A mobile device contains a large amount of personal information such as names, addresses, and phone numbers, and this information can be easily obtained by apps using the Android APIs. Moreover, many users are unaware of smartphones' lack of security and built-in anti-malware software. Therefore, there is a possibility of information leakage due to malware, without the user's knowledge.

An Android app is executed in a sandbox; communication with other apps is severely restricted, and requires the use of an intent [7]. Key features such as external communications and the collection of sensitive information require permissions that are granted by the user. However, the user can neither detect the collection of sensitive information by the app nor determine whether that sensitive information has been leaked.

Journal of Internet Services and Information Security (JISIS), volume: 4, number: 2, pp. 55-69

*Corresponding author: Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, kita-ku, Okayama, 700-8530 Japan, TEL: +81-86-251-8188, Email: yamauchi@cs.okayama-u.ac.jp, Web: http://www.swlab.cs.okayama-u.ac.jp/~yamauchi/index.html

In this paper, we propose DroidTrack, a method for tracking information diffusion and preventing information leakage through the Android OS. This method tracks information diffusion after the app has obtained sensitive information. DroidTrack alerts the user if there is a possibility of information leakage, and allows the user to limit the use of the API. It monitors any app that uses an information-gathering API, and also displays a warning when the app uses an API that sends information externally. Sensitive information can also be leaked when one app obtains such information and sends it to another app, which in turn sends the information externally. For this reason, DroidTrack manages apps by controlling the use of APIs and intents. In addition, the user is allowed to decide whether the use of the API that sends information out of the device is permitted; thus, DroidTrack can prevent information leakage. Further, the visualization of the information-diffusion API reveals the information diffusion path, thereby enabling the user to make such a decision when he/she inputs the availability of the API. Thus, the chances of the user making an error in judgment are reduced.

The contributions of this paper are as follows:

(1) The proposal of a new method to prevent information leakage, which tracks the diffusion of sensitive information by means of intents or APIs.

(2) Visualization method of information diffusion of Android is proposed. Although existing studies track information diffusion using Intents or APIs, to the best of our knowledge, they do not provide visualization and control of such information. The proposed method provides such visualizations, which enable the user to check the information leakage path visually, and give the user more control over the execution of information-diffusing APIs.

(3) Reduce burden of user judgment with regard to granting permissions to intents or APIs, as a consequence of (2).

(4) The proposed method can support to analysis of the operation of a new app. Thus, the user can determine whether the acquisition of sensitive information is necessary for the operation of the app. It also enables the user to check whether the app communicates any information externally.

(5) The proposed method requires only modifications to the Android framework, as opposed to existing studies, which require modifications to both the framework and the Dalvik virtual machine (VM). Therefore, our method is easier to implement.

2 Android Component and Security Issues

2.1 Android Component

In the Android OS (Figure 1), all applications run in the application layer. If an application requires resources, it must use the APIs provided by the application framework. Android apps have individual user IDs (UIDs), and communication with an app having a different UID is highly improbable, except when the intent class is used.

2.2 Permission

Apps cannot use the Android APIs to access a protected resource or to gather sensitive information without a user's permission [3]. An app can request specific permissions, e.g., permission to connect to the Internet (INTERNET) or permission to read the status of the unit (READ PHONE STATE). The safety of the resources is preserved by granting only the minimum permissions required by the app.

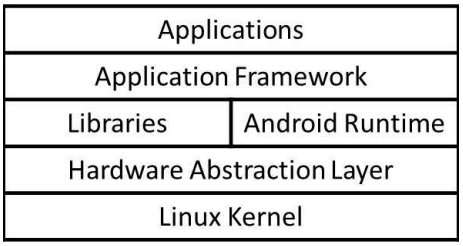

Figure 1: Android component

2.3 Intent

Each Android app runs in its own memory space. By default, apps cannot communicate with another app because they are strictly isolated from each other. However, it is possible to enable communication between apps by using the intent class, which allows an app to access another app and receive the processing results. In addition, an app can pass data either as a string or as an object.

2.4 Security issues in Android

The following are the problems associated with malware infection in the main security areas of the Android OS:

(1) Problems obtaining administrator authority

(2) Problems with development tools such as Android Debug Bridge (ADB) [5]

(3) Permission abuse

(4) Difficulty in detection of information leakage

(5) WebKit abuse

Problems (1), (2), and (3) are the cause of malware infection. Problem (4) and (5) are related to malware behavior. Problem (4) makes it very difficult to inform the user when the app gathers information and what kind of sensitive information it gathers. Therefore, if a user installs malware unknowingly, he/she cannot detect the leakage of sensitive information. In this study, we deal with problem (4) by restricting the transmission of information outside a smartphone.

3 Design principles of the proposed method

3.1 Requirements and challenges

In order to deal with the problems associated with malware infection in the Android OS, we propose the following requirements:

(1) Detection of all APIs vulnerable to information leakage.

(2) User judgment as to whether there is a risk of information leakage.

(3) Prevention of information leakage by disallowing the execution of the API.

To satisfy these three requirements, we propose the following challenges:

(A) Clarifying the conditions for information leakage.

(B) Detecting all APIs that are vulnerable to information leakage.

(C) Controlling the operations of all apps that are vulnerable to information leakage.

(D) Allowing the user to decide whether the app can receive sensitive information.

(E) Showing the details of sensitive information that may be leaked, the accessing app name, and the name of the API used.

3.2 Solution

3.2.1 Solution for challenge (A)

Information leakage can occur when an app uses the Android APIs to obtain sensitive information and then sends it externally (diffusion of information). The app can also obtain sensitive information by using the intent class instead of an information-gathering API. In another scenario, information leakage can occur when one app uses an information-gathering API, and then communicates with another app that uses an information-diffusing API.

In this proposal, we address the following scenarios in which information leakage occurs:

(1) A single app uses an information-gathering API, an information-diffusing API, or an intent.

(2) One app uses an information-gathering API, and then communicates with another app by using either an intent or an information-diffusing API.

3.2.2 Solution for challenge (B), (C) and (D)

As mentioned in Section 3.2.1, information leakage can occur when an app uses APIs that obtain sensitive information and APIs that diffuse information or the intent class. Therefore, to deal with challenges (B), (C), and (D) (detecting all uses of these APIs and controlling the operations of apps), we propose a method to control the behavior of the app in the following manner:

- By intercepting calls to information-gathering APIs, information-diffusing APIs, or the intent class
- By determining the user's preferences for controlling the use of APIs if either scenario described in the Solution for challenge (A) occurs, and
- By controlling the use of the APIs or the intent class based on the user's preferences

3.2.3 Solution for challenge (E)

The conditions for information leakage are determined when the app uses information-diffusing APIs. When DroidTrack shows a user the potential for information leakage in a particular scenario, the user has to judge whether there is the possibility of information leakage via the use of information-diffusing APIs alone. In this case, it is difficult for the user to correctly judge whether information leakage may take place.

For example, when the app is a game in which users compete for a score, it is necessary to acquire a terminal ID in order to identify the user's own score, and to communicate externally for the transmission of that score. In this case, there is a possibility that the user will decide that the likelihood of information

leakage using this app operation is low. However, the user may allow information to be leaked if the app is malware that transmits not only the terminal ID but also information that is unrelated to the game, such as the user's phone number. Furthermore, when we consider Requirement (2), there is a high probability that the user will not be able to correctly judge whether there is a danger of information leakage. For this reason, we need to show the user when the app is diffusing sensitive information processing including the intent.

Hence, we propose a method for visualizing the information diffusion path as a solution for challenge (E). This enables us to assist the user in judging whether information leakage might occur when running the app.

4 Method for tracking diffusion of information and preventing information leakage on Android

4.1 Design principle

To address the solutions for challenges (B), (C) and (D), we propose the following requirements:

(1) The user should be informed if there is potential for information leakage.

(2) The use of APIs and the intent class should be limited based on the user's preferences.

Requirement (1) is necessary because information must be securely provided to the user, in the form of a warning, in order to prevent information leakage if the app exhibits potential to leak information (as described in Section 3.2.1). In addition, the user's preferences must be enforced at the point when the possibility of leakage is detected. Therefore, requirement (2) is necessary.

4.2 Basic method

We propose the following modifications to the Android framework:

(1) "Hook" or intercept calls to information-gathering APIs and information-diffusing APIs, and inform the user of both the name of the app using the API as well as the name of the API used.

(2) "Hook" or intercept calls to the Intent class, and inform the user of both the name of the app that uses the Intent class and the name of the app called by the Intent.

(3) Execute a process based on the user's preferences regarding the use of the APIs or the Intent class.

The APIs are used differently depending on the type of sensitive information that is gathered by the app. Therefore, the change described in (2) allows the user to be informed about when an app obtains sensitive information, what kind of sensitive information is obtained, and when the app attempts to transmit it externally. Furthermore, the change described in (3) can be used to prevent information leakage according to the user's preferences. In the following section, we describe a method for tracking and preventing information diffusion by using the modified framework described above.

4.3 Control of API in the framework

Figure 2 shows the flow of control of an API in the framework. DroidTrack consists of two "Control Apps" in the application layer and one "Calling Control AP Unit" in the application framework layer. In Figure 2, the "Calling Control AP Unit" informs the "Control AP" whenever an app calls the information-diffusing APIs, and returns to the "Determine Unit," which determines the user's preferences. "Control

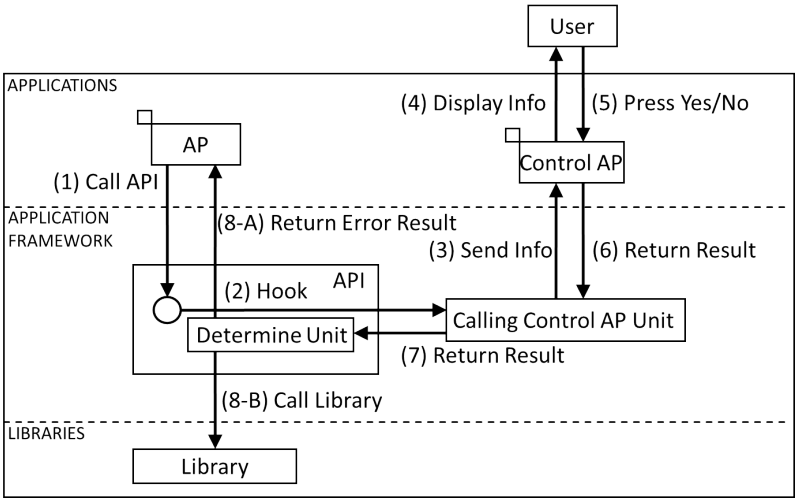

Figure 2: Flow of proposed method

AP" is an app that provides information about the API to the user, and prompts the user to choose whether to restrict the use of the API. The following describes the details of the process flow in the framework:

(1) The app "AP" calls an information-diffusing API.

(2) The "Hook" intercepts the call to the information-diffusing API in the framework.

(3) The "Calling Control AP Unit" passes information about the intercepted call to the "Control AP."

(4) The "Control AP" displays a warning dialog to the user if it suspects that an information leak is possible.

(5) The user responds to the dialog to indicate whether he/she will allow the use of the API.

(6) The "Control AP" forwards the user's preferences to the "Calling Control AP Unit."

(7) The "Calling Control AP Unit" returns the result to the "Determine Unit."

(8) The "Determine Unit" handles the API based on the user's preferences, in the following manner:

(A) Error handling is invoked if the user disallows the API call by the app.

(B) The API process resumes API processing if the user permits the API call by the app.

The procedure mentioned above satisfies requirement (2) by asking for the user's preferences before using APIs. The APIs are then processed according to the user's preferences.

4.4 Hook function

In order for an app to access sensitive information, it is necessary to use an information-gathering API. For this purpose, DroidTrack hooks the information-gathering API, and obtains information about the calling app and the sensitive information thus acquired.

The Intent class is used when sensitive information is exchanged between apps. When the Intent class is used, DroidTrack hooks the Intent call and acquires information about the calling app. Generally, information leakage occurs when information-diffusing APIs are used and sensitive information is

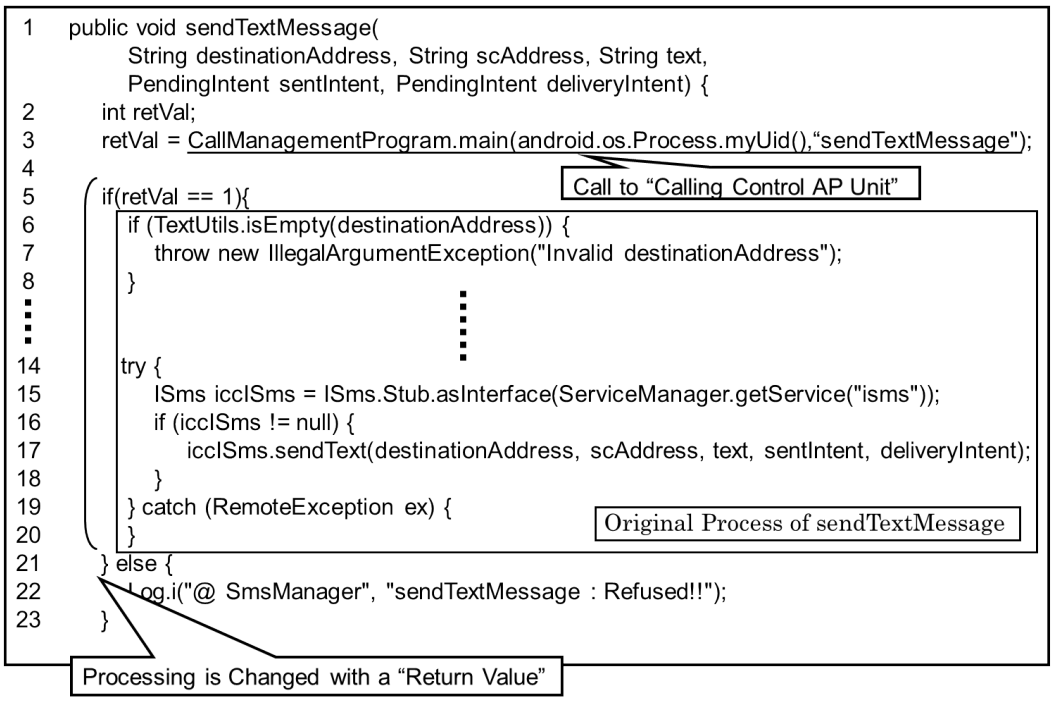

Figure 3: Method of Hooking API

transmitted externally from the device. In this situation too, DroidTrack hooks the information-diffusing APIs.

Figure 3 shows the method of hooking "sendTextMessage" API. The following describes the details of the process in "Hook function":

(1) "Calling Control AP Unit" is called. The uid of app and the name of API to be called are transmitted before operating the original process of the API.

(2) A "Return Value" is returned by the user's judgment.

(3) Processing is selected by the "Return Value".

(A) If the user permits the API call by the app, an original process of API is performed.

(B) If the user disallows the API call by the app, error handling is invoked.

We investigated Android 2.3.3 (API Level 10). Table 1 shows the list of information-gathering APIs which are hooked by our method. In addition, Table 2 shows the list of information-diffusing APIs. The group names to which the relevant permissions concerning personal information belong to are PERSONAL INFO, LOCATION, ACCOUNTS, and PHONE CALLS. Another permission that is considered as high-risk is that of COST MONEY. If this permission is abused, problems related to fee collection arise. Moreover, app needs to use the permission which belongs to NETWORK group since network access is needed in order to information leakage. The permissions belonging to the aforementioned group names whose protection level is listed as "dangerous" are considered to be high-risk permissions. Thus, we see that Table 1 and Table 2 provide us with information diffusion and information leakage details by showing us which APIs are "hooked" by our methods.

Table 1: Information-gathering API

API Name	API Name
getCellLocation	getDeviceId
getNetworkOperator	getPhoneType
getSubscriberId	getLine1Number
getSimSerialNumber	getVoiceMailAlphaTag
getVoiceMailNumber	getAllProviders
getBestProvider	getGpsStatus
getLastKnownLocation	editProperties
getAccounts	getAuthToken
getPassword	getUserData
peekAuthToken	getName
getProfileConnectionState	getProfileProxy
getParams	getUngzippedContent
getCertificate	getAllBookmarks
getAllVisitedUrls	

Table 2: Information-diffusing API

API Name	API Name
sendDataMessage	sendMultipartTextMessage
sendTextMessage	

4.5 Calling Control AP Unit

The "Calling Control AP Unit" receives information from the "Hook Function" and then calls the Control AP. This unit also receives a result from the "Control AP" and returns the result to the "Determine Unit."

4.6 Control AP

4.6.1 Basic mechanism

Figure 4 illustrates the "Control AP" mechanism as follows:

(1) Search-Leakage Function
The "Search-Leakage Function" triggers the check for information leakage performed by the "Information Diffusion Management Unit". If there is a possibility of information leakage, it transfers control to the "Control-Write-Out Function." If there is no possibility, it allows the API calls to be processed.

(2) Information Diffusion Manage Unit
The "Information Diffusion Management Unit" updates the "Information Diffusion Data Structure"; examines the possibility of information leakage; and returns the result of whether there is possibility of an information leakage to the Search-Leakage function. We discussed in the next chapter how to detect information leakage.

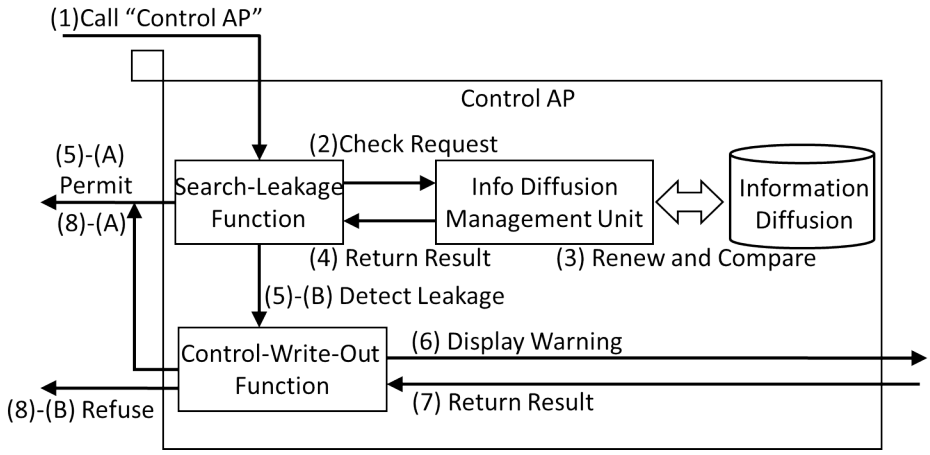

Figure 4: Basic mechanism of Control AP

Table 3: Diffusional information stored in the “Control AP”

Holding Information	Content
UID of app	Identifier of app
App name	Name of app
API name	Name of API which was used by App
Time	The time when App used API

(3) Control-Write-Out Function
The “Control-Write-Out Function” displays a warning dialog, and then accepts and returns the user’s preferences regarding the use of the APIs.

4.6.2 Tracking Information Diffusion and Preventing Information Leakage

The “Control AP” holds the diffusional information that is shown in Table 3. The “Calling Control AP Unit” passes this information to the “Control AP” when the app uses information-gathering APIs, information-diffusing APIs, or processes an intent. The “Information Diffusion Management Unit” adds the API name and the time to the diffusional information on the app, which called the API when the hook data for the information-gathering API is received. If the app which called the information-gathering API is not the management app at this time, the abovementioned processing is performed after adding the new UID and the app name. The current app is then made the management app.

The “Information Diffusion Management Unit” checks whether the app which called the intent is the management app, when the hook information for the intent is received. If it is the management app, the “Information Diffusion Management Unit” adds the app which received the intent to the management app. It also returns the result of the check, i.e. whether the intent could lead to the diffusion of sensitive information, to the “Search-Leakage Function.” If there is no information diffusion, the result that there is no possibility of information leakage is returned to the “Search-Leakage Function” without the renewal of diffusion information. When the hook information regarding the information-diffusing API is received, the “Information Diffusion Management Unit” will judge whether there is the possibility of information leakage. If the app which called information-diffusion API is the management app, the “Information Diffusion Management Unit” judges with possibility of information leakage. If the app is

not management app, it judges with no possibility of information leakage.

5 Visualization of Information Diffusion

5.1 Remarks

In order to deal with Challenge (E), we propose the following method for visualization of the information diffusion path. The requirement for this visualization is as follows:

(1) The visualization of the information diffusion path should be shown to the user when the "Control AP" displays the warning dialog.

The user must judge whether there is the possibility of information leakage, and determine whether to use a particular API by analyzing the information displayed in the warning dialog. In order to assist in the user's decision making, it is necessary to show the information diffusion path intelligibly when the warning dialog is displayed.

Besides helping the user determine whether there is information leakage, the visualization of information diffusion enables us to investigate which API the specific app has used in the past. This is made possible by using the accumulated diffusion information. Thereby, the user can find whether the installed app has acquired information which it does not require for its operation. Moreover, the visualization of the information diffusion path can also be used for the analysis of malware action.

In this paper, we propose an app, which is mounted on the Android OS, for visualization of information diffusion. This app visualizes the information diffusion path of sensitive information, and displays this visualization when a user arbitrarily calls the "Control AP." The "Visualization AP" is a function that analyzes the log of the information diffusion, and displays the text output in the form of a flow chart.

5.2 Implementation

5.2.1 The Call from the Control AP

In order to satisfy Requirement 5.1-(1), the visualization function has to be called from the "Control AP." However, it is also considered that it may be able to judge not displaying propagation information propagation only in a warning dialog. Then, the button which calls the "Control AP" is displayed in a warning dialog; and depending on the user input, the "Visualization AP" is called.

5.2.2 Method of Visualization

The "Control AP" publishes the intent and passes the information to the "Visualization AP". The "Visualization AP" displays a flowchart based on the information received.

In the flowchart, personal information and the management app for pursuit are each expressed by a node. Each node is represented by a button, and the user can view detailed information about the node by pressing the corresponding button. This makes the graph display very simple, and also makes the mechanism of information diffusion easy to grasp. The direction in which information is spread is expressed by an edge (arrow). The information displayed at each node (form) and edge is given below:

(1) Information displayed at each node

(A) Sensitive information
It is desirable to display the personal information which has been acquired by information-gathering APIs. However, when an API acquires a large amount of personal information, all

the information cannot be fit in a node. Therefore, it is displayed at the node as "sensitive information." The sensitive information acquired and the list of APIs which acquire the personal information are displayed by pressing the button at the node. This is making the graph to display as simple as possible, and is for making the whole information diffusion easy to grasp. Moreover, the direction which information diffuses is expressed with edge (arrow). The information displayed on each node (form) and edge is flowing.

(B) Management AP
The "Visualization AP" displays the name and icon of the management app. Package name and UID are displayed by pressing the button at the node.

(C) Information of address
The information of the address which the information-diffusing API communicates with is displayed.

(2) Information displayed at each edges
One information-diffusing API may diffuse two or more pieces of information. For this reason, the time at which information is diffused is displayed beside the edge. Additionally, the name of the API that diffused information for the management app, or is trying to transfer information out of the device, is displayed.

6 Evaluation

6.1 Viewpoint

We evaluated the proposal method by the following viewpoints:

(1) Execution control of API
When the same AP used the information-gathering API and the information-diffusion API, it was evaluated whether an information leakage is detectable. It was also evaluated that the execution of the use of API according to the user's judgment.

(2) Experiment of visualization
When an information leakage has been detected by (1), we evaluated whether the information diffusion path could be displayed by starting the "Visualization AP."

6.2 Execution control of API

Prevention of information leakage by apps was tested using the following procedure:

(1) Obtain the phone number of the mobile device by using "getLine1Number," the unique device ID by using "getDevicedId", the serial number of the SIM by using "getSimSerialNumber" etc. which serve as the information-gathering API.

(2) Transmit the personal information out of the device by using "sendTextMessage," which serves as the information-diffusing API.

Figure 5 shows the dialog displayed by DroidTrack when the example app runs. The user can detect the use of the API by various information from the dialog. In this case, the user presses "Yes" to allow the use of the API and "No" to disallow the use of the API. DroidTrack could prevent information leakage by pressing "No."

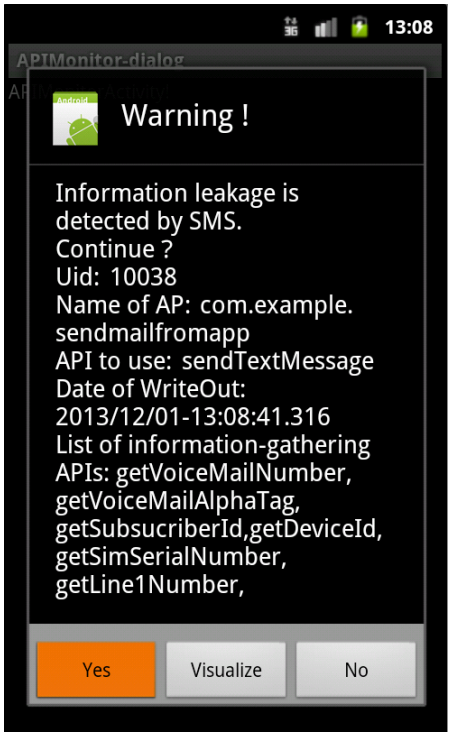

Figure 5: Warning Dialog

6.3 Experiment of Visualization

The "Visualization AP" is started by the "Control AP" when it detects an information leakage in Section 6.1. Figure 6 shows the visualization of information diffusion path. Since the user can understand the flow of information diffusion from a figure, the user can judge whether the use of API is acceptable or not. Moreover, detailed information is displayed by pushing the button of each node or edge. Figure 7 shows the list of sensitive information acquired by information-gathering API. For example, "15555215554" was acquired by "getLine1Number".

7 Related Work

MockDroid [1] allows the user to provide fake or "mock" data to prevent a real sensitive information leakage. This method requires the user to set up permissions for each app. DroidTrack, however, does not require user setup, but needs the user's input only when there is a possibility of information leakage. Furthermore, DroidTrack tracks all transmitted information, including transmissions without leakage, in order to check all API communications coming in or going out of the device.

AppFence [8] (which uses TaintDroid [2]) provides a similar approach, but it modifies the framework and the Dalvik VM. It also uses the policy which is made except on Android. On the other hand, DroidTrack modifies only the Android framework and is therefore easier to implement.

SEEdit [9] makes creation of a security policy in SELinux easier by implementing it in a higher-level language. In contrast, DroidTrack can prevent information leakage without the need for specifying a security policy.

Reference [10] traces the diffusion of sensitive information and prevents information leakage. This

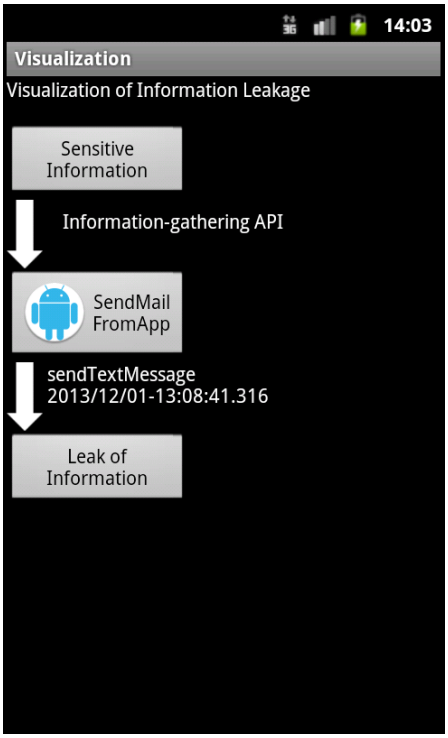

Figure 6: Visualization of information diffusion path

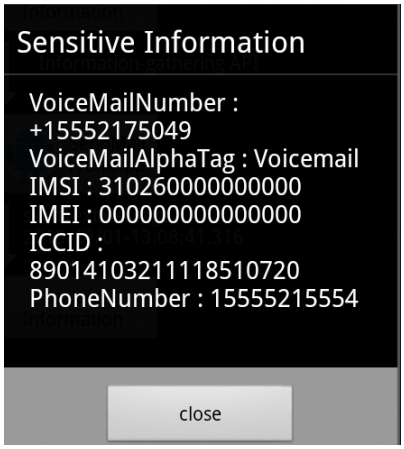

Figure 7: List of sensitive information

method uses the tracing function to trace the sensitive information being exchanged among multiple computers in internal network and to prevent information leakage outside internal network.

Moreover, Android has a security issue such as the abuse of WebView. App that uses WebView can load and display web pages. App can also interact with web pages by using the APIs provided in WebView. If this WebView feature were exploited by an attacker, JavaScript code could be used to launch attacks, such as stealing from or tampering personal information in the device. Reference [12] performs access control on the security-sensitive APIs at the Java object level was proposed to address

these threats.

8 Conclusion

We proposed DroidTrack, a method for warning of the risk of information leakage, by monitoring apps that obtain sensitive information and by keeping track of information diffusion. DroidTrack prevents the leakage of sensitive information by controlling the behavior of APIs based on the user's preferences. These preferences are determined when a warning dialog is displayed. DroidTrack can also detect information leakage in scenarios where the Intent class or information-diffusing APIs are used. In addition, DroidTrack informs the user of the risk of information leakage, and displays a list of information-gathering APIs that could diffuse information to other apps. Furthermore, since the sensitive information which AP used can be grasped by the visualization of information leakage path, DroidTrack can supports the analysis of information leakage path, possibility of information leakage, etc.

Moreover, we evaluated DroidTrack by the evaluation AP and compared the information diffusion path which generated by DroidTrack with the operation of the example AP. We showed that DroidTrack was able to prevent the information leakage and provide accurate visualization of the information diffusion path when example AP ran.

In future work, we will track sensitive information that is encrypted. "Control AP" judges information diffusion by comparing the obtained sensitive information by information-gathering API with the sensitive information which is transmitted to other app using an intent. For this reason, information diffusion becomes difficult when obtained sensitive information is encrypted.

Acknowledgement

This research was partially supported by the Kayamori Foundation of Information Science Advancement.

References

[1] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proc. of the 12th Workshopon Mobile Computing Systems and Applications (HotMobile'11), Phoenix, Arizona, USA*, pages 49–54. ACM, March 2011.

[2] W. Enck, P. Gilbert, B.-G. Chun, L. P.Cox, J. Jung, PatrickMcDaniel, and A. N.Sheth. TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones. In *Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10), Vancouver, British Columbia, Canada*. USENIX, October 2010.

[3] Google. Access permissions. `http://developer.android.com/reference/android/Manifest.permission.html`, last viewed January 2014.

[4] Google. Android. `http://www.android.com`, last viewed January 2014.

[5] Google. Android debug bridge. `http://developer.android.com/tools/help/adb.html`, last viewed January 2014.

[6] Google. Google play. `https://play.google.com/`, last viewed January 2014.

[7] Google. Intent. `http://developer.android.com/reference/android/content/Intent.html`, last viewed January 2014.

[8] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. "these aren't the droids you're looking for": Retrofittingandroid to protect data from imperious applications. In *Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11), Chicago, Illinois, USA*, pages 639–652. ACM, October 2011.

[9] Y. Nakamura, Y. Sameshima, and T. Yamauchi. SELinux security policy configuration system with higher level language. *Information and Media Technologies*, 5(4):1349–1360, 2010.

[10] N. Otsubo, S. Uemura, T. Yamauchi, and H. Taniguchi. Design and evaluation of a diffusion tracing function for classified information among multiple computers. In *Proc. of the 7th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE'13), Seoul, Korea, Lecture Notes in Electrical Engineering*, volume 240, pages 235–242. Springer, May 2013.

[11] webopedia. Droiddream. `http://www.webopedia.com/TERM/D/droiddream.html`, last viewed January 2014.

[12] J. Yu and T. Yamauchi. Access control to prevent attacks exploiting vulnerabilities of webview in android OS. In *Proc. of the 2013 IEEE International Conference on High Performance Computing and Communications (HPCC'13) and the 2013 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC'13), Zhangjiajie, China*, pages 1628–1633. IEEE, November 2013.

Author Biography

Shunya Sakamoto received B.E. and M.E. degrees from Okayama University, Japan in 2012, 2014, respectively. His research interests include computer security and virtualizati on technology.

Kenji Okuda received B.E. degree from Okayama University, Japan in 2012. His research interests include computer security.

Ryo Nakatsuka received B.E. and M.E. degrees from Okayama University, Japan in 2009, 2011, respectively. His research interests include computer security.

Toshihiro Yamauchi received B.E., M.E. and Ph.D. degrees in computer science from Kyushu University, Japan in 1998, 2000 and 2002, respectively. In 2001 he was a Research Fellow of the Japan Society for the Promotion of Science. In 2002 he became a Research Associate in Faculty of Information Science and Electrical Engineering at Kyushu University. He has been serving as associate professor of Graduate School of Natural Science and Technology at Okayama University since 2005. His research interests include operating systems and computer security. He is a member of IPSJ, IEICE, ACM and USENIX.