國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

藉系統層的資訊流動追蹤以偵測 Android 平台上竊取敏感資料的行為

DroidTracking : Detecting Sensitive Data Stealing on Android with System-Wide Information Flow Tracking

研 究 生：蘇修醇　　Student：Hsiu-Tsun Su

指導教授：謝續平　教授　　Advisor：Dr. Shiuhpyng Shieh

中華民國一百年八月

藉系統層的資訊流動追蹤以偵測 Android 平台上竊取敏感資料的行為

研究生: 蘇修醇　　指導教授: 謝續平 教授

國立交通大學

資訊科學與工程研究所

摘要

Lookout Mobile Security(手機防毒公司)指出，Google Android Market 上已超過 50 個以上的應用程式被發現遭注入 DroidDream 惡意程式，DroidDream 送出大量敏感資料到遠端伺服器上，而它是第一個被發現到具有攻擊並利用 Android 作業系統漏洞能力的惡意程式。爲了要準確分析 malware，我們藉修改虛擬的 ARM CPU，提出具有系統層、精確性的資訊流動追蹤能力的 DroidTracking 分析工具，以虛擬機器爲基礎的 DroidTracking 可分析整個 Android 作業系統以了解關於竊取敏感資料的行爲，不同於以往的分析工具， DroidTracking 藉分析系統層的資訊可避免欲分析的資訊已遭受惡意程式所影響，再對系統物件做 byte-level 的分析可達到更精確的資訊流動追蹤。我們的實作包含追蹤 GPS, IMEI, IMSI 和 ICC-ID，未來也將追蹤更多手機上的敏感資料，而實驗中，我們蒐集大量已被 DroidDream 感染的已知應用程式，並用 DroidTracking 做分析，可成功的偵測並證實被感染的應用程式正在竊取敏感資料的事實。

DroidTracking : Detecting Sensitive Data Stealing on Android with System-Wide Information Flow Tracking

Student: Hsiu-Tsun Su Advisor: Dr. Shiuhpyng Shieh

Department of Computer Science

National Chiao Tung University

Abstract

A large number of Android applications injected with DroidDream malware have been found on the Google Android Market by Lookout Mobile Security. According to Lookout, DroidDream sends a variety of sensitive data to a remote server. It is the first malware that exploits vulnerabilities of the Android operating system (Android OS). To cope with the problem, we propose DroidTracking, a system-wide and fine-grained information flow tracking system with emulated ARM CPU. DroidTracking analyzes the entire Android OS to detect sensitive data stealing behaviors. Unlike the conventional operating system call tracking schemes, our VM-based, system-wide analysis can avoid malware interference, and its fine-grained information flow tracking supports accurate byte-level system objects analysis. DroidTracking has been implemented to track sensitive information leakage, such as GPS, IMEI, IMSI and ICC-ID. To evaluate the DroidTracking, we collected a number of popular Android applications infected with DroidDream. Our experiment showed that the infected applications's behaviors of stealing sensitive data can be accurately identified and detected.

誌謝

誠摯感謝於去年度甫榮獲 American ACM Distinguished Scientist 的指導教授謝續平老師，學生我深感老師兩年來的悉心指導，從老師所給予的實驗室計畫中逐步累積經驗，及實驗室會議上的討論和指點，直到嚴謹的論文要求，學生步步地從中學習點滴，在老師細心地循循善誘之下，不才學生我方能在新知的領域中學習且進步，進而理解新知領域中的深奧，如今，已能完成此篇論文爲新的領域做出貢獻，這些蛻變都得再次感謝謝續平教授指導的一切。

本論文的完成更得感謝 DSNS (Distributed System and Network Security) Lab 的各位，因爲學長、姐的指導與激發、同儕的激勵和分享，及學弟妹的支持與聆聽，此篇論文才能更趨完善。

感謝王繼偉學長，無論在研究方向和實作技術上的指導均令我感受良多，若非學長辛苦又耐心的指導一切，我無法獨力完成此篇論文。再次感謝。

感謝 Mashi 學長，給予我無數想法和思考上的啓發，是此篇論文的活力與點子來源之一，也因此學長總是我每次會議後必問的對象，每次的激發都令我學習不少，屢屢的討論更爲此篇論文提出不少解決的辦法，而每次的細節都令我銘謝在心。

感謝 Sky、Michael、Pokai、Vic 等學長，每次會議後的討論都令我了解更多自己的不足，從中所獲得的寶貴經驗，總是成爲論文進步的原因，也是令我成長的元素。

感謝嘉偉、江江、蘋果、峰哥，兩年來的彼此激勵，都成爲彼此繼續進步的動力。亦感謝實驗室的學弟妹們、助理們和各位，兩年來的日子，實驗室歡愉的氣氛總令人忘卻研究的辛苦，大家不僅爲實驗室帶來了歡樂，更帶來了榮譽，我將爲曾經身爲 DSNS Lab 的一份子而驕傲。

最後，謹以此榮譽獻給我的家人，特別是我的爺爺，感謝您在我踏入交大前最後的祝福和支持，而我也相信，此後您還是不停的關照著我，這都成爲我不畏困難繼續努力的原因，再次歡愉且崇敬地謝謝你們。

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

DroidDream malware becomes Android Market nightmare. On 1st March, 2011, Lookout Mobile Security [4] finds that there are more than fifty Android Market applications have been implanted with malware called DroidDream. These popular applications have been stolen, encrypted, garbled and republished by someone. Google has removed these infected applications from the Google Android Market. Google also issues a remote kill to applications which are injected with DroidDream from infected Android smart phones. We are not trying to be sensationalist. Android malware poses serious threats to user privacy. In order to know something about DroidDream, we study reference reports written by Lookout Mobile Security. According to Lookout researches, DroidDream steals mobile identification information, including IMEI (International Mobile Equipment Identity Number) and IMSI (International Mobile Equipment Subscriber Identity Number). In our opinion, malware would be interested in SMS (Short Message Service), GPS (Global Positioning System) and other files in the future. DroidDream is a warning to arouse our attention to mobile security.

Android malware have been able to invade the Android OS for concealment. Malware authors republish appealing applications with malicious DroidDream code. Garbled documentations and manifests (installation list) are always used to induce users to install republished and malicious applications. These malware-infested and republished malicious applications silently download other applications, and silently send information to a remote server. The thorniest problem is that malware attacks Linux kernel directly to get root access in Android OS. In order to do malicious

behaviors, malware root an Android using exploits named “exploid” and “rageagainstthecage”. It helps malware to totally control whole Android OS. These problems come from vulnerabilities derived from the Linux file system YAFFS2 (Yet Another Flash File System 2). It helps malicious applications to steal user information and hide their malicious behaviors. There comes several challenges, including: a) malware could affect the integrity of the Android OS, so that we could not believe messages retrieved from Android OS libraries, b) malware could affect the accuracy of analysis tools developed in the Android OS environment, c) An Android OS manifest (installation list) could not really limit behaviors of specific applications, so that related analysis techniques based on manifest mechanism could be circumvented by malware with exploit ability.

Users could download third-party Android applications from the Internet and Android Markets. It is noteworthy that more and more Android Markets are established by telecommunications industry and private agency. In view of today’s sophisticated malware technology, but Android Markets impose no checking on published applications uploaded by programmers. Users could only know manifest about behaviors of downloaded application. However, DroidTracking supports an automatic checking mechanism to reveal behaviors that steal sensitive information being monitored. Our proposed scheme shows that DroidTracking proposes an fine-grained and system-wide information flow tracking on the Android. Even if malicious applications exploit the Android OS, destroy integrity of Android libraries and download third-party modules, DroidTracking could keep information flow tracking accurate. Because the Android Emulator provides the hardware level simulation, DroidTracking is implemented on the Android Emulator to prevent our scheme from malicious application attacks. In order to achieve fine-grained information flow tracking and fine-grained object analysis, we append an information

flow tracking to the emulated ARM CPU. Therefore, the modified and the emulated ARM CPU can trace each instruction execution, memory access and register access. Furthermore, we record byte-level granularity status of memory and registers after each instruction execution. Our scheme could help program analysts to reveal behaviors of stealing sensitive information. Actually, DroidTracking could benefit end users and Android Markets.

In order to prevent Android users from sensitive data leakage attacks, related work such as information flow tracking [8, 9, 10, 11, 12, 13], manifest-based access control [14, 15, 16] and user awareness [19] have been proposed to solve the problems. Coarse-grained information tracking [13] find behaviors about information leakage caused by malware. However, their scope is limited inside Java application and Android libraries. Because of static analysis used to analyze Android libraries, they could not trace information flow of downloaded unknown malware modules. According to manifest (installation list), access control provides the right application with the right access to information. Although access permissions are granted by user, user is always the weakest link of computer security. User could not pay attention to garbled and complicated documentations, but grants it all. The worst of all, manifest could not prevent permissions from abusing. User awareness reminds user to understand the information is being used. Generally speaking, customized hardware device such as camera light is designed for Android users to know camera is in used. User awareness does not apply to a multi-resource Android phone. These limitations make related work inapplicable in advanced Android malware such as DroidDream, since DroidDream could totally control whole Android OS.

1.1 Contribution

This paper present DroidTracking, a fine-grained and system-wide information flow tracking. We equip the Android emulator with system-wide instruction tracking, byte-granularity memory tracking and byte-granularity register tracking. We do system-wide instruction tracking to prevent DroidTracking from malicious interference. Even if the Android OS is infected by malware, analysis results of DroidTracking are still correct. Our evaluation shows that DroidTracking exactly reveals behaviors of DroidDream. By putting DroidDream under observation, more than two monitored resources are stolen through packet sending by DroidDream. We highlight our contributions in this paper below.

- Behaviors of stealing information are modeled and revealed with byte-granularity object analysis. Each instruction execution of the target application is monitored by DroidTracking.
- Malware cannot touch our modules since it locates below the emulator. DroidTracking is uncompromisable. We make our claim according to state-of-the-art Android malware.
- DroidTracking supports a comprehensive monitoring. The scope of our scheme comprehends not only Java applications but also whole Android machine.

1.2 Synopsis

The rest of this paper is organized as follows. Chapter 2 discusses previous work related to information flow tracking and information protection. Chapter 3 provides a high-level overview of DroidTracking. Chapter 4 describes the system design of DroidTracking and instruction design of ARM architecture. Chapter 5 presents analysis result of our experiments. Chapter 6 summarizes and concludes this paper.

Chapter 2

Related Work

Ubiquitous mobile devices become the part of life. A mobile operating system includes iOS, Android OS, web OS, Windows Mobile, or Symbian OS that controls a mobile device or information appliance. Mobile security becomes the important concern. There are approaches to prevent sensitive information from being stolen by malware. The first one is information flow tracking [8, 9, 10, 11, 12, 13] to track sensitive information flowing in the Android operating system. Sensitive information is labeled as a specific identity label. By tracking the taint label, we could reveal behaviors about information leakage cause by malware. It helps users to know what process accesses the sensitive information and where the sensitive information is flowing. Importantly, it reveals that the sensitive information leaves the system at a taint sink. The second one is manifest-based access control [14, 15, 16]. To get the right access to information, programmer has to claim their permission request in the manifest (installation list). When users go to install an application, there is a manifest to be granted by users. It helps users to understand what the application does. A manifest provides the right application with the right access to information. The third one is user awareness mechanism. It protects users from unnecessary use of microphone, camera, Bluetooth, and other sensors. Customized hardware device such as camera light is used to remind users of camera in use. This technique is widely used on small hardware device.

2.1 Information Flow Tracking (IFT)

Information flow tracking means that it tracks information flowing between

application-level objects (e.g., function parameters, libraries and messages passing between applications) or system-level objects (e.g., registers, memory, and I/O events); furthermore, dynamic information flow tracks running process, whereas static information flow tracks application with static source code analysis.

2.1.1 Fine-grained DIFT

Fine-grained dynamic information flow tracking (Fine-grained DIFT) tracks sensitive information by analyzing system-wide information [8, 9, 10, 11, 12]. By using hardware extensions and emulation environments supports, fine-grained DIFT analyzes whole system (e.g., memory, registers, instruction set, and emulated hardware device) and operating system (e.g., applications, user libraries, and kernel modules). The approach identifies the sensitive information with a taint source. Fine-grained DIFT tracks the taint source at the instruction translation level. By analyzing instruction set, accurate tainted data is propagated and recorded by fine-grained DIFT system. Finally, fine-grained DIFT checks tainted data when information is transmitted to the remote server by the network interface card. These related works are designed for x86 architecture. Of course x86-based design concepts cannot all be used to ARM-based DIFT. According to our survey, there is the related work [13] for the first time proposed some ARM-based DIFT concepts not being implemented yet. The concepts are not enough, because ARM-based smartphone has particular architecture (e.g., ARM instruction set, Thumb instruction set, coprocessor, register banking, addressing mode and so on), sensors (e.g., camera, GPS, microphone and so on) and variety of sensitive information (e.g., IMEI, IMSI, ICC-ID and so on) to be solved. Finally, we propose total solution to address issues on Android operating system.

2.1.2 Coarse-grained DIFT

Coarse-grained dynamic information flow tracking (Coarse-grained DIFT) tracks sensitive information inside the application [13]. Coarse-grained DIFT analyzes whole operating system, including applications, user libraries and kernel modules without hardware extensions and emulation environments supports. The approach identifies the sensitive information with a taint source also. However, coarse-grained DIFT proposes the multi-level (message-level, variable-level, method-level and file-level) approaches to track taint source in a physical smartphone. According to our knowledge, TaintDroid [13] proposes the most complete solution to solve the taint propagation in many issues, including Android-specific inter-process communication (IPC), Dalvik VM interpreter and native methods. The most important of all, TaintDroid minimizes a runtime overhead to make realtime analysis real. Unfortunately, the approach relies on native libraries' integrity, taint interface libraries' integrity and firmware's integrity. In our comment, TaintDroid could not prevent their scheme from malicious attacks caused by DroidDream, because DroidDream can attack integrity of the Android OS.

2.2 Manifest-based Access Control

Manifest-based access control supports an application-level permission mechanism. The permission mechanism provides the right application with the right access to information. In order to access confidential information and device service (e.g., phone call, SMS message, camera and GPS, etc.), Android application lists the manifest (install list) to acquire user permission. During the period of installing an Android application, users could check the manifest to grant it all or not. Nevertheless,

users cannot grant partial permission also, and refuse unneeded permission that can be abused.

Related works [14, 15, 16] devote to enhance manifest-based access control mechanism. Kirin [15] and Saint [14] propose the enhanced permission mechanism to prevent sensitive information from being accessed. These two systems propose a concept of selective Android permissions. The goal enables users to refuse unwanted permissions that can be maliciously used. The worst of all, user is the weakest link of computer security. In addition, shared user-ID is another problem. If an Android application A declares a shared user-ID permission to require the other application B's permission, it is difficult to predict behaviors of the application A actually. For example, application A has an "INTERNET" and "shared B" permission, and application B has a "GPS" permission. Application A has ability to transmit GPS to the Internet therefore.

In our comments, manifest could not prevent permission from being abused, and could not show what the application actually has the behaviors at runtime. In addition, by using a garbled documentation or a malware-downloaded application, DroidDream is able with the root ability to read sensitive information. Manifest-based access control could not have mediation methods to solve problems.

2.3 User awareness

In order to increase the interaction between user and mobile phone, mobile applications have ability to access sensors such as GPS, camera, and microphone. By confirming access permission during the period of installing mobile applications, mobile applications could collect sensitive data via sensors. Our related work [19] focuses on the problem that sensors may be maliciously used without user awareness.

Therefore, there is an idea related work [19] propose to protect users from unwanted use of sensors. By appending hardware device warning to a mobile phone, it reminds users that mobile resources are in used. For example, camera-used LED indicator lights up if camera is in used to capture video data. However, the Android OS has many hardware devices and information (e.g., IMEI, IMSI, SMS, address book, and etc.). Especially, Android malware is interested in information, but related work is not designed for widespread information. Because of DroidDream has the root ability to destroy integrity of the Android OS, user awareness mechanism could not prevent itself from malicious attacks.

Chapter 3

Approach Overview

By appending an information flow to emulated ARM CPU, we could know how Android applications steal sensitive information. In addition, it is important to know what information is stolen by malicious application, and what device is used to transmit the sensitive data.

3.1 Challenges

To understand these problems, there are several challenges we present below.

a) Track the sensitive resources which are read and written to the memory space. Therefore, we could tag the specific memory space as a dirty space (tainted space). After each executed instruction, we track the whole memory and keep a record of memory changes affected by the dirty space.

b) Track the memory space which is read and transmit to remote server by hardware device. We should also check the memory space to know whether the memory space is tagged as a dirty space. If the memory space is tagged as a dirty space, it is the fact that sensitive resources are transmitted to the network.

c) Process identification, register banking and instruction analysis are problems to be discussed. These problems help us to know which process has the malicious behaviors. In addition, it is certainly necessary to track the status of register and memory after each executed instruction.

The challenges are proposed above. The rest of this section demonstrates that proposed approaches solve the problems in detail.

3.2 Two-phase Scheme

In order to analyze whole Android OS, there are two phases we propose below. The first phase, we modify emulated hardware devices on the Android emulator to track triggered events including Taint Source events and Taint Sink events. For example, character device is a virtual device used for communication with the shell command (Android Emulator users could use telnet protocol to connect to the Android OS for shell commands). Sensitive information such as GPS and SMS are sent to the Android OS through emulated character devices. By modifying emulated character devices, we could start to track these monitored resources and handle Taint Source events. By modifying network interface card (NIC), there is a packet sending events occurred to be monitored by DroidTracking. Therefore, the challenge a) and challenge b) are overcome with modified emulated hardware devices. Of course, information flow tracking system helps us to know what information flows from the Taint Source module to the Taint Sink module during the period of running Android OS. The Chapter 4 will demonstrate the information flow tracking in detail. The second phase, we analyze ARM or Thumb instructions to take down behaviors of whole Android OS including Android applications, Android libraries and the Android-based Linux kernel . In order to analyze system-wide behaviors of the Android OS, there are Taint Metadata modules including memory metadata and register metadata to be used to keep track of memory status and register status. To acquire physical memory access space of instructions, we modify the memory management unit (MMU) of the Android Emulator to record memory access. Because of unique visible register issue of ARM architecture, re-banked register with CPU state for instruction analysis is necessary. We do re-bank registers during the instruction translation time. Additionally, a process identification issue is applied to

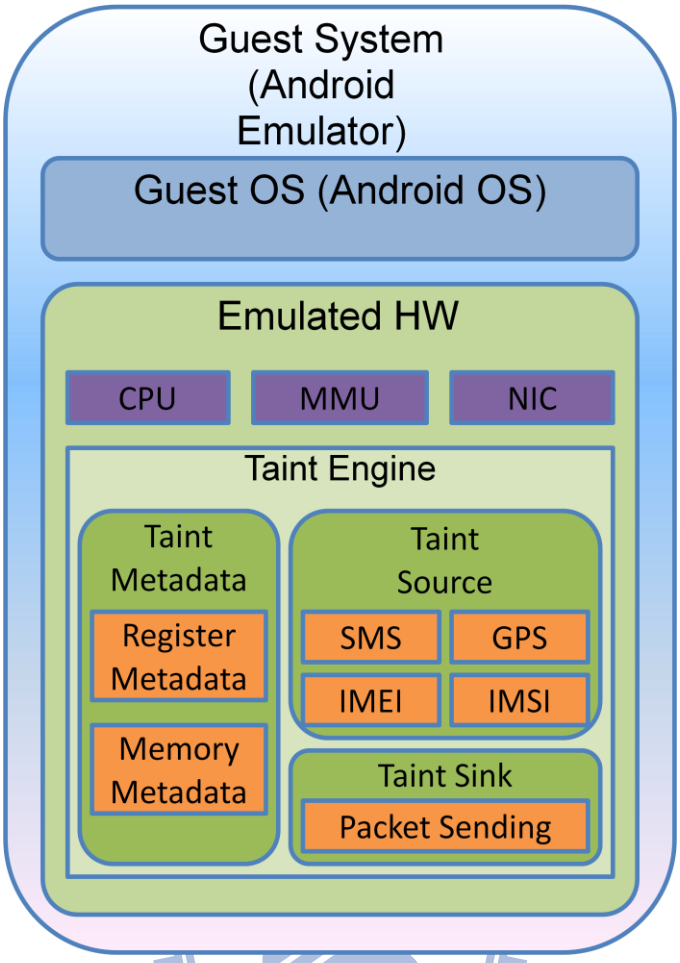


Figure 3.1: System Architecture

identify the process having the malicious behaviors. We trace the context switch to identify the process with the Linux-based Process ID (PID).

3.3 System Architecture

In this paragraph, we show our proposed system architecture and system components with detail demonstration below.

Figure 3.1 presents our system implemented on the Android Emulator. The Android Emulator emulates lots of physical Android hardware devices. By modifying these emulated hardware devices, we finally carry out DroidTracking to track information flows of sensitive data on the Android OS. There are CPU, MMU, NIC module are enhanced on the Android Emulator. By modifying CPU module,

DroidTracking analyzes instruction execution so that memory access and register access could be tracked. By modifying memory management unit (MMU) module, DroidTracking acquires physical memory address access of the analyzed instruction. By modifying network interface card (NIC) module, DroidTracking keeps the track of each packet sending event. NIC module helps us to keep the track of the memory spaces sent by NIC. The most of important, NIC module checks the memory spaces to reveal behaviors of stealing sensitive data.

There are three modules Taint Metadata, Taint Source and Taint Sink in our core Taint Engine. In order to achieve fine-grained and system-wide information flow tracking, Taint Metadata module records the byte-granularity objects in system. Under the state-of-the-art related work, byte-granularity object analysis is the most effective and fine-grained as we know. Taint Source module is set of monitored resource. There are ICC-ID (Integrated Circuit Card Identifier), GPS (Global Positioning System), IMEI (International Mobile Equipment Identity Number) and IMSI (International Mobile Subscriber Identity) in the Taint Source module at present. In the future, we will add more and more monitored resources to the set of Taint Source module. Taint Sink module is used to record specific triggered events. At present, we only track the packet sending events to reveal malicious behaviors of stealing sensitive data.

At the end of this section, we demonstrate the system control flow of our proposed system to readers. First of all, DroidTracking uses Taint Source module to know where sensitive data is written to the memory space. DroidTracking marks this memory space as a dirty memory space then. At the following steps, DroidTracking tracks the information flow during the period of program execution, and propagates the taint tags in Taint Metadata module. Therefore, Taint Metadata components is frequently updated by Taint Engine because of lots of executed instructions. The last, Taint Sink module is the most important protector to reveal behaviors of stealing

sensitive data by checking status of memory space which emulated hardware device reads. Of course DroidTracking also indicates the malicious process stealing sensitive data.

Chapter 4

DroidTracking

There are two-phase scheme to monitor whole Android OS. The first one is triggered events analysis. The second one is system-wide and fine-grained information flow tracking. Figure 4.1 shows our proposed flow chart.

4.1 System Flow Chart

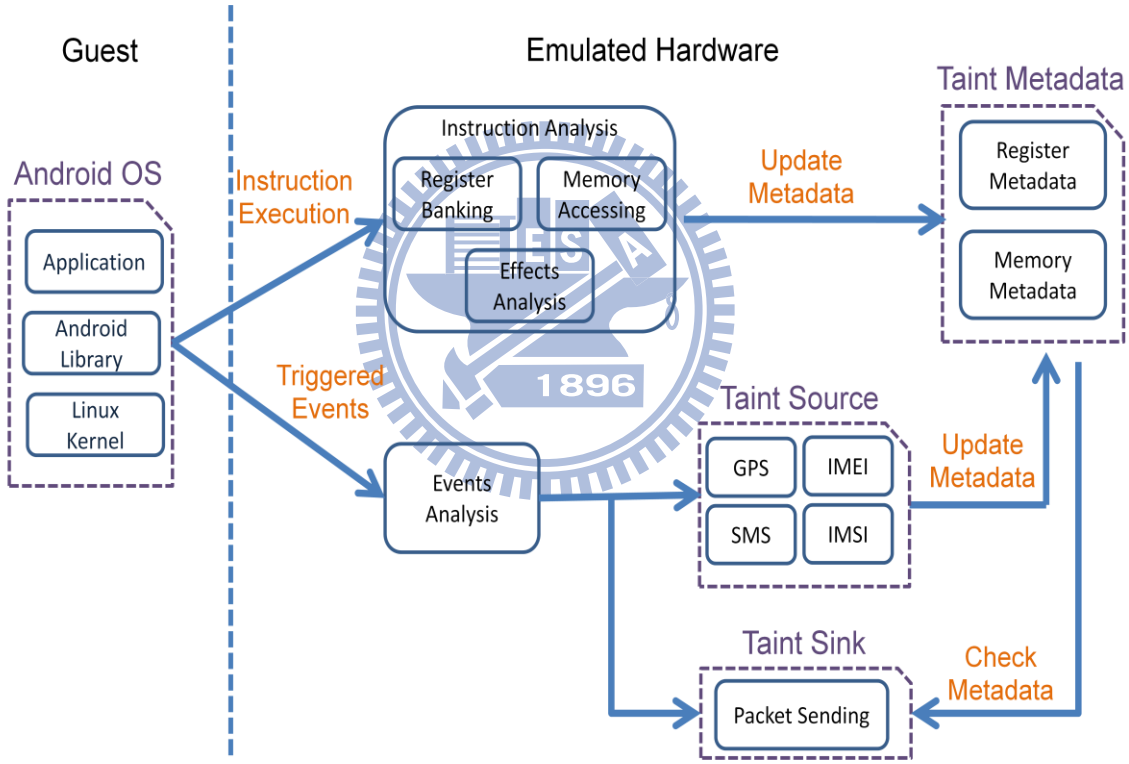


Figure 4.1: System Flow Chart

The left side of Figure 4.1 is monitored guest environment. Our scheme do not limit the scope inside Java application; moreover, it monitors whole Android OS including Java applications, Android libraries, Android-based Linux kernel and downloaded unknown malware modules. By comprehensive monitoring, it prevents our scheme from unknown malicious attacks. The right side of Figure 4.2 is

DroidTracking system based on Android Emulator. In order to achieve fine-grained object analysis, we track system-wide objects including memory and register instead of Java parameter, library parameter, and cross-application message. we propose a Taint Metadata in Figure 4.2 to keep track of byte-granularity memory status and byte-granularity register status. Memory Metadata could record whole Android memory space emulated by the Android Emulator. Register Metadata could record all general purpose registers of the ARM architecture. On the other hand, the rest components of DroidTracking maintain the same Taint Metadata keeping track of system-wide information. Taint Source component is used to identify what the sensitive information is read to memory. By updating Taint Metadata, Taint Source component keeps track of memory address contaminated by sensitive data. To detect behaviors of stealing sensitive data, Taint Sink module distinguishes status of memory address by checking Taint Metadata. In addition, Taint Metadata is frequently updated by Instruction Analysis module also because of changes after each executed instruction.

4.2 Instruction Analysis

We use Register Banking, Memory Accessing and Effects Analysis components to analyze each ARM or Thumb instruction. By the way, we do not analyze Thumb-II instructions because of the Android Emulator version not supported. It helps us to realize how instructions make an impact on the Android OS. For example, an instruction could move a register/memory/immediate object to a register/memory object. Even if CPU processes a data processing instruction such as ADD, MUL and MOV, results of the instruction execution are still a data movement from source objects to destination objects. Therefore, DroidTracking analyzes the relationship

between source objects and destination objects to model behaviors of an instruction. By updating a common Taint Metadata after each executed instruction, it helps us to record the execution process of the executed Android OS including Android application, Android libraries, Android-based Linux kernel and downloaded unknown malicious module. Finally, we make a simple conclusion. Taint Source module reveals what the monitored resource is accessed to the Android System. Instruction Analysis module tracks the executed process of whole Android OS system. Taint Sink module reveals what the monitored resource is stolen and transmitted to the Internet.

4.3 Information Flow Expression

We begin this section with a rigorous expression of information flow and granularity of objects. If a destination object computed by CPU is affected by a source object, there is an information flow from the source object to the destination object. By the information flow tracking system we proposed, it is possible to comprehend behaviors of the whole Android OS through instruction analysis.

Before our introduction to instruction analysis, we demonstrate object granularity design to achieve our goal of fine-grained analysis. In order to achieve byte-granularity object analysis, a study of the ARM instruction set architecture is carried out. Because an instruction operand could be a register, memory or immediate value, these operands should be considered as analyzed objects. Nevertheless, a destination operand of an ARM instruction cannot be an immediate value. Therefore, an immediate value cannot be affected by source objects. There is a Taint Metadata to track the status of each byte-granularity memory and the status of each byte-granularity register only. Figure 4.2 shows that the third and the forth bytes of Register 1 (R1) are contaminated by IMSI information, but the first and the second

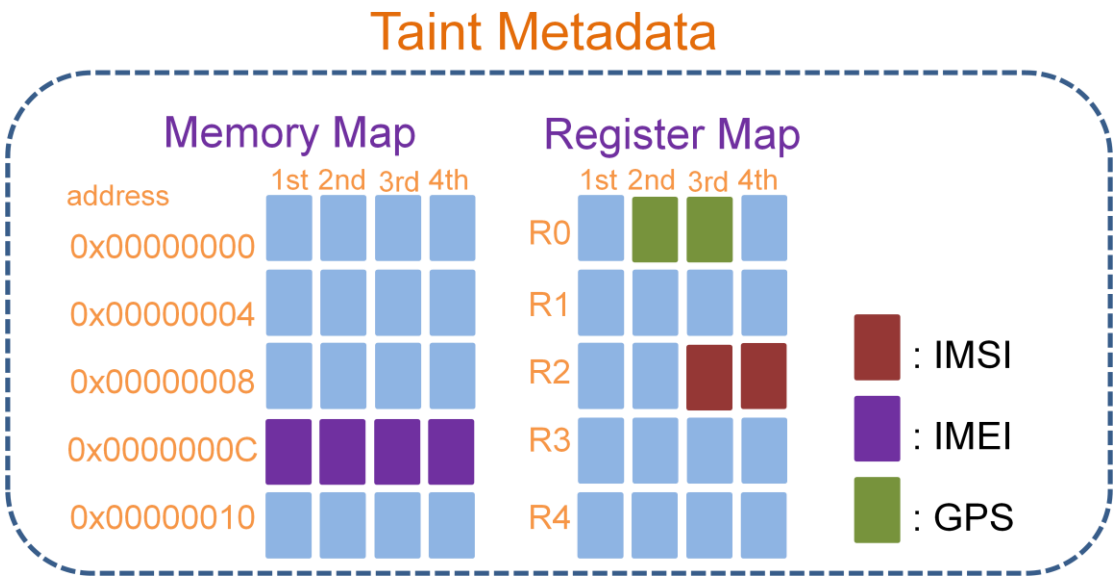


Figure 4.2: Taint Metadata

bytes of R1 are not contaminated.

By analyzing the ARM instruction set architecture, there are five standard effects we defined could describe that source objects make an impact on destination objects (operands). The first one is one-to-one effect that each byte of source object affects one corresponding byte of destination object. The second one is a mixed effect that each byte of source object affects all bytes of the destination object. The third one is an assigning effect that the destination object is contaminated if the source object is contaminated. The forth one is an appending effect that the destination object is contaminated if the source object or the destination object is contaminated. The fifth one is a clear effect that the destination object is not considered to be contaminated with sensitive information.

Table 4.1: Information Flow Expression

Category	Notation	Algorithm	Instruction	Information Flow Expression (DST exp SRC)
One-to-One Assigning	<-	For i = 0 to 3 DST[i] = SRC[i]	MOV R0, R1	R0[3,2,1,0] <- R1[3,2,1,0]
One-to-One Appending	<+	For i = 0 to 3 DST[i] |= SRC[i]	AND R0, R1, R2	R0[3,2,1,0] <- R1[3,2,1,0] R0[3,2,1,0] <+ R2[3,2,1,0]
Mixed Appending	<<+	For i = 0 to 3 If SRC[i] == dirty for j = 0 to 3 DST[j] = dirty	MUL R0, R1, R2	R0[*] <<+ R1[3,2,1,0] R0[*] <<+ R2[3,2,1,0]
Clear	<<0	For i = 0 to 3 DST[i] = clear	MOV R0, #9855	R0[*] <<0

Table 4.1 shows expression of information flows. There are few examples shown in this paragraph. The “MOV” instruction has an one-to-one assigning effect on its destination operand. If each byte of the source object is contaminated, the corresponding byte of the destination object is contaminated. The “ADD” instruction moves the sum of two source operands to the destination operand. The first source operand has an one-to-one assigning effect on its destination operand. The second source operand has an one-to-one appending effect on its destination operand, because it cannot change the status of bytes contaminated by the first source operand. The “MUL” instruction moves the product of two source operands to the destination operand. Because a computed result of the “MUL” instruction has a byte-mixed computation, we evaluate the result with a rough estimate. As a result, the product of two source operands has a mixed appending effect on the destination operand.

4.4 Information Flow Analysis

The instruction analysis system is quite a help to know any change of the Android system. By checking and updating the Taint Metadata, the instruction

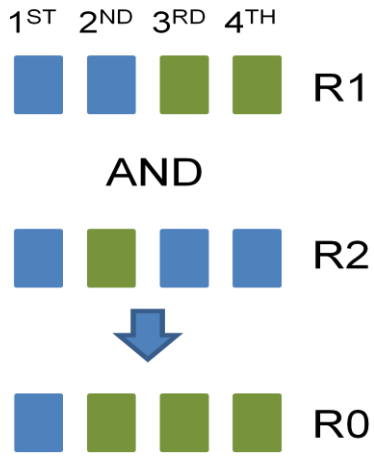


Figure 4.3: AND R0, R1, R2

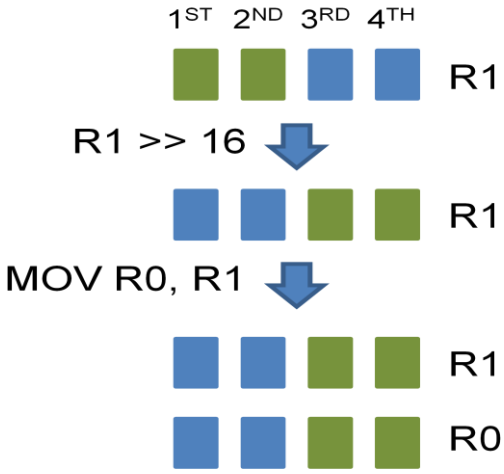


Figure 4.4: MOV R0, R1, LSR#16

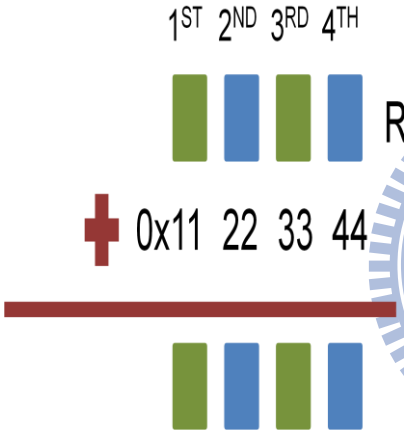


Figure 4.5: ADD R0, R1, 0x11223344

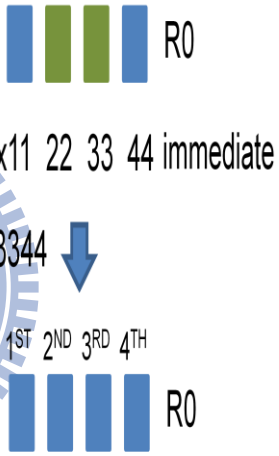


Figure 4.6: MOV R0, 0x11223344

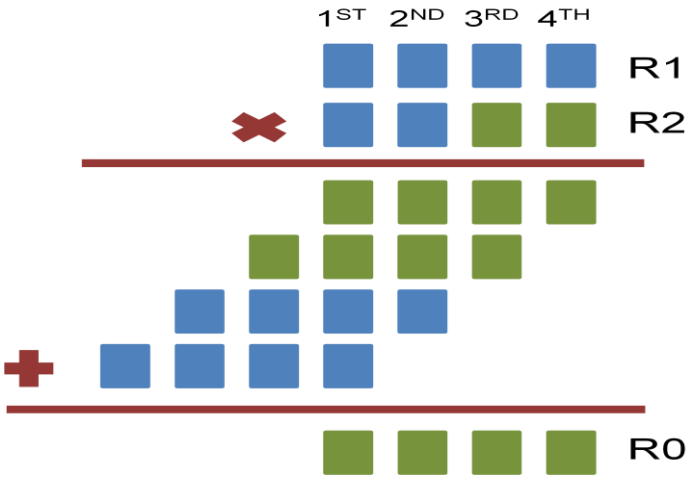


Figure 4.7: MUL R0, R1, R2

analysis system records the changes of a whole execution process. A process of updating metadata is also known as taint propagation. In other words, there is tainted information propagated (or updated) to destination objects. In order to do taint propagation, we are necessary to study the ARM instruction set architecture. We now discuss topics to understand specific instruction set architecture.

4.4.1 Data-processing Instruction with Register Operand

Figure 4.3 shows an example. The third and the fourth bytes of R0 are contaminated by R1, because R1 has an one-to-one assigning effect on R0. The rest clear bytes of the R0 may be contaminated by R2. Therefore, we use a one-to-one appending effect to keep track of an impact on R0.

4.4.2 Data-processing Instruction with Shifter Operand

Figure 4.4 shows an example. That all bytes of R1 are logical shifted right by 16 bits (2 bytes). Thus the third and the fourth bytes of R1 are contaminated by the first and the second bytes of R1. The first and the second bytes of R1 are clear, because zeros are inserted into the vacated bit position. The move instruction moves R1 to R0 finally. R1 has an one-to-one assigning effect on R0. In this case shown in Figure 4.4, there is no false positive and false negative. However, there is byte-granularity object analysis used to analyze instruction. If the shift amount is not a multiple of 8, there is few bits lead to false positive and false negative. In order to balance efficiency and precision, byte-granularity object analysis is good enough according to our evaluation and background knowledge.

4.4.3 Data-processing Instruction with Immediate Operand

Figure 4.5 shows an example. The ADD instruction adds the value of register operand R1 to the immediate value 0x11223344, and stores the result in the destination register R0. The source register R0 has an one-to-one assigning effect on the destination register R1. Therefore, the first and the third bytes of the destination register R1 are contaminated by the source register R0. The second source operand is an immediate value treated as a clear object, and could not affect the status of the destination register R1.

There is another example shown in Figure 4.6. The MOV instruction moves the clear immediate value 0x11223344 to the destination register R0. Although the original register R0 is contaminated by sensitive information (tainted data), the result of computation is that the register R0 is clear with no sensitive information.

4.4.4 Multiply

Figure 4.7 shows an example. The MUL instruction multiplies the value of register operand R1 to the value of register operand R2, and stores the result in the destination register R0. The source register operand R1 and R2 have a mixed appending effect on the destination register operand R0. In other words, if any byte of source operands is contaminated by sensitive information, the all bytes of the destination operand are contaminated by sensitive information. Register-granularity object analysis is used to analyze the MUL instruction. However, there are false positive and false negative in this case. In order to improve this problem, we could take care of displacement from the product. It incurs much overhead for instruction

analysis system. To balance efficiency and precision, byte-granularity object analysis is good enough according to our evaluation and background knowledge.

Chapter 5

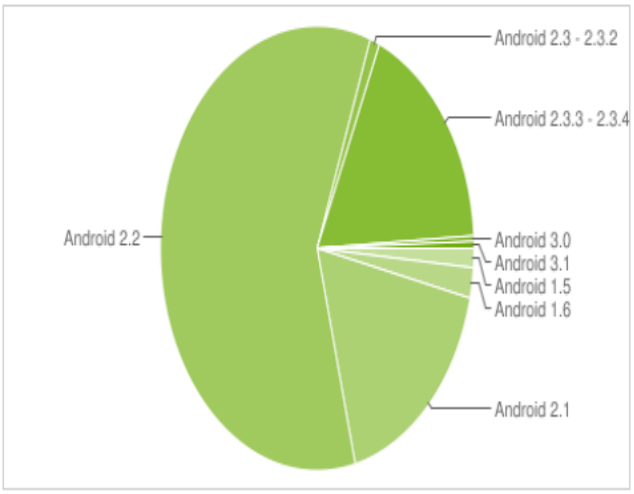
Evaluation

To demonstrate DroidTracking, a fine-grained, system-wide information flow tracking, several experiments are conducted. To demonstrate effectiveness of approaches proposed in this paper, self-programmed C applications, self-programmed JAVA applications and applications injected with DroidDream are evaluated. By tracking information flow, it helps us to know system-wide behaviors of the target application. DroidTracking records system logs including complete and detailed information flow of whole Android OS: taint source events, taint sink events, information flow occurred by instructions, changed objects status, process identification of each instruction block and the program counter of each instruction block. However, system-wide behaviors and system-wide information are beyond analyst comprehension. Instead of sophisticated information, the process of information flow is modeled by DroidTracking during the period of application execution. We show triggered taint source events, triggered taint sink events, unsophisticated objects status (e.g. Table 5.2 shows a memory working set to readers) and unsophisticated information flow.

5.1 Experiment Environment

Before the experiments, we show our experiment environment in detail. We implement our DroidTracking on the Android emulator 8.0. The Android emulator emulates ARMv5TEJ processor supporting ARM instruction set, Thumb instruction set, system reserved coprocessor (CP15) and Jazelle DBX (Direct Bytecode eXecution). It was known as the advanced RISC (Reduced Instruction Set Computer)

Table 5.1: Current Android Distribution



Platform	API Level	Distribution
Android 1.5	3	1.4%
Android 1.6	4	2.2%
Android 2.1	7	17.5%
Android 2.2	8	59.4%
Android 2.3 - Android 2.3.2	9	1%
Android 2.3.3 - Android 2.3.4	10	17.6%
Android 3.0	11	0.4%
Android 3.1	12	0.5%

Data collected during a 14-day period ending on July 5, 2011

machine. There is the Android 2.2 based on 2.6.29 Linux kernel running on the Android emulator. Why we choose this version is according to the current Android distribution released by Google during a 14-day period ending on July 5, 2011. Table 5.1 shows that the Android 2.2 is the most popular for Android users.

5.2 Self-programmed C and JAVA

We program C applications sharing a tainted data array between processes and transmitting this tainted data to the Internet through the HTTP POST request and the HTTP GET request. The HTTP POST request is the way that malware usually transmits sensitive information to a specific remote server. We also program JAVA applications accessing sensitive information (IMEI, IMSI, ICC-ID and GPS) and transmitting sensitive information to the Internet through the HTTP request. These two experiments help us to test our DroidTracking system. The taint source subsystem records sensitive information reading events when the applications access sensitive

Applications Infected by DroidDream	Publisher
1. Falling Down	Myournet
2. Super Guitar Solo	Myournet
3. Chess	Myournet
4. Falling Ball Dodge	Myournet
5. Spider Man	Myournet
6. Bowling Time	Kingmall2010
7. Advance Barcode Scanner	Kingmall2010
8. Music Box	Kingmall2010
9. Sexy Legs	Kingmall2010
10. Piano	we20090202
11. Bubble Shoot	we20090202
12. Funny Face	we20090202
13. Tie a Tie	we20090202
14. Quick Notes	we20090202
15. Basketball Shot Now	we20090202

Table 5.2: Applications infected by DroidDream

information. The memory space to which sensitive information is written could be marked as a taint tag by the taint source subsystem. During execution time of the applications, DroidTracking analyzes instructions to record system-wide information flow for the tested applications. Taint tags in taint metadata (including memory metadata and register metadata) is also propagated by DroidTracking. Therefore, we could tracks system-wide behaviors of the tested applications. When tainted memory space is sent out by the HTTP POST request or the HTTP GET request, the taint sink subsystem records sensitive information stealing events. According to our process identification logging, we could identify the process that steals sensitive information. The tested self-programmed C and JAVA help us to understand the correctness of information propagation in sophisticated instruction analysis. Our DroidTracking system could detect sensitive information stealing correctly.

5.3 DroidDream

A lot of applications are injected with DroidDream shown in Table 5.2. These known applications all are deleted by Google Android Market, because Lookout Mobile Security found the malicious semantic on March 1, 2011. DroidTracking analyzes the majority of these applications to reveal malicious behaviors. We could detect the malicious behaviors of DroidDream. We show our evaluation of DroidDream below.

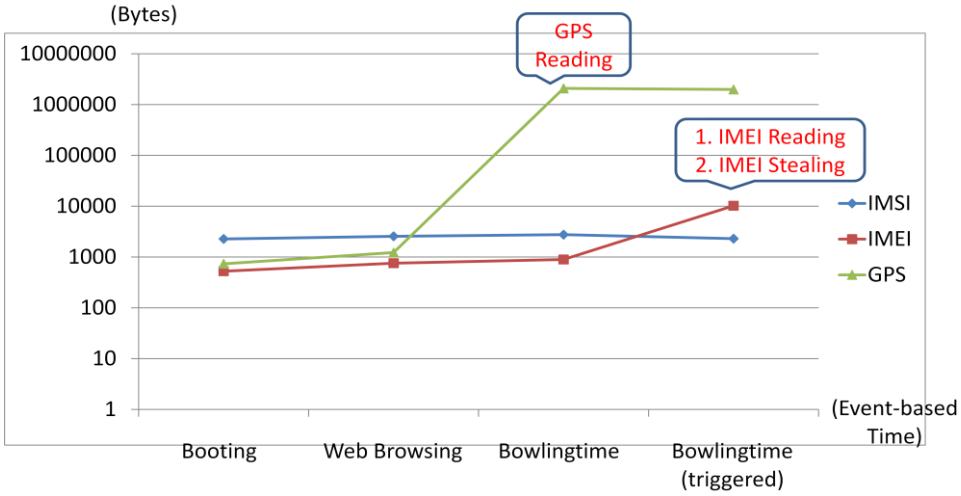
In order to demonstrate our evaluation precisely, we take a Bowling Time as an example. There is the memory working set shown in Table 5.4 for Bowling Time evaluation. The first, we start our emulator-based DroidTracking. There is sensitive information such as IMEI, IMSI and GPS are read to memory by Android OS during the booting time. The second, we use our Android emulator normally. Browser is opened to browse lots of web pages. Because a browser does not access sensitive information, a percentage of sensitive information has no huge changes in memory. The following steps, we run Bowling Time and check the memory working set. In our finding, a percentage of GPS is about two thousand times the bytes of the past in memory. In fact, Bowling Time logs GPS frequently. However, there is no GPS sent out by Bowling Time. Next, there is an events in Table 5.4 occurred. Almost in the same time, there is an advertisement downloaded and taint sink events occurred. A percentage of IMEI is about twelve times the bytes of the past in memory. In fact, Bowling Time accesses IMEI frequently. Table 5.3 shows partial processes. DroidTracking logs process identifications, program counters, instructions, destination operands, source operands, and status changes for instruction analysis. It is the evidence that IMEI is accesses by Bowling Time. Behaviors of Bowling Time change the memory status and the register status in taint metadata. After all,

DroidTracking finds that lots of bytes with IMEI taint tag are sent to a remote server. Therefore, DroidTracking finds an IMEI stealing in this evaluation.

Table 5.3: Instruction Analysis – Bowlingtime

```
(PID:PC      :Insn.   ) Dst. Op. = Src. Op. (changed status)
(69:80216500:e7951103) R1[2] = 3070e08[2] (0->134217728)
(69:80216500:e7951103) R1[3] = 3070e08[3] (0->134217728)
(69:80216500:e7950102) R0[2] = 3070e08[2] (0->134217728)
(69:80216500:e7950102) R0[3] = 3070e08[3] (0->134217728)
(339:afd34524:e01c2ba0) R_TMP[0] = R0[2] (0->134217728)
(339:afd34524:e01c2ba0) R_TMP[1] = R0[3] (0->134217728)
(339:afd34524:e01c2ba0) R2[0] |= R_TMP[0] (0->134217728)
(339:afd34524:e01c2ba0) R2[1] |= R_TMP[1] (0->134217728)
(339:afd34524:e01c2ba0) R_TMP[0] = 0, (134217728->0)
(339:afd34524:e01c2ba0) R_TMP[1] = 0, (134217728->0)
(339:afd34524:101c3ba1) R_TMP[0] = R1[2] (0->134217728)
(339:afd34524:101c3ba1) R_TMP[1] = R1[3] (0->134217728)
(339:afd34524:101c3ba1) R3[0] |= R_TMP[0] (0->134217728)
(339:afd34524:101c3ba1) R3[1] |= R_TMP[1] (0->134217728)
(339:afd34524:101c3ba1) R_TMP[0] = 0, (134217728->0)
(339:afd34524:101c3ba1) R_TMP[1] = 0, (134217728->0)
(339:afd3453c:e0822003) R_TMP[0] = R3[0] (0->134217728)
(339:afd3453c:e0822003) R_TMP[1] = R3[1] (0->134217728)
(339:afd3453c:e0822003) R_TMP[0] = 0, (134217728->0)
(339:afd3453c:e0822003) R_TMP[1] = 0, (134217728->0)
```

Table 5.4: Memory Working Set – Bowlingtime



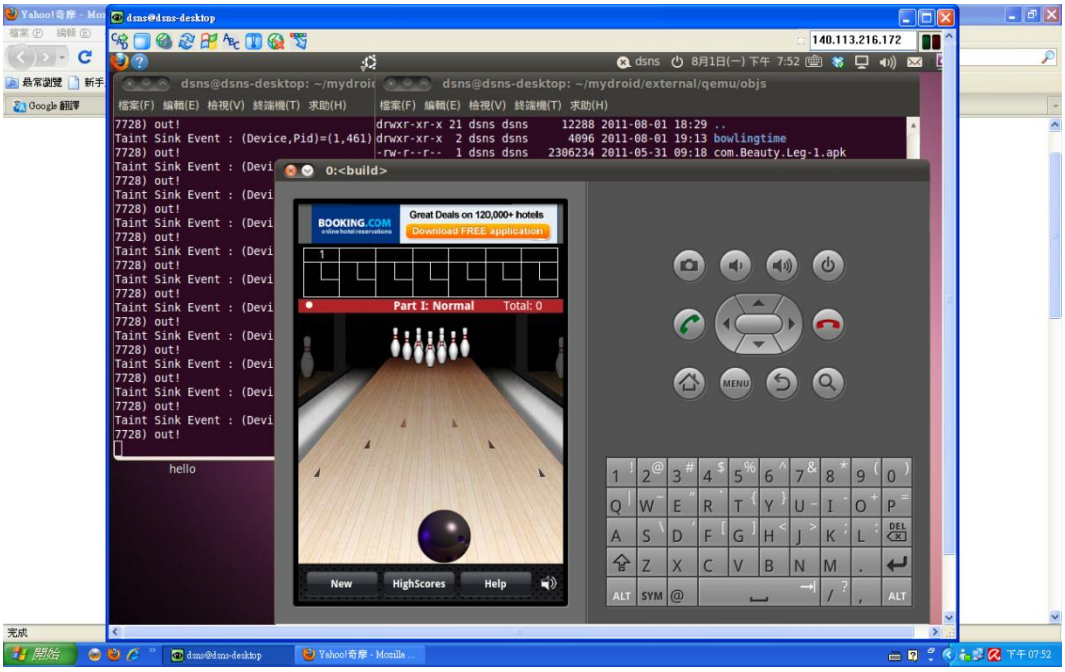


Figure 5.1: Bowling Time

To know DroidDream more, we also evaluate the majority of applications listed in Table 5.2. In our findings, Sensitive information such as IMEI, IMSI and GPS is accessed by these applications. So far as DroidTracking analyzing, behaviors of stealing sensitive information are revealed. According to process identification logging, malicious application is confirmed and identified by the system.

In our findings, DroidTracking analyzes the DroidDream with root capability. Detailed system-wide information is listed for analyst by the system. DroidTracking keeps analyzing correctly, Even if malware downloads new applications and infects the Android OS. Because of emulated-based analysis system, it prevents our system from being circumvented. Our findings demonstrate the effectiveness with emulated-based analysis tools such as DroidTracking.

Chapter 6

Conclusion

DroidDream and others malware could get the root access in Android OS even if Android OS beforehand denies the application permission to have root access. OS-based analysis system fails to protect or detect behaviors of stealing sensitive information. Our primary goals are to prevent DroidTracking from being attacks and to support accurate analysis results. To achieve this, we present DroidTracking, a fine-grained, system-wide information flow tracking that can reveal behaviors of stealing sensitive information (GPS, IMEI, IMSI, ICC-ID and other sensitive information presented in future). A key design concept of DroidTracking is that we equip Android emulator with information flow tracking capability and byte-granularity system object tracking capability.

We run a lot of Android applications injected with DroidDream in Android emulator quipped with DroidTracking. It is the fact that DroidDream accesses sensitive information while an advertisement is downloaded by DroidDream. There is a sequential of information flow in whole memory space and registers. In the course of program execution, several packets with sensitive information are transmitted to the Internet. DroidDream keeps track of memory space sent by malware, sensitive information, the process of information flow and malware's process ID for malware analysis. Our findings demonstrate the Android analysis platform monitoring the whole Android OS and prevent the system from being attacks.

Reference

[1] Android Project. http://source.android.com

[2] Android Developer. http://developer.android.com/index.html

[3] ARM. http://www.arm.com

[4] Lookout Mobile Security. http://www.mylookout.com

[5] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev. Google Android: A State-of-The-Art Review of Security Mechanisms. CoRR, abs/0912.5101. 2009.

[6] W. Enck, M. Ongtang, and P. McDaniel. Understanding Android security. IEEE Security & Privacy Magazine, 7(1):10–17, 2009.

[7] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: A Comprehensive Security Assessment. IEEE Security & Privacy, vol. 8, no. 2, 2010, pp. 35–44.

[8] C.W. Wang and S.P. Shieh. SWIFT: Decoupling System-Wide Information Flow Tracking for Malware Analysis. 2011.

[9] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In Proceedings of ACM Computer and Communications Security. 2007.

[10] F. Qin, C. Wang, Z. Li, H. Kim, Y, Zhou, and Y. Wu. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006.

[11] D. Chandra and M. Franz. Fine-Grained Information Flow Analysis and Enforcement in a Java Virtual Machine. In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC) , 2007.

[12] R. Whelan and D. Kaeli. Toward Whole-System Dynamic Analysis for ARM-Based Mobile Device. In Proceedings of the 13th International Conference

on Recent Advances in Intrusion Detection, 2011.

[13] W Enck, P Gilbert, B Chun, LP Cox, J Jung, P McDaniel, and AN Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010.

[14] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in Android. Proceedings of the 25th Annual Computer Security Applications Conference, ACSAC '09.

[15] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. CCS '09 : Proceedings of the 16th ACM Conference on Computer and Communications Security.

[16] A. Fuchs, A. Chaudhuri, and J. Foster. SCanDroid: Automated Security Certification of Android Applications. Proceedings of the 31st IEEE Symposium on Security and Privacy, 2010.

[17] W. Enck, M. Ongtang, and P. McDaniel. Mitigating Android Software Misuse Before It Happens. Technical Report NAS-TR-0094-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, November 2008.

[18] A. Chaudhuri. Language-based Security on Android. In PLAS'09: Programming Languages and Analysis for Security, pages 1-7. ACM, 2009.

[19] J. Howell and S. Schechter. What You See is What they Get: Protecting users from unwanted use of microphones, camera, and other sensors. Proceedings of Web 2.0 Security and Privacy Workshop, 2010.