Hou et al. EURASIP Journal on Wireless Communications and Networking (2016) 2016:227
DOI 10.1186/s13638-016-0711-4

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Investigation of taint analysis for Smartphone-implicit taint detection and privacy leakage detection

Rui Hou*, Zhigang Jin and Baoliang Wang

Abstract

Today's Smartphone operating systems frequently fail to provide users with adequate control and visibility into how the third-party applications use their private data. With TaintDroid realized on Android system, we can detect user's implicit taint and privacy leakage. But TaintDroid has some inherent defects. To better detect user's implicit taint and privacy leakage in the Android platform, this paper analyzes implicit taint detection and then proposes an automated detection system based on dynamic taint tracking, called TaintChaser. Monitoring sensitive data with TaintChaser provides informed use of third-party applications for phone users and valuable input for smart-phone security service firms seeking to identify misbehaving applications. TaintChaser can detect behaviors of user's data leakage in Android applications at a fine granularity level and the system can also analyze and test massive Android software in an automatic way. It uses TaintChaser to automatically analyze 38,268 popular Android applications and finds that 34.41 % of them may leak user's privacy.

Keywords: Dynamic taint track, Taint analysis, TaintDroid, Android system, Privacy leakage, Implicit Taint, Automated test

1 Introduction

In recent years, the Android system spread very quickly and became the most popular Smartphone system. According to the IDC report at the end of the third quarter of 2015, the Android operating system has accounted for 53.54 % of the global intelligent mobile phone market. It indicates an increase of 6.53 % compared to the same period in 2014 [1, 2]. It means that among every five Smartphone shipments to customers, four Smartphones are based on the Android system. In China, the occupation rate of the Android operating system is close to 90 %, but this does not include copycat mobile phones. In other words, more than 90 % of Smartphones use the Android system in China [3]. However, with the increasing popularity of the Android system, user data privacy in mobile phones (e.g. mail list, personal information, email, etc.) has become a prominent problem. Beresford et al. [4] tested 200 pieces of software in the electronics market and found that there were privacy leakage problems in 45 % of the software. Since the number of test samples is few and new Android applications are rapidly increasing, these studies cannot fully reflect the real situation of applications of privacy information leakage in Android electronic markets. Therefore, this paper puts forward and realizes an automated privacy leak detection tool which can automatically detect the mainstream of domestic electronic market in large scale.

2 Current status

2.1 Reference study and shortage

In 2010, Enck et al. achieved the dynamic taint tracking system TaintDroid (refer to Appendix D) [5] based on the Android 2.1 operating system. In 2011, a series of systems based on TaintDroid is also presented, including AppFence and Mock Droid. They are extended based on TaintDroid in different degrees [6], but some inherent defects of TaintDroid have not been resolved. These defects include the following five points:

* Correspondence: hankrui@aliyun.com
School of Electronic Information Engineering, Tianjin University, Tianjin 300191, People's Republic of China

(1) The types of privacy data which can be detected are not comprehensive. IMEI, phone number, location information, photos, and audio can be detected, while phonebook, SMS, and other important information cannot. A variety of privacy sensitive information types is acquired through low-bandwidth sensors, e.g. location and accelerometer. This information often changes frequently and it is usually used by multiple applications simultaneously. Therefore, it is common for a smartphone OS to multiplex access to low-bandwidth sensors using a manager. This sensor manager represents an ideal point for taint source hook placement. For our analysis, we placed hooks in Android's Location Manager and Sensor Manager applications [5]. Device identifier information that uniquely identifies the phone or the user is privacy-sensitive. Not all personally identifiable information can be easily tainted. However, the phone contains several easily tainted identifiers: the phone number, SIM card identifiers (IMSI,ICC-ID), and device identifier (IMEI) which are all accessed through well-defined APIs. We instrumented the APIs for the phone number, ICC-ID, and IMEI.

(2) The privacy leak detection is not enough, only contains the general network communication (socket), and bluetooth, microphone camera, and other means of communication are not monitored. Privacy-sensitive information sources such as the microphone and camera are high-bandwidth. Each request from the sensor frequently returns a large amount of data that is only used by one application [5]. Therefore, the smartphone OS may share sensor information via large data buffers, files, or both. When sensor information is shared via files, the files must be tainted with the appropriate tag. Due to flexible APIs, we placed hooks in both data buffer and file tainting for tracking microphone and camera information.

(3) The string trace granularity is not detailed enough, which may cause possible excessive diffusion on taint produced in tracking process, which results in a false alarm. In TaintDroid, a string object is used as a whole to get taint marks. When a tainted string is split, all of its sub-string can be tainted. This can lead to excessive diffusion of taint. For example, suppose string A "This is a test" is normal data without taint, and string B is a telephone number "12345678901" marked with taint. Strings A and B are put together to form string C, then string C is tainted with its content "This is a test 12345678901." Then take the first four characters from string C with the new string D "This," string D is also tainted. If string D is sent by privacy leak point, this kind of behavior is called "privacy leak" because D is tainted data. But string D should not be tainted from the whole data process, this is actually a false alarm.

(4) The execution path information cannot be provided in the program testing process. The TaintDroid system cannot provide any path information implemented in the software testing process and cannot distinguish if the implementation of the multiple tests have the same program path, thus leading to a very large blindness.

(5) When testing requires manual participation, automated testing is not conduced. A lot of human resources will be requested in artificial participation, it is a big cost with low efficiency.

2.2 Current detection and defect of privacy disclosure

Current taint analysis mainly has two kinds of information flow: explicit flow and implicit flow [7]. Explicit flow will correspond to program data dependency, i.e. stain information of variable x which will be directly (by assignment or arithmetic) transferred to variable y. Implicit flow corresponds to control and dependence of a program, i.e. stain information of variable x indirectly transfers to variable y by the conditional expression which contains x.

Existing technology of detection of user privacy data leakage according to performing procedures can be divided into two categories: static method and dynamic method. The static method mainly includes control flow analysis, data flow analysis, structural analysis, etc. But because Android programs are basically in Java code, the Android program will be the existence of a large number of implicit function calls (such as virtual function, etc. [8]; these implicit functions are always existed in implicit flow, they are major sources of implicit taint. We will instruct work process of implicit flow.). But with regard to this kind of call, static analysis cannot deal with it effectively. At the same time, through static analysis we can get the specific path of execution procedure of privacy disclosure, but could not confirm whether this path can be executed, only via the dynamic method to validate it is possible.

Dynamic methods include traditional sandbox technology, dynamic taint tracking technology, and so on. Sandbox technology is applied to isolate the operating mechanism of a program, which is currently widely used in the fields of software testing and virus detection. For privacy leak detection using sandbox technology, we need to monitor system to read the user's sensitive information, network communication [9], and other important interfaces. When the interface is called by the program, it can be timely recorded. Through this method, we can well detect if the program reads the user-sensitive information and whether or not the network sent data. As the monitoring point of the system is not continuous, and there is a lack of the necessary logic relationship and contextual

relevance between them, the sandbox technology can not accurately determine whether the program reveals the user's private data.

Different from sandbox technology, dynamic taint tracking technology can continuously track data flow, which effectively solves the defect of sandbox technology of context information lack of association. At present, dynamic taint tracking technology is widely used in vulnerability discovery [10]. When dynamic taint tracking is used to detect privacy data leakage, the privacy data are used as tainted data and then track the spreading. When these detected tainted data are sent through the network, the program can judge the existing privacy problems.

For privacy data leakage behavior in the detection program, dynamic taint tracing is a very effective method [11]. Based on this method, this paper implements a TaintChaser system which could perform fine-grained tracking on information privacy. The system can produce the path information when processing during testing and realizes automatic testing.

2.3 Android system problem and interdependency

(1) Existing Protection Mechanism and Defects of Android System (refer to Appendix A).
Android is an open source operating system based on the Linux system and developed for mobile phone platforms. The Android operating system itself provides a series of mechanism for privacy data protection. Android extends the Linux system which allows each application program to operate in a different identity (i.e. the original UID and GID in Linux), so as to guarantee the relative independence of program-run environment. Android provides a permission system which provides a mechanism that the user resources (mobile phone equipment information, cyber source, etc.) could be controlled if accessed. In this permission system, the important resources in the mobile phone are divided into several types, each type of resource corresponds to an authority; and when the program accesses a type of resource, it could get authorization if it has corresponding permission. But this protective mechanism has great limitation which cannot effectively prevent leakage of user privacy information. The permission for the system resources on the realization of access control granularity is too rough and cannot achieve the desired effect. The implementation of the permission system is not flexible. When a program is installed, the authority is completely determined. When installing a program, for the selection of the program authority, Android also provides two possible ways. One is meeting all the required permissions from procedures or entire negation, rather than letting the user select only part of authority according to their own actual needs [12]. If the program has the permissions it required at the installation stage, in the operational phase it can arbitrarily reveal user privacy while not subjecting to system constraints. In summary, the mechanism of Android itself cannot effectively prevent the program from leaking user privacy data.

(2) Dalvik Virtual Machine
In essence, the Dalvik virtual machine is a Java virtual machine, but it is very different from the general Java virtual machine. The main differences are: (1) the system architecture of the Dalvik virtual machine is different from the Java virtual machine. The Dalvik virtual machine is based on virtual registers and the Java virtual machine is based on stack. (2) The instruction set of the Dalvik virtual machine (Dalvik byte code) is also completely different from the Java virtual machine (Java byte code). The Dalvik virtual machine is the key part of the whole system. In the process of dynamic taint tracking, most taint communication happens in this part.
DEX is a register-based machine language, as opposed to Java byte code, which is stack-based. Each DEX method has its own predefined number of virtual registers (frequently referred to as "registers"). The Dalvik VM interpreter manages method registers with an internal execution state stack. The current method's registers are always on the top stack frame [5, 13]. These registers loosely correspond to local variables in the Java method and store primitive types and object references. All computation occurs on registers, therefore values must be loaded from and stored to class fields before use and after use. Note that DEX uses class fields for all long-term storage, unlike hardware register-based machine languages (e.g. x86), which store values in arbitrary memory locations.

(3) Java Native Interface
Java Native Interface (JNI) is a kind of mechanism of Java program and native code library interaction. Through this mechanism, Java code can easily call other databases written with other languages (typically C and C++). For dynamic taint tracking, when carrying on the JNI call, the code would transfer from the Java layer to the C/C++ layer, it will affect the normal tracking of tainted data which needs to appropriate process to C/C++ library involved.

3 Principle of dynamic taint tracking

3.1 TaintDroid on Android

Figure 1 shows an overview of TaintDroid architecture, so that we can better understand dynamic taint tracking technology. It is also based on Enck's achievement [5, 12, 13].

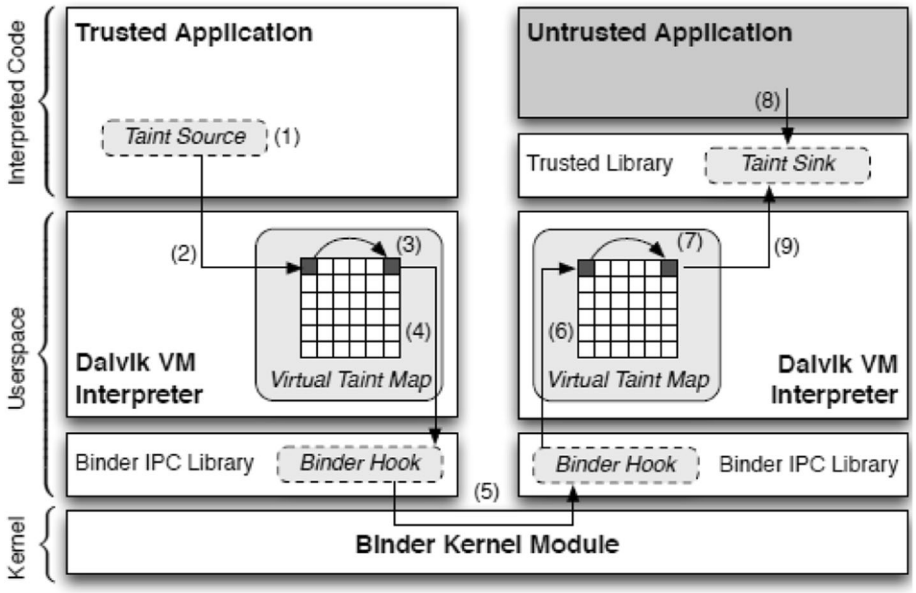

Fig. 1 TaintDroid architecture within Android

3.2 Implicit flow and implicit taint

At the same time, we can look at the implicit flow working process to have a clear view of implicit taint on theory against dynamic taint tracking technology. Then the background of privacy information is very much obvious. The example code of implicit flow is as follows:

Program 1

```
begin
L1    X=get_input();
L2    msg=uri= ‘  ’;
L3    If(x== ‘ a ’){
L4          uri = ‘ post’;
L5          msg = ‘ a ’;
L6    }
L7    else if(x== ‘ b ’){
L8          uri =  ‘ post’;
L9          msg = ‘ b ’;
L10   }
L11   send (uri,msg);
```

Program 2

```
begin
L1    X=get_input();
L2    msg=uri= ‘  ’;
L3    If(x > ‘ a ’){
L4          tmp = x + ‘ a’;
L5          msg = tmp - x;
L6    }
L7    else {
L8          uri =  ‘ post’;
L9          msg = ‘ a ’;
L10   }
L11  send (uri,msg);
```

In the code shown in program 1, program reads and user inputs variables x, then it will generate msg submitted by post methods. Taint x is marked as a tainted attribute, in accordance with the taint analysis method of explicit flow, msg is assigned const {‘a’,‘b’} and marked as an untainted attribute. But msg values depend on T and F values of the judgment of L3 and L7 conditions, that is having control and dependency relationship with x (also known as implicit stream pollution), here will produce a failure alarm, msg should be marked as tainted attribute; and the value of uri is not affected by L3 and L7, labeled as an untainted attribute. The code segment has a security risk because msg is submitted by the post; the attacker can infer the value of x input by the user after capturing msg. According to the explicit taint analysis method, in program 2, the code L4 and L5 msg are tainted by x directly, but the value of msg is a constant ‘a’ after tainted, then it will generate false alarm, msg should also be marked as an untainted attribute.

The judge sentence is abstract expressed as:

$$S0 = e : S1, S2, \ldots Sn,$$

Implicit taint will be existed in the following three conditions:

(1) The conditional expression e of S0 is a tainted attribute.
(2) There is control dependence between judge statement S0 and assignment statements S1, S2…Sn.
(3) There is a difference in assignment value of the same variable in multiple branches of assignment statements S1, S2…Sn.

The three features above are described in two aspects including control flow (corresponding to condition (2)) and data flow (corresponding to conditions (1) and (3)). The implicit taint test, based on taint analysis of data flow, diagnoses implicit taint problems and amends

relevant variables. There may have multiple nested statements in the program (as shown in Fig. 2a). This paper detects variable values in all code blocks of the program point between judging statement and all subsequent assignment statement. If one variable has more than one different value before confluent point (Fig. 2b), the variable should be marked as tainted attribute; if all the values of the variable are the same, it should be marked as untainted attribute.

3.3 Principle of dynamic taint tracking technology

The basic principle of dynamic taint tracking technology is as shown in Fig. 3. Among them, the six lines of programs A and B in the box below represent this program to run within the process. The curve representing the thread does not currently contain tainted data. The dotted line represents the thread exits tainted data and the different dotted line represents the tainted data of different types included.

When detecting that the program reads the privacy information of users, it will mark the privacy data read as tainted. But when the user-sensitive data with taints is operated by the program, we can carry on the corresponding processing of the operation to ensure the taint can follow the privacy data in communication. When communication is on between the two procedures, taints can also follow the data tracking normally. For example, a process of program A (dotted line) sends tainted data to a thread of program B and taints can continually track data transmitted in the thread. When a program sends tainted data to the outside through the privacy leak transmission, it will record the behavior in real-time. Finally, as per the results of the log analysis, we can determine whether there are privacy leakages in the program. If the log shows that tainted data were sent out in the process, we could determine that there is a privacy leakage in the procedure.

3.4 The realization of the taint tracking system

Compared with TaintDroid, the TaintChaser system proposed by this paper could monitor more user privacy information, not only IMEI, phone number, location information, photos, and audio, but also communication, SMS, email, and other important information, detected as pollution point sources. At the same time, the system achieves a more fine-grained taint tracking mode through the detection of each byte in memory content. The system detects more privacy leakage points (socket communication, HTTPS encryption communication, SMS, or Bluetooth communication), gives the path information of the detected program executed in detail, and has automatic tests of procedures [14].

To better understand how to track implicit taint, the following will explain the privacy leakage method. Based on the implicit control flow analysis method of SSA form: (1) in the program control flow graph (CFG), there contains control dependence of code in judging block and assignment statements; and (2) calculate the point

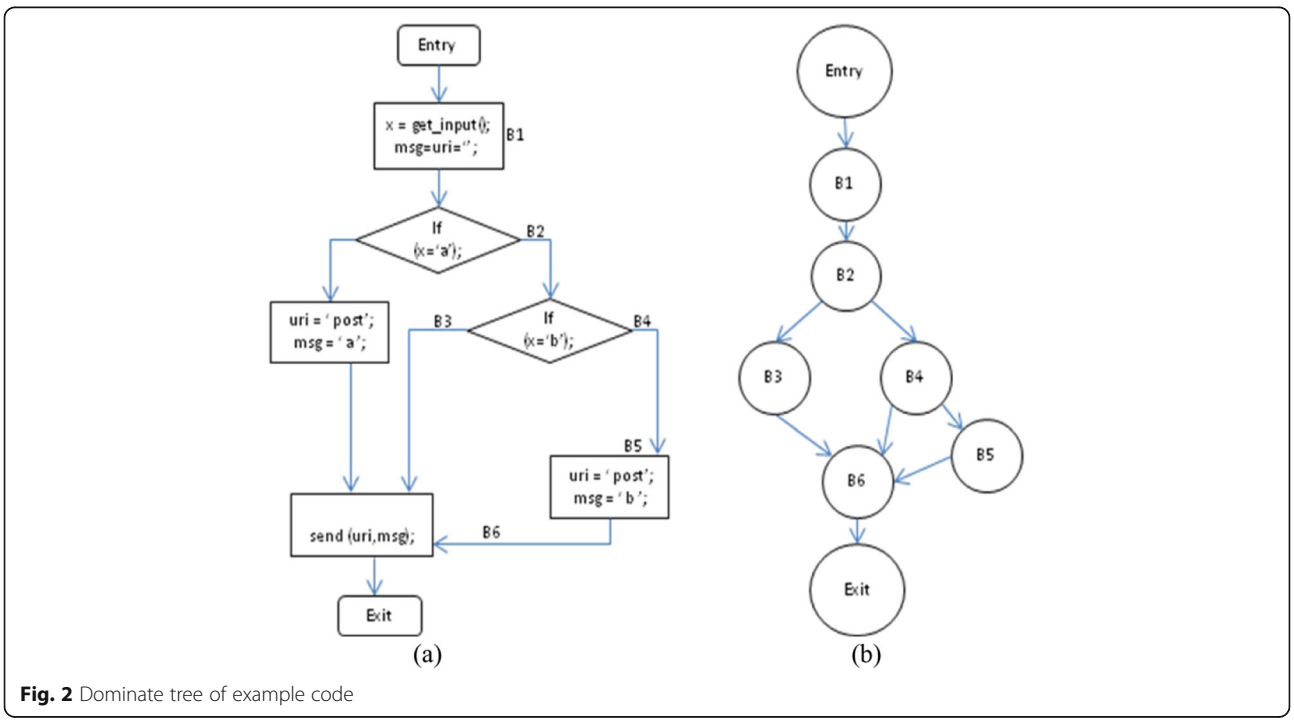

Fig. 2 Dominate tree of example code

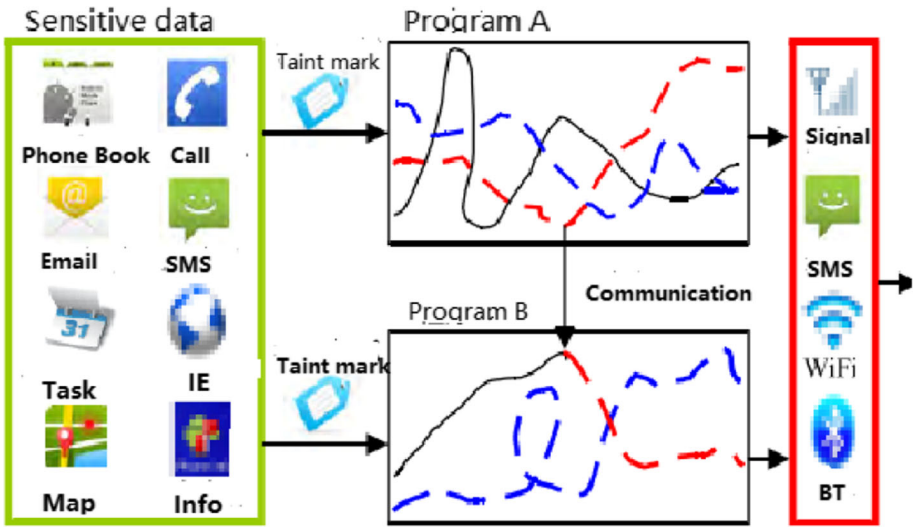

Fig. 3 The basic principle of dynamic taint tracking technology

of code block of assignment statement and convert the program number flow into the form of SSA, in point count value of multiple versions of code block variables, finally according to the value of each variable to determine taint attribute of variables.

The reason for implicit taint is that there is control dependency between assignment statements and judge statements. The control dependency of a program can be expressed as: if the execution of statement S2 determines whether statement S1 executes, statement S1 control is dependent on statement S2. In this paper, we use a dominate tree to find this kind of dependence. First, the CFG of the program is extracted, the dominate tree is calculated, and then the control dependency between judgment and assignment statements from the tree is obtained. Definitions are given as follows.

(1) Control dependency

Immediate dominate: for $x \neq y$, dominate x is an immediate dominate of y, if x is an immediate dominate of y, and there is no dominate z, such that x is the only immediate dominate of z and z is the immediate dominate of y, marked as idem (y).

Dominate tree: contains each dominate of CFG in the tree, and for each dominate x, there is a path from idom (x) to the x side, because each dominate has an exactly immediate dominate, the tree known as the dominate tree.

The paper uses the dominate tree of depth-first search tree calculation program to CFG [15]. The complexity of the method is similar to linear time. The dominate tree of CFG describes the control dependency of the program, as shown in Fig. 2b. Judgment dominate B2 is an immediate dominate of assignment dominates B3 and B4 and judgment dominate B4 is an immediate dominate of B5 at the same time. That is {B3, B4} control and rely on B2 and {B5} controls and relies on B4.

(2) Path convergence criterion

Implicit taint is caused by control dependence and there is assignment difference in the branch of judgment statement; the assignment differences have an impact on the following variables in branches merging. As shown in Fig. 2a in the sample code, judgment dominate B4 is nested in judgment dominate B2, only consider B4 cannot correctly respond to the effects of variables assignment; in fact, the paths of {B2, B4} converge to B6. Therefore, in order to analyze the effects of the judgment statement of variables, we shall examine the path confluence of judgment statements. The following is a definition of convergence criterion of the program path: when the program path is to meet all of the following conditions, dominate n is the confluence of the variable a.

1) There is a code block x that contains an assignment a.
2) There is a code block y($y \neq x$) that contains an assignment a.
3) There is a non-empty path Pxn from x to n.
4) There is a non-empty path Pyn from y to n.
5) There is no other path in common between path Pxn and Pyn other than dominate n.
6) Dominate n does not appear in these two paths before the point of path Pxn and Pyn, but it can occur in each path.

By the above criteria, there are one-to-one mapping relations between the assignment and path of variable A. There are a number of precursors a of convergence n, then there are many kinds assignment of variable A.

(3) Discovery of path convergence

It is not practical if we use the convergence criterion method directly to discover the convergence because of the need to traverse all the way from the dominate x and y to the confluence n. To use the dominate tree to decide dominate boundary of judgment statement code block, it can be found that convergence of judgment sentence efficiently.

(4) Virtual value function

Variables value analyze and use static single assignment (SSA) of the program in program convergence. SSA forms an intermediate presence. In the process, each variable has only one assignment, the static assignment may be located in a cycle which can be dynamically executed many times, so it is called SSA. The SSA form is not associated with the application form for the same variable to change into different variables and two control flows in control flow chart merge into a virtual value function ϕ, e.g. dominate n is as the variable a inserted into function ϕ (a1, a2), used to distinguish multiple assignments of variable a. The characteristics of value function are as follows: the number of parameters of function are the same with the possible value number of the variable. Each parameter corresponds to the precursor with a particular control flow. According to the above characteristics, all values of variables for the convergence can be acquired through calculation of the parameter of virtual value function. The calculation process is as follows: it is from the CFG of the program and completes the import of function ϕ and parameter calculation of function ϕ [16]. The value of using virtual value function ϕ in SSA is so we can judge whether the variables are the same, so as to determine the taint attribute [17]. Based on the knowledge of the above method, let us review Taint tracking and TaintChaser.

3.5 Taint data

The TaintChaser system can detect most user privacy information, identification numbers of mobile phone equipment, phone numbers, location information, e-mail, contacts, SMS, schedule, and browser history. Because we obtained this privacy information in different ways, we need to treat them separately as taint marks. The related service process by the Android system supplies the identification number of mobile phone equipment, phone number and location information [5, 17], we need to process the service as process-related. The taint marked process is shown in Fig. 4. At the time of data reading, the program is as a client through binder sends out a data requested to the corresponding service process; server side acquire privacy data; TaintChaser system will mark these data as taint, so as to ensure the data of the process got taint labeled.

Email, contacts, SMS, schedule, and browser history have been stored in the database. For the taint marked data, the behavior of the reading database will be processed. Among which, the general process of data reading from the database is first we need to get the database cursor, then by using the cursor to call correlation function (get String) to read the contents of the database. The taint marked process of user-sensitive information stored in the database is shown in Fig. 5, when the program obtains the storage of the user privacy information database cursor, the cursor will be taint marked.

Later in the running program, if the program detects the specific content by the tainted cursor, all data read are marked as taint. In this way, we can realize the taint mark on email, short messages, and other privacy information stored in the database.

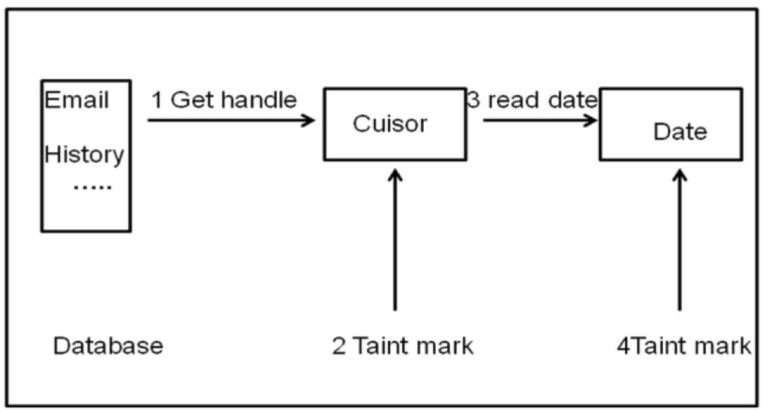

Fig. 4 Taint marked process of TaintChaser I

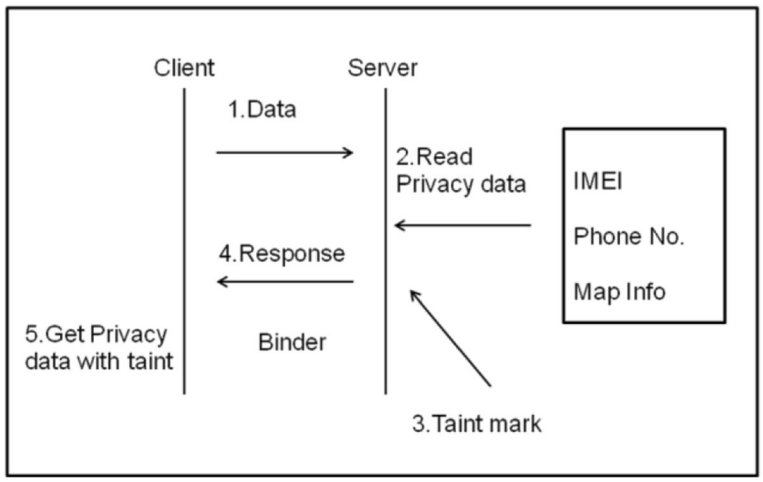

Fig. 5 Taint marked process of TaintChaser II

3.6 Taint storage

Different from TaintDroid taint storage, TaintChaser uses a fine-grained way to describe taint, each byte in memory corresponds to a taint. In this way, in the tracking of tainted data, we can track more accurately to reduce greatly the probability of excessive diffusion of taint. The storage structure of the taint as shown in Fig. 6; the Android system is 32 bit, taint storage table has two level address index organizations. This storage method can mostly save memory space and fast lookup taint of corresponding memory [18].

3.7 Taint tracking

The Android program is executed by parsing the Dalvik virtual machine. TaintChaser system achieves the tainted data tracking by processing all instructions in the Dalvik virtual machine. This is processed by the following three aspects:

(1) General instructions mainly comprise a series of basic operations of variables (the assignments and add, subtract, multiply, and divide, etc.) and function call. For the basic operating of variables, if operands data involved contain taint, the final result will mark taint. In the function call, tainted data may be spread by input parameters and return values of function. The instruction of invoke and return need to be processed.

(2) The file read and write refers to the detection of file read and write. When tainted data are written to a file it will mark taint to the file; and when detected the contents with taint of the file be read out. It also can be processed in real time.

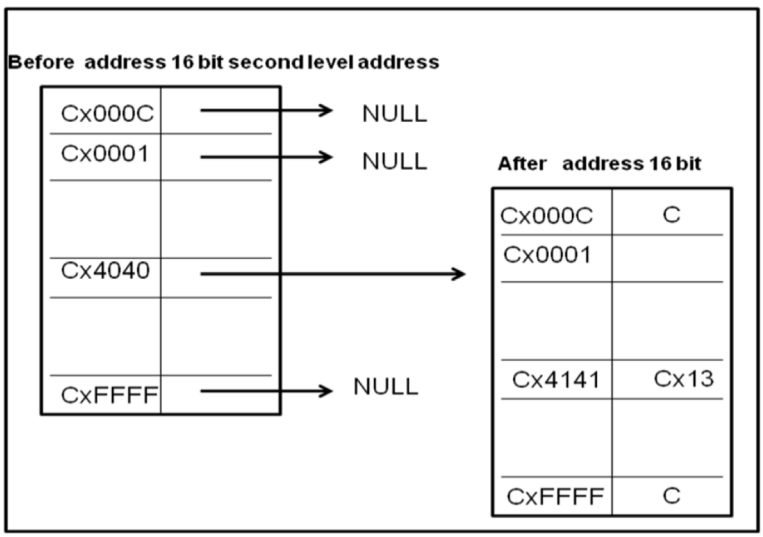

Fig. 6 Taint storage architecture of TaintChaser

(3) Inter process communication: when detecting the inter process communication, if tainted data are included in the communication content, real-time tracking can disseminate tainted data to ensure taint can continue to follow the data communication in the new process. But because of the JNI mechanism in the Android applications process, the program will break away from the Dalvik virtual machine to enter the local C/C++ database, the system cannot carry on the normal taint tracking [19, 20]. In order to realize taint on normal communication, we have to deal with it by the related hook function.

3.8 Privacy leak point

Privacy leakage detected by the TaintChaser system has socket communication interface, Https encryption communication interface, Bluetooth, short message, etc. For each type of privacy leakage, because interface of data sent is not the only one we need to process all the function related, first, according to the length and its memory address of transmitted data, check in taint storage table whether the data is tainted data, if it is, it will be recorded including content and destination address of sent data, etc. other related information [21, 22].

3.9 The output of path information of testing program execution

The output of path information of testing program execution is achieved through the function call instruction (invoke) in the Dalvik virtual machine. With the execution of the invoke command, the class and method name will be printed out. It contains the procedure call and prints out the information.

But because testing the program will call a large number of system functions in the implementation process (such as interface rendering, inter process communication, event processing, and so on), the printed result not only includes the execution path of testing the program, but also a lot of information about system function call. Therefore, we need to add the filtering mechanism to filter the call information of system function and other useless information. Because the system function and software class name are different, it can be filtered based on the name of class of call function [23, 24].

First, before the program is tested, we need to extract the relevant class of tested program and import it to the specified configuration paper; then, in the test when the invoke command is executed, the content will be matched with configuration file name with calling function. If matched, the related information of the called function will output or be discarded directly; it can eventually get the path information of the tested program which has been executed.

3.10 Automated testing program

The automated testing program needs to have automatic setup to the system and can automatically start the process in the system and, during the execution of the program, automatically sends a series of events. To achieve this, the automatic installation and start-up can use the tools adb and am provided by the Android system. Before testing the program automation, we need to do some pre-processing. First, testing procedures have to be decompressed, by reading related program information from the file AndroidManifest.xml. Then, according to the file directory structure of the program, that program name is speculated, and the class name is organized into files in the specified directory, so as to output the path information of the testing program executed. After these preparations are done, we can use the adb tool to install the testing program on the system and begin the test. In the testing process, we use the am command to send a series of events to the program.

4 Implementation of automatic test system, test results, and evaluation

4.1 Development environment

The automatic test system based on TaintChaser is implemented in the security threat monitoring platform of mobile Internet, which provides a relevant interface that could complete automated testing and has a large number of real samples of Android application software for analysis.

The TaintChaser system is processed in the simulator provided by the Android system. The simulator has the same performance problems as a real mobile phone. This is mainly because, in the course of program execution, the simulator needs to spend extra time in transforming the ARM instruction into the x86 instruction. But because the simulator can operate in the general PC, it can pass through the massively parallel deployment to remedy the defects on the properties. TaintChaser is the massive parallel deployment based on the KVM virtual machine, through a unified dispatching system to carry out the parallel detection [25].

4.2 Carrying out the automatic test system

The automated test system structure based on TaintChaser is shown in Fig. 7. It mainly consists of a host machine (real computer) installed with the KVM virtual system and a series of virtual machines (kvm#1, kvm#2). The virtual machines kvm#1 and kvm#2 are used as privacy leakage problems of the automatic detection program and clone a series of complete test environment in the host machine. The system consists of three modules: Schedule, TaintChaser, and Restore. Among them, the Schedule module runs in the virtual machine and is mainly used for the virtual machine to register the security threat monitoring platform of the mobile Internet and get the

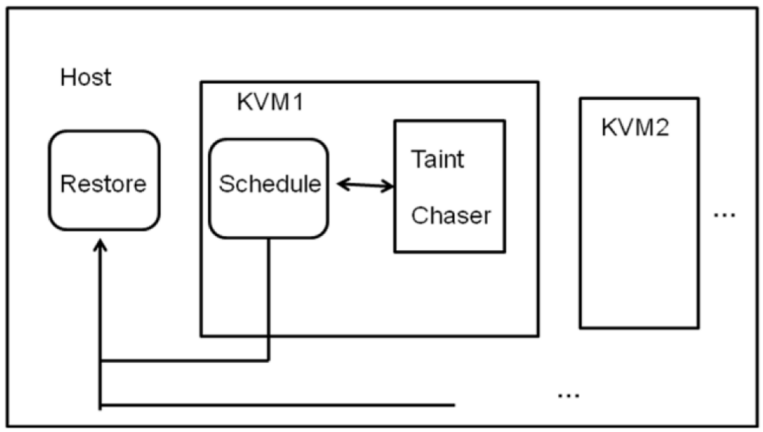

Fig. 7 Automatic test architecture of TaintChaser

test tasks and the testing program from the platform. It will be handed over to the TaintChaser program to be analyzed and then send the processed results back to the monitoring platform. TaintChaser runs in the virtual machine with complete detection to the automatic program. The Restore module runs on the host machine, mainly responsible for recovery of the KVM virtual machine completed test to its original clean working environment [26, 27].

Taking virtual machine kvm # 1 of the testing system as an example, the entire workflow is as follows:

(1) When the testing system starts, the Schedule module is automatically registered to monitor the mobile Internet platform for security threats. Then it will wait to monitor the platform for the allocation of detection tasks.
(2) When the Schedule module receives the task, according to the content of the task (program ID, program location), it downloads the procedures to be tested from a specified location to the local and hands the program over to TaintChaser for analysis.
(3) After detection of the program, TaintChaser will hand the detected print log down to the Schedule module.
(4) The Schedule module will analyze the log obtained and send the results of the analysis back to the monitoring platform.
(5) The Restore module called by schedule on host recover virtual machine kvm#1 to original work environment. Then re-steps (1), to begin a new round of testing.

4.3 Test results and valuation

The test sample is provided by the security threat monitoring platform of the mobile Internet, mainly from each Android electronic market in China. The concrete condition is shown in Fig. 8; the sample is obtained chiefly from the electronics market appchina.om, eoe marke.com, goapk.com, etc.. The total sample number is 58,468.

The use of the automated testing system of TaintChaser was analyzed and detected the large-scale automation based on these Android sample programs. It finally found that 19,592 (33.51 %) programs may leak user privacy information. The information is shown in Table 1.

We then randomly took out 100 pieces of software from the leaked privacy information to analyze manually. The reasons for the leakage of private information have been analyzed more deeply [28] (http://www.google.com/mobile/products/maps.html).

(1) IMEI
Of the 100 test software, 85 pieces of software will send the IMEI to the advertising service provider and the remaining 15 will send the IMEI to the respective software developers. The user's privacy information that the software sends to the advertising service provider, mostly due to software developers for the purpose of profit, is from embedded advertisements in the software. Inserting advertisements in the software is accomplished by inserting corresponding advertising of the third party in software code.
(2) Telephone number
Thirty-nine tested pieces of software will send the telephone number to the advertising services, while the remaining 11 pieces of software will send it to the respective software developers.
(3) Location information
Among the 100 test pieces of software, 24 belong to the normal application, 50 will send location

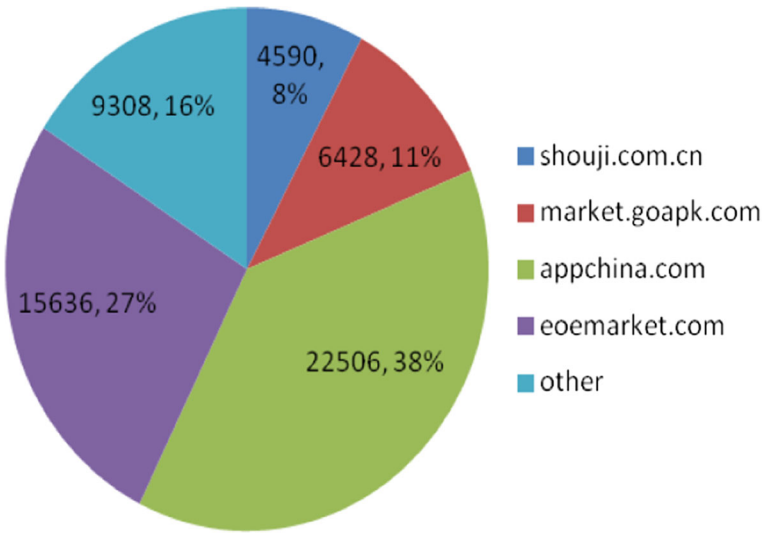

Fig. 8 Android Sample source of test program

information to the advertising service provider, and the remaining 13 will send location information to the corresponding software developers. Of the 24 pieces of software in normal use of location information, 10 provide the normal map service and the other 14 use the GPS position coordinates to find food, movies, etc. nearby.

(4) Serial number

Different from the above privacy information, from the 100 test pieces of software, none sends the serial number to the advertising service provider, but all send to the respective software developers.

Software privacy leakage information is shown in Table 2.

The privacy leakage of the domestic software of each Android electronic market is then analyzed statistically. The results are shown in Table 3. We find that each domestic electronics market has software that leaks user privacy information.

At the same time, through the automatic detection system, we also found third party plug-ins that may leak privacy information of users. Take malicious plug-in "com.nl" as an example; the plug-in will start running when the mobile phone is started, first send a request command to a botnet, and then some corresponding processing would be processed according to the command, for instance, automatically download and install some popular software from a specified server (e.g. QQ, Renren client), send fee deduction SMS, etc. The plug-in will intercept the address which is "10658166" or the content including "83589523," "customer service call," "1RMB/msg," "1RMB/times," "1 RMB," and "2 RMB" strings of short message. Among them, with the botnet communication process, the plug-in will leak the user's phone number and IMEI information.

We randomly selected 50 pieces of software from the samples (0.18 % of the total), which were detected by TaintDroid and TaintChaser and we found that 29 may

Table 1 Program statistics showing users' privacy leaks

Type of privacy leakage data	Program QTY	Percentage (%)
IMEI	13,783	22.57
Phone number	3651	6.24
Location information	1779	3.04
Serial number	379	0.6
SMS	15	–

Table 2 Software privacy leakage information

Information type of privacy leakage	Destination	Domain name	Software QTY
IMEI	Advertisement facilitator	youmi.net	45
		wiyun.com	30
		mobclix.com	7
	Software developer		
	Developer	-	15
Phone number	Advertisement facilitator	oumi.com	47
		sosceo.com	34
	Software developer	-	19
Location information	Advertisement facilitator	youmi.net	24
		lsense.cn	18
		mobclix.com	8
	Software developer	-	13
Serial no.	Software developer	-	100

Table 3 The software number of privacy leakage of electronics market

E-market name	Software QTY	Privacy leakage QTY	Percentage (%)
goapk.com	6428	2440	37.96
eoemarket.com	15,636	5754	36.80
appchina.com	22,506	4332	19.25
shouji.com.cn	4590	491	10.70

have private information leakage detected by TaintDroid and 31 detected by TaintChaser.

For software privacy leakage problems in a domestic electronics market, the reasons can be summarized as follows [29, 30]:

(1) Because of the domestic Android software developers' lack of awareness of the protection of user privacy data, software will often reveal the user's privacy data.
(2) When part of the foreign Android software was "localized," it was with some third party plug-ins (such as advertising plug-ins), which may lead to leakage of user privacy data. For example, for the same version of the game "Angry Birds," there is no revealing user privacy information in English version, while the "localized" version provided by the domestic electronics market has ad mob advertisement plug-ins which would leak user location [31].
(3) In order to collect user information and analyze user behavior in some electronics markets, they will implement third party plug-ins to their own market in popular software in which may reveal user privacy information.
(4) There is a lack of standardized management in the domestic Android electronics market and no system or mechanism to deal with software leakage [32].

5 Conclusions

While some mobile phone operating systems allow the control of applications' access to sensitive information, such as location sensors, camera images, and contact lists, users lack visibility into how applications use their privacy data. With the rapid development of the Android mobile phone platform, privacy problems in Android software is becoming more serious. In this paper, based on dynamic taint tracking technology TaintChaser (it is improved from TaintDroid), it is proposed and demonstrated that automatic inspection system, scale, and the privacy problems of software in the domestic electronics market were detected and analyzed by the system. The next step is to optimize the system, improve the working efficiency of the system, and analyze in depth the behavior patterns of user privacy data leakage.

6 Appendix A

6.1 Android system architecture

Architecture of [33] Android operating system, as shown in Fig. 9.

The top layer is the implementation of the Java program functions; the next layer is a Dalvik virtual machine; it is Google-specific for the Java interpreter for the design and development of the Android as the Java program can run on its interpretation. The middle layer is the Native Interface, it is by the upper Java procedure call native code library; Binder layer is the communication mechanism of a lightweight Android system which provides inter process; the bottom is modified through Linux system, provides the interface to interact with the mobile phone hardware. Android contains two types of native methods: internal VM methods (see "Appendix B") and JNI methods (see "Appendix C"). The internal VM methods access interpreter-specific structures and APIs. JNI methods conform to JNI standards specifications, which requires Dalvik to separate Java arguments into variables using a JNI call bridge. Conversely, internal VM methods must manually parse arguments from the interpreter's byte array of arguments.

7 Appendix B

7.1 Internal VM methods

Internal VM methods are called directly by interpreted code, passing a pointer to an array of 32-bit register arguments and a pointer to a return value. The stack augmentation provides access to taint tags for both Java arguments and the return value. As there are a relatively small number of internal VM methods which are infrequently added between versions, we manually inspected

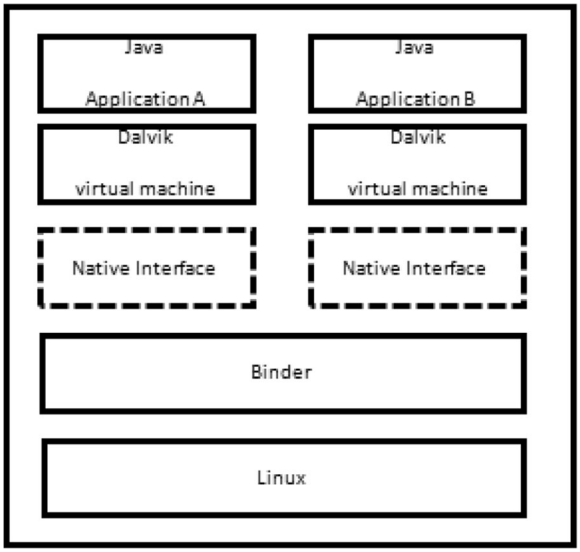

Fig. 9 Architecture of Android operating system

and patched them for taint propagation as needed. We identified 185 internal VM methods in Android version 2.1; however, only five required patching: the System. arraycopy() native method for copying array contents and several native methods implementing Java reflection.

8 Appendix C

8.1 JNI methods

JNI methods are invoked through the JNI call bridge. The call bridge parses Java arguments and assigns a return value using the method's descriptor string. We patched the call bridge to provide taint propagation for all JNI methods. When a JNI method returns, TaintDroid consults a method profile table for tag propagation updates. A method profile is a list of (from, to) pairs indicating flows between variables, which may be method parameters, class variables, or return values. Enumerating the information flows for all JNI methods is a time-consuming task best completed automatically using source code analysis (a task we leave for future work). We currently include an additional propagation heuristic patch. The heuristic is conservative for JNI methods that only operate on primitive and String arguments and return values. It assigns the union of the method argument taint tags to the taint tag of the return value. While the heuristic has false negatives for methods using objects, it covers many existing methods.

We performed a survey of the JNI methods included in the official Android source code (version 2.1) to determine specific properties. We found 2844 JNI methods with a Java interface and C or C++ implementation. Of these methods, 913 did not reference objects (as arguments, return value, or method body) and hence are automatically covered by our heuristic. The remaining methods may or may not have information flows that produce false negatives. Currently, we define method profiles as needed. For example, methods in the IBM Native Converter class require propagation for conversion between character and byte arrays.

9 Appendix D

9.1 TaintDroid

TaintDroid is a realization of our multiple granularity taint tracking approach within Android. TaintDroid uses variable level tracking within the VM interpreter. Multiple taint markings are stored as one taint tag. When applications execute native methods, variable taint tags are patched on return. Finally, taint tags are assigned to parcels and propagated through binder. Note that the Technical Report [17] version of this paper contains more implementation details.

Figure 1 depicts TaintDroid's architecture. Information is tainted (1) in a trusted application with sufficient context (e.g. the location provider). The taint interface invokes a native method (2) that interfaces with the Dalvik VM interpreter, storing specified taint markings in the virtual taint map. The Dalvik VM propagates taint tags (3) according to data flow rules as the trusted application uses the tainted information. Every interpreter instance simultaneously propagates taint tags. When the trusted application uses the tainted information in an IPC transaction, the modified binder library (4) ensures the parcel has a taint tag reflecting the combined taint markings of all contained data. The parcel is passed transparently through the kernel (5) and received by the remote untrusted application [34]. Note that only the interpreted code is untrusted. The modified binder library retrieves the taint tag from the parcel and assigns it to all values read from it (6). The remote Dalvik VM instance propagates taint tags (7) identically for the untrusted application. The untrusted application invokes a library specified as a taint sink (8), e.g. network send, the library retrieves the taint tag for the data in question (9) and reports the event.

Implementing this architecture requires addressing several system challenges, including: (1) taint tag storage; (2) interpreted code taint propagation; (3) native code taint propagation; (4) IPC taint propagation; and (5) secondary storage taint propagation. The remainder of this section describes our design.

Acknowledgements

First, I would like to thank my doctoral tutor (Professor Jin ZhiGang) for his valuable suggestions and productive discussion. I would like to thank Tianjin University Broad Band & Wireless Communication Labs, Peter Li, and Zhang Qian for their support and feedback during the design and prototype implementation of this paper. I thank Jim Yao and Tom Wang for their feedback during the writing of this paper. I also thank Kevin Chen, Stephen Ma, Mac Li, and the Tianjin University Mobile Internet design team as a whole for their helpful comments. This paper is based upon work supported by a lot of reference paper and their study. This paper is partially supported by my wife who made sure of all translations and grammar—I also thank her hard work. Finally, I thank my family as it is based on their full support so that I could finish this paper.

Competing interests

The authors declare that they have no competing interests.

About the authors

Hou Rui is currently a Ph.D. candidate at Tianjin University, Tianjin, China. In June 2005, he received a Master's degree in software engineering from NanKai University, Tianjin, China. He has been a PMP member since 2013. His research interests include mobile platform security, wireless networks, and resource management in heterogeneous networks. His major representative academic achievements: Paper "Forecast of the fourth generation mobile communication technology (4G)", City construction, 2010.05; "Discussion on the key technology of communication middleware", City construction, 2010.05; "Mode selection algorithm based on fast frame stereo video compression of H.264", The research and application of computer, volume 27, 2010; "Security Mechanism Analysis of Open-Source", Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference; "Comparison of Windows Phone 8 Windows 8", issue on processing; "Application research and analysis based on Bitlocker", issue on processing.

Jing ZhiGang is a Ph.D. (post), Professor, Expert consultation and decision-making of Tianjin Government. He was the chief engineer at the network center of Tianjin University and deputy director of Computer Department of Tianjin University. He is now the professor of communication at the engineering

college of Tianjin University. In June 1996, he received his Master's degree in computer system structure from Tianjin University. In June 1999, he received a doctorate of engineering signal and information processing from Tianjin University. In March 2001, he received a post-doctoral in electrical engineering from Tianjin University. In 2002, he was visiting scholar at Ottawa University of Canada. His research interests include wireless networks, the technology of the Internet of things, network security, and network management. His major representative academic achievements: Paper "Non-Gaussian characteristics of traffic and the impact to network performance", SCI EI, Journal of Comp. Sci and Tech and "User-oriented network behavior prediction and control", SCI EI Chinese Journal of Electronics. Book "Computer network", major editor, Xi'an Electronic and Science University Press, 2009.
Scientific research. Completed projects: the National Development and Reform Commission, a CNGI project, National 863 plan project, two projects; National Natural Science Fund project, four projects, one was named the 20th anniversary of the establishment of the National Natural Science Fund for outstanding achievements. These are in addition to completing the Tianjin city and other provincial and ministerial projects
These projects are ongoing: one project from National Natural Science Foundation, one project from the National 863 plan project, multi-projects from Tianjin Municipal Science and Technology Commission
Wang BaoLiang received his Master's degree in computer engineering from Shandong University. In June 2010, he received a doctorate of Communication and information system from Tianjin University. He has issued a paper, "Mode selection algorithm based on fast frame stereo video compression of H.264", The Research and Application of Computer, volume 27, 2010.

Received: 31 May 2016 Accepted: 26 August 2016
Published online: 22 September 2016

References

1. Android, http://www.android.com. Accessed date 6 Feb 2016.
2. Android Market, http://market.android.com. Accessed date 6 Feb 2016.
3. Apache Harmony – Open Source Java Platform, http://harmony.apache.org. Accessed date 18 Feb 2016.
4. AR Beresford, A Rice, N Skehin, R Sohan, *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications* (ACM, New York, 2011), pp. 49–54
5. W Enck, P Gilbert, B-G Chun, LP Cox, J Jung, P McDaniel, AN Sheth, *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones Tech. Rep. NAS-TR-0120-2010* (Pennsylvania State University, University Park, 2010).
6. P Hornyack, S Han, J Jung, S Schechter, D Wetherall, *CCS'11 Proceedings of the 18th ACM Conference on Computer and Communications Security* (ACM, New York, 2011), pp. 639–652
7. EJ Schwartz, T Avgerinos, D Brumley, *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (ACM, New York, 2010), pp. 125–132
8. W Enck, D Octeau, P McDaniel, S Chaudhuri, *Proceedings of the 20th USENIX Conference on Security* (USENIX Association, Berkeley, 2011), p. 21
9. I Goldberg, D Wagner, R Thomas, EA Brewer, *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography – Volume 6* (USENIX Association, Berkeley, 1996), p. 1
10. A Sabelfeld, AC Myers, Language-based information-flow security. IEEE J. Selected Areas Commun. **21**(1), 5–19 (2003)
11. GE Suh, JW Lee, D Zhang, S Devadas, *ASPLOS XI Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, ACM, 2004), pp. 85–96
12. W Enck, *Proceedings of the 7th International Conference on Information Systems Security* (Springer-Verlag, Berlin, 2011), pp. 49–70
13. W Enck, M Ongtang, P McDaniel, *Proceedings of the 16th ACM Conference on Computer and Communications Security* (ACM, New York, 2009), pp. 235–245
14. EJ Schwartz, T Avgerinos, D Brumley, *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (ACM, New York, 2010), pp. 317–331
15. SK Nair, PN Simpson, B Crispo, AS Tanenbaum, Electronic Notes Theor. Comput. Sci. **197**(1), 3–16 (2007)
16. A Ho, M Fetterman, C Clark, A Warfield, S Hand, *EuroSys '06 Proceedings of the 1st ACM SIGOPS/Eurosys European Conference on Computer Systems 2006* (ACM, New York, 2006), pp. 29–41
17. W Cheng, Q Zhao, B Yu, S Hiroshige, *IEEE Symposium on Computers and Communications (ISCC)* (IEEE, Washington, DC, 2006), pp. 749–754
18. J Ligatti, L Bauer, D Walker, Edit automata: enforcement mechanisms for run-time security policies. Int. J. Inf. Secur. **4**(1–2), 2–16 (2005)
19. J Newsome, D Song, in *Proceedings of the 12th Annual Network and Distributed System Security Symposium* (NDSS, San Diego, 2005), pp. 17-24
20. T Bao, Y Zheng, Z Lin, X Zhang, D Xu, *Proceedings of the 19th International Symposium on Software Testing and Analysis* (ACM Press, New York, 2010), pp. 13–24
21. P Gilbert, B Chun, LP Cox, J Jung, *Proceedings of the second international workshop on Mobile cloud computing and services* (ACM, New York, 2011), pp. 21–26
22. J Clause, W Li, A Orso, *Proceedings of the 2007 international symposium on Software testing and analysis* (ACM, New York, 2007), pp. 196–206
23. J Newsome, D Song, in *Proceedings of the 12th Annual Network and Distributed System Security Symposium* (NDSS, San Diego, 2005), pp. 56-62
24. Y Zhu, J Jung, D Song, T Kohno, D Wetherall, *Privacy Scope: A Precise Information Flow Tracking System for Finding Application Leaks* (University of California, Tech. Rep.: UCB/EECS-2009-145, Berkeley, 2009)
25. J Newsome, D Song, in *Proceedings of the 12th Annual Network and Distributed System Security Symposium* (NDSS, San Diego, 2005), pp. 101-124
26. N Nethercote, J Seward, *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation* (ACM, New York, 2007), pp. 89–100
27. T Wang, T Wei, G Gu, W Zou, *Proceedings of the 2010 IEEE Symposium on Security & Privacy* (IEEE Computer Society, Washington, DC, 2010), pp. 497–512
28. Google Maps for Mobile. Accessed date 1 March 2016.
29. Apple, INC. Apples App Store Downloads Top Three Billion. http://www.apple.com/pr/library/2010/01/05Apples-App-Store-Downloads-Top-Three-Billion.html, January 2010. Accessed date 5 March 2016.
30. AW Appel, *Modern Compiler Implementation in C* (Cambridge University Press, Cambridge, 2004)
31. MG Kang, S McCamant, P Poosankam, D Song, in *Proceedings of the Network and Distributed System Security Symposium* (NDSS 2011) (NDSS, San Diego, 2011), pp. 56-70
32. A Yip, X Wang, N Zeldovich, MF Kaashoek, *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (ACM, New York, 2009), pp. 201–304
33. P Zeng, YJ Shao, Android System Architecture & Application Study. Comptr. Inform. **27**(9), 1–3 (2011)
34. Flurry Mobile Application Analytics, https://developer.yahoo.com/analytics/. Accessed date 6 March 2016.