Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Đà Nẵng, ngày 17-18/08/2017
DOI: 10.15625/vap.2017.00011

DETECT SECURITY THREAT IN ANDROID CUSTOM FIRMWARE BY ANALYZING APPLICATIONS FRAMEWORK AND DEFAULT SETTINGS

Nguyen Tan Cam[1], Pham Van Hau[2], Nguyen Anh Tuan[2]

[1] Hoa Sen University

[2] University of Information Technology, Vietnam National University Ho Chi Minh City

cam.nguyentan@hoasen.edu.vn, haupv@uit.edu.vn, tuanna@uit.edu.vn

ABSTRACT: *Android operating system has become the world's most popular operating system in recent years. Security threat assessment on the Android operating system becomes imperative. Previous studies focused on analyzing applications on Android devices. However, security threats may exist in other components of the operating system, rather than in applications. In this study, we propose a system, uitXFROID, that allows to detect sensitive data leakage in the Applications Framework and security threat from default settings of Android custom firmware (Custom ROMs) instead of pre-installed applications. The experimental results show that the system accurately detects cases of sensitive data leakage in the samples that we propose ourselves. Besides, the system can be used to analyze Android custom ROMs downloaded from the Internet.*

Keywords*: Custom ROM analysis, Android Framework analysis, sensitive data leakage, ROM default settings.*

I. INTRODUCTION

According to Symantec's statistics [1], they detected 18.4 million mobile malwares in 2016, 5% higher compared with it in 2015. There are 36.000 new variations of Android malware which belong to 4 new malwares detected in 2016. According to IDC statistics [2], Android operating system accounts for 85% of mobile operating system market. As explained in Figure 1, Android operating system has expanded its market recently and continuously keep on that trend. Another results of Symantec [3] revealed that stealing sensitive data behaviour the most popular one with 36%.

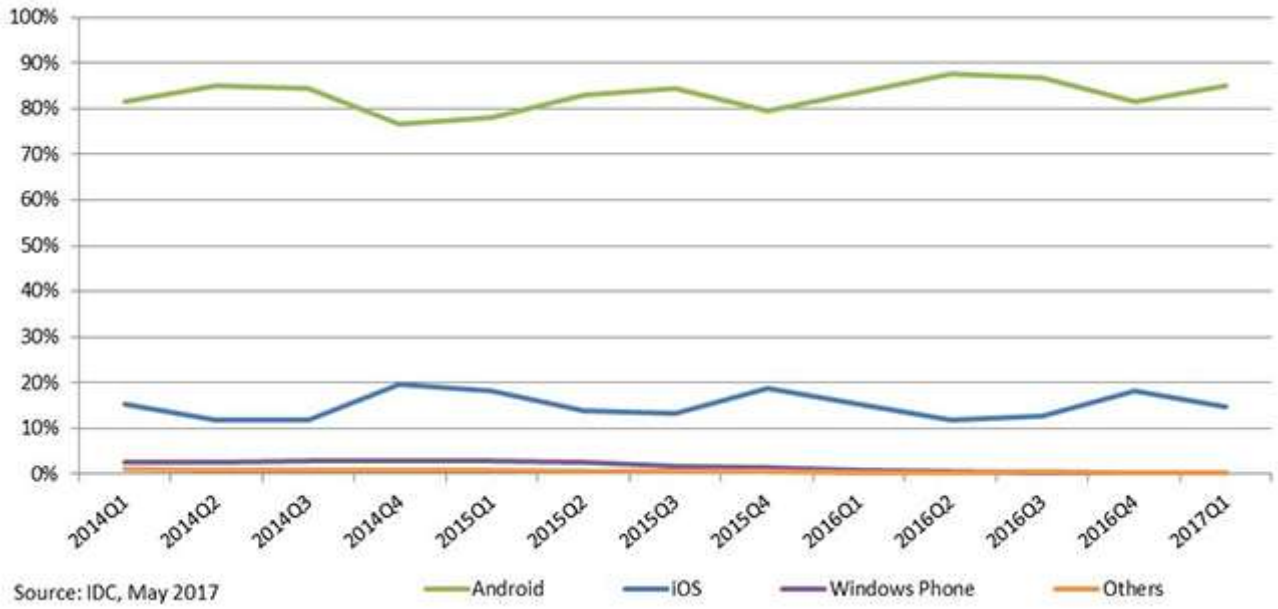

Figure 1. Worldwide smart phone operating system market share (Share in unit shipments) [2]

Android Operating system has five main components, such as Applications, Applications Framework, Libraries, Android Runtime, and Linux Kernel [4]. Figure 2 shows the architecture of Android operating system in detailed.

The Application layer provides main applications offer core features of Android Operation system such as email, SMS messengers, web browser, contact list management,… Application of third-parties can be pre-installed applications in Android Operating system. This is the reason why we should analyze security threat of pre-installed applications in Application layer.

The Applications Framework layer provides a full set of Android Operation system's methods via APIs written by Java. It offers simple ways to reuse system components and services such as View System, Resource Manager, Notification Manager, Activity Manager, and Content Providers. Assuming that all pre-installed applications are benign, security threats still exist in other components of Android Operating system. Applications Framework

component is a special one. It is the reason why security analysis at the Applications Framework layer is necessary during analyzing security on the Android operating system.

The Libraries layer has libraries written by C and C++ that can support to components and services in Applications Framework layer.

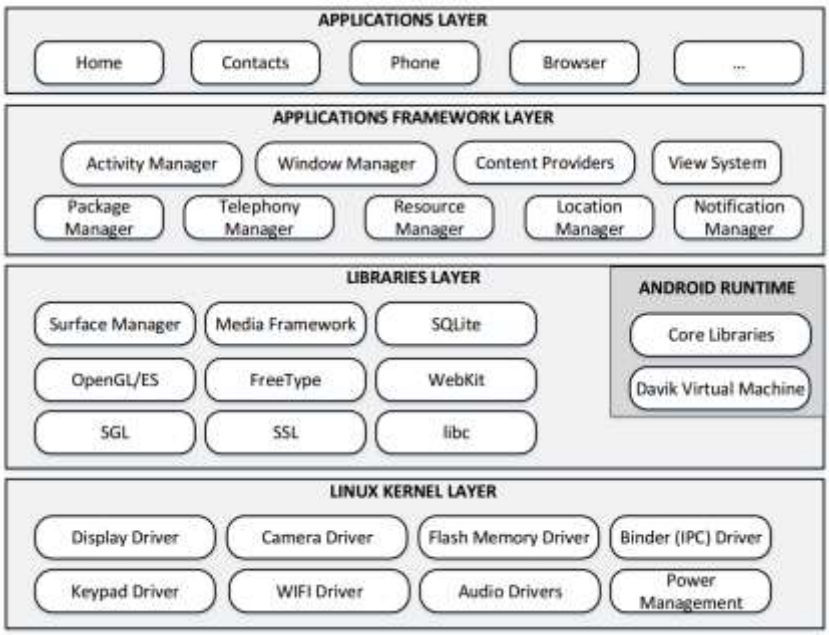

Figure 2. Android operating system architecture [4]

The Android Runtime is used to run multiple virtual machines on low memory devices by executing DEX files. Each application runs in a separate Android Runtime instance. This technique reduces the effects of errors caused by another application.

Linux Kernel is the core component of the Android operating system. Using the Linux kernel allows Android to inherit important security functions and helps device manufacturers develop their hardware drivers. Security analysis on the Android operating system can be deployed at this lowest level. Security analysis at the Linux kernel layer can detect security threats that can not be detected in the upper layers. However, security analysis at this layer will be more challenges because the gathered information is large and difficult to understand.

Currently, there are many studies perform security analysis on Android firmware [5, 6]. However, these studies only focus on single application analysis at Applications layer, and default setting analysis in Android firmware. They didn't perform security analysis at Application Framework layer.

Michael Grace et al propose Woodpecker [6] to detect sensitive data leakage in Android custom ROMs, by analyzing each application in these ROMs. This study uses the Control Flow Graph (CFG) to detect any sensitive data flow. However, Woodpecker only analyzes data flows by using single application analysis instead of multiple application analysis.

DroidRay [5] analyzes pre-installed applications and default settings of 250 Android custom ROMs in the wild. To assess the security threats in pre-installed applications, DroidDray uses VirusTotal [7] to scan each application in the ROM. DroidDray uses single application analysis. Although, DroidRay analyzes more components of the Android firmware than Woodpecker [6], such as analyzing the */etc/hosts* file to detect abnormal URL redirects. However, it does not conduct sensitive data flow analysis on inter-applications and the Application framework.

In this study, we propose a system, named uitXFROID, that enables to detect sensitive data leakage in Android custom ROMs by analyzing sensitive data flows in Android Framework layer and default setting of Linux kernel of Android Operating system.

The main contributions of this study are:

- We propose a method to analyze the security threats in Android custom ROMs by analyzing sensitive data flow in the Android Framework layer and default setting in Linux Kernel layer of Android Operating system. This is the first study conducted sensitive data flow analysis at the Android Framework layer and the default settings of Android custom ROMs.
- We build a dataset that has 10 Android custom ROMs that contain sensitive data leakage scenarios. It can be used as a reliable dataset of related studies in the future.

• We evaluate the proposed system not only on our dataset but also Android custom ROMs downloaded from the Internet.

The rest of this paper is organized as follows: The proposed system is described in Section II. The related works are presented in Section III. Implementation and evaluation are mentioned in Section IV. Section V concludes the paper.

II. THE PROPOSED SYSTEM

In this study, we propose a system that allows analysis the security threats of Android custom firmware by analyzing the Android Framework and the default settings in the kernel. Figure 3 shows the main components of the proposed system.

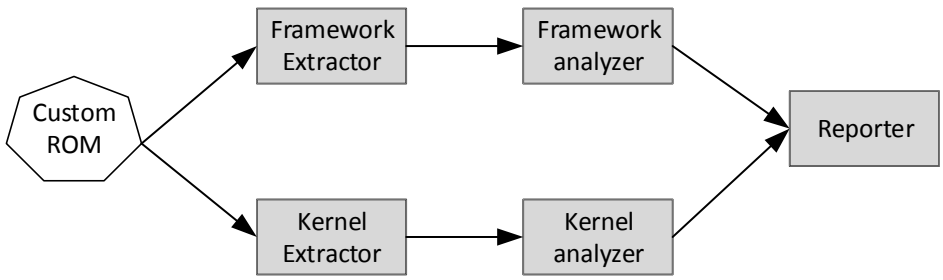

Figure 3. The proposed system, uitXFROID

There are five main components of the proposed system, uitXFROID, including Framework Extractor, Framework Analyzer, Kennel Extractor, Kernel Analyzer, and Reporter.

A. Framework extractor

The framework extractor is used to extract the Android Framework from the Android custom firmware. Because the firmware of different manufacturers use different file system formats. Therefore, the Framework Extractor uses a variety of techniques to extract the corresponding Android frameworks and Linux kernels.

• **In terms of Android custom ROMs has *system* partition formatted in *img*,** such as ViVo smartphone, we use the system mount command. The statement is as follows: `sudo mount -o loop system.img system/.` Listing 1 shows an example of the *mount* command.

Listing 1. An example of the *mount* command.

```
os.system('sudo mount -o loop ' + folderdat + '/system.img ' + tmp)
```

• **In terms of Android custom ROMs have *system* partition formatted in *ftf* (Flashtool Firmware)**, such as Sony smartphone, we unpacked these Android custom ROMs as a process described in Figure 4.

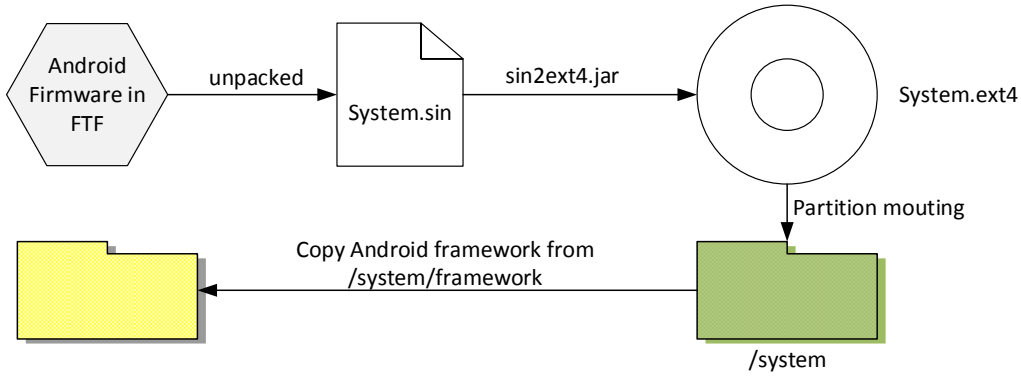

Figure 4. The process is used to extract Android framework from Android custom ROMs in FTF and SIN format

Figure 4 describes the extracting process that mentions four steps as follow:

Step 1: Extract the *.ftf* file to get the *system.sin* file by using Python's *zipfile* module.

Step 2: Convert *system.sin* file to *system.ext4* by using *sin2ext4.jar*. In this study, we developed *sin2ext4.jar* based on Flashtool tool [8].

Step 3: Mount the *.ext4* file to the folder */system*.

Step 4: Copy Android framework files in folder */system/framework*.

• **In terms of Android custom ROMs with */system* partition in *.dat* format**, such as CyanogenMod, Resurrection Remix. The extracting steps are shown in Figure 5.

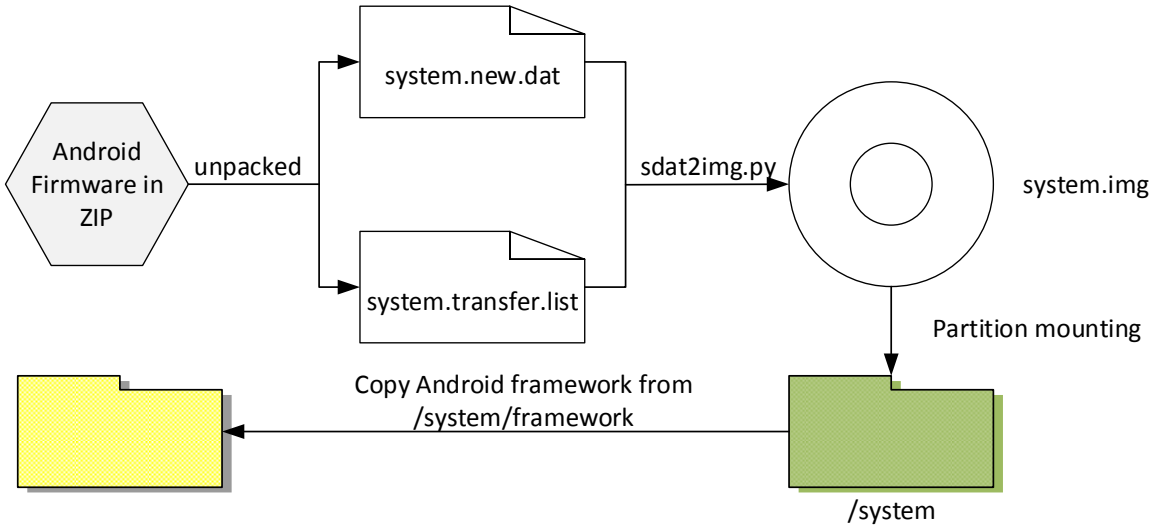

Figure 5. The process is used to extract Android framework from Android custom ROMs in ZIP and dat format

In the first step, we extract the Android custom ROM's ZIP file to get the *system.new.dat* and *system.transfer.list* files. To unzip the *zip* file, we uses the Python's zipfile module. Listing 2 shows an example of *zipfile* module using.

Listing 2. An example of *zippfile* module using

```
    with zipfile.ZipFile(rom,"r") as zip_ref:
        zip_ref.extract('system.new.dat', folderdat)
        zip_ref.extract('system.transfer.list', folderdat)
```

In the second step, we use *sdat2img.py*. This is an open source tool that unzips system.new.dat (sparse file) into system.img (raw image). This tool requires *system.new.dat* and *system.transfer.list* files as the input files. We use `python sdat2img.py <transfer_list> <system_new_file> [system_img]`, such as `python sdat2img.py system.transfer.list system.new.dat system.img`. Listing 3 shows an example of *sdat2img.py* used in this module.

Listing 3. An example of *sdat2img.py*

```
    call(["python", "./sdat2img.py", folderdat + "/system.transfer.list", folderdat + "/system.new.dat", folderdat + "/system.img"])
```

In the third step, we mount the *system.img* file to the folder */system* to copy the Android framework files, such as `sudo mount -o loop system.img system/.`

B. Framework Analyzer

This module is used to analyze the risk of leakage of sensitive information in the Application framework layer of Android custom ROMs. From Android frameworks extracted from the Framework Extractor module, this module conducts analysis of potentially sensitive information flows.

In this study we created a virtual main method to create a starting point for data flow analysis in the framework layer. However, unlike Android applications, the Android Framework has a lot of methods (each framework has an average of 65,100 methods). Therefore, we selected the difference methods between the Android Framework of custom ROMs needed to analyze and the Android Framework from Android standard ROMs before creating the virtual main method. In this study, Android standard ROMs are from Google official website or Android custom ROMs verified safe in previous analysis. The procedure is described in Figure 6.

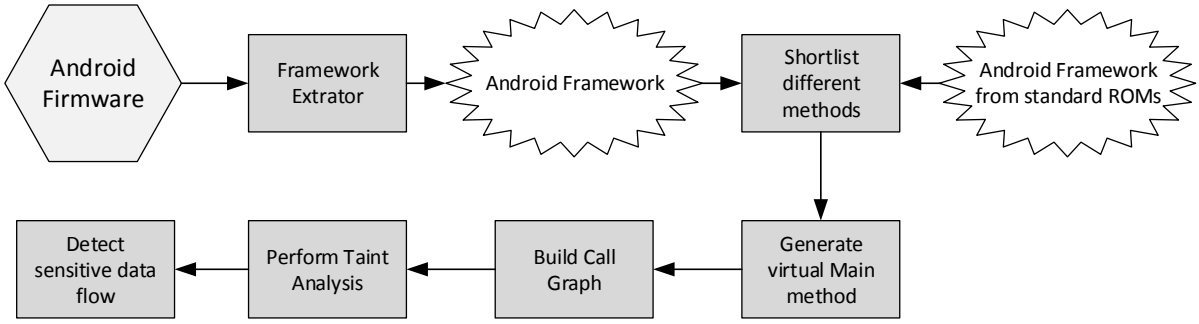

Figure 6. The process is used to extract Linux kernel from Android custom ROMs of ASUS

Step 1: List down methods in Android custom ROM which are different from methods of Android standard ROMs.

Step 2: Create the virtual main method for these selected methods in Step 1.

Step 3: After shortlisting the different methods and creating the virtual main method, we conduct sensitive data flow analysis in the Application framework.

We proposed a system based on FlowDroid [9] to analyze sensitive data flows in the Application framework (After creating the virtual main method in Step 3). Framework analyzer uses this system to detect all possible sensitive data flows that started by sensitive source system calls and ended by critical system calls. The list of these system calls is listed in SuSi [10].

C. Kernel Extractor

Kernel Extractor is a module used to extract the kernel of Android custom ROMs. The procedure for extracting the kernel from Android custom ROMs can be summarized in four steps as follow:

Step 1: Locate and extract the boot image from the Android custom ROM.

Step 2: Extract the boot image into difference components, including *zImage*.

Step 3: Split *zImage* into difference components, to extract the compressed kernel from the zImage decompression.

Step 4: Detect the compression type of the kernel and unpack it with the appropriate tools.

The steps of this process can be used for all firmware of different vendors, but there are some changed points which applied for different vendors because of different firmware structures and file formats. The Linux kernel extracting are showed in Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12.

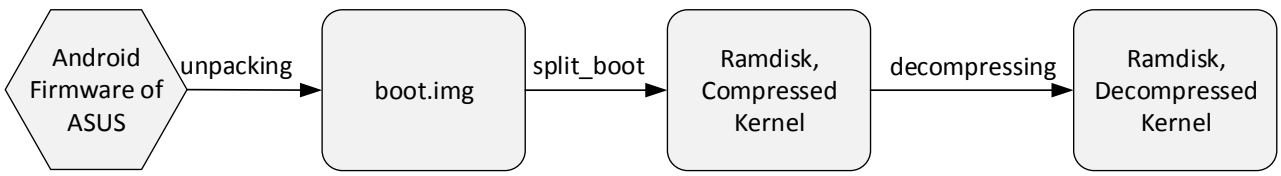

Figure 7. The process is used to extract Linux kernel from Android custom ROMs of ASUS

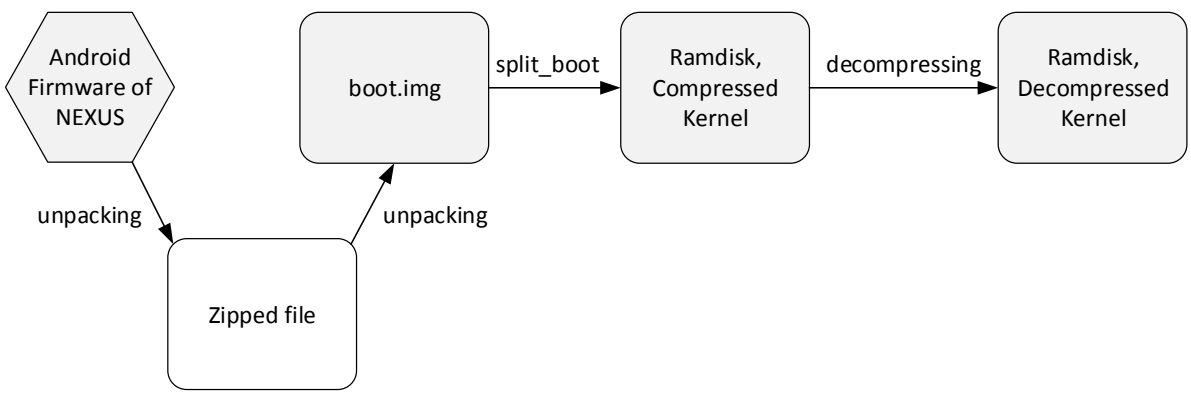

Figure 8. The process is used to extract Linux kernel from Android custom ROMs of NEXUS

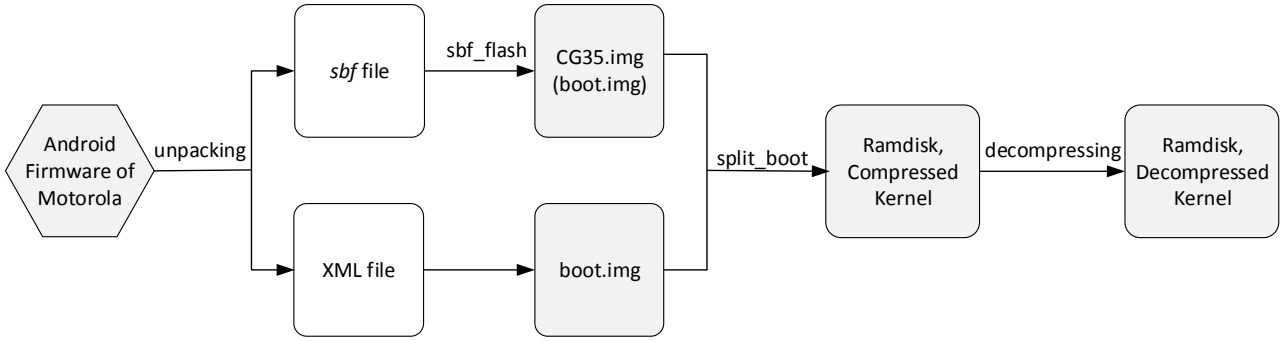

Figure 9. The process is used to extract Linux kernel from Android custom ROMs of Motorola

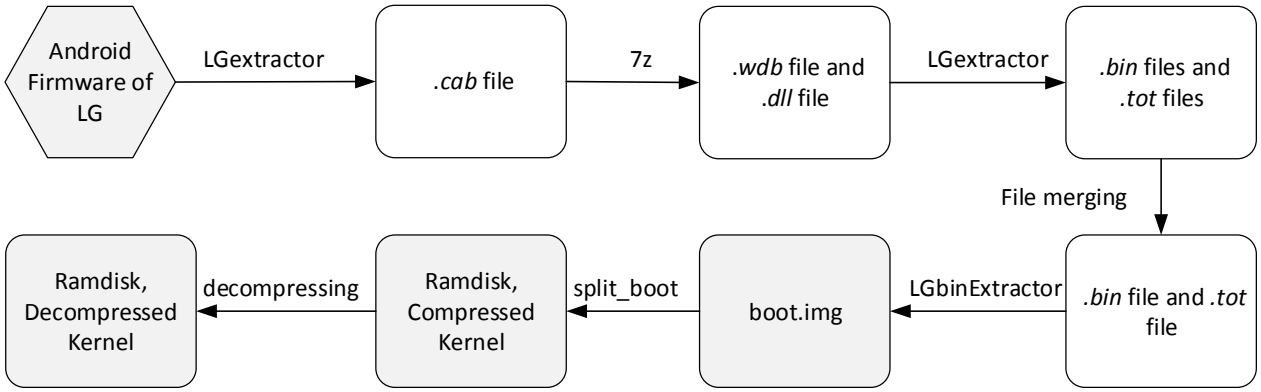

Figure 10. The process is used to extract Linux kernel from Android custom ROMs of LG

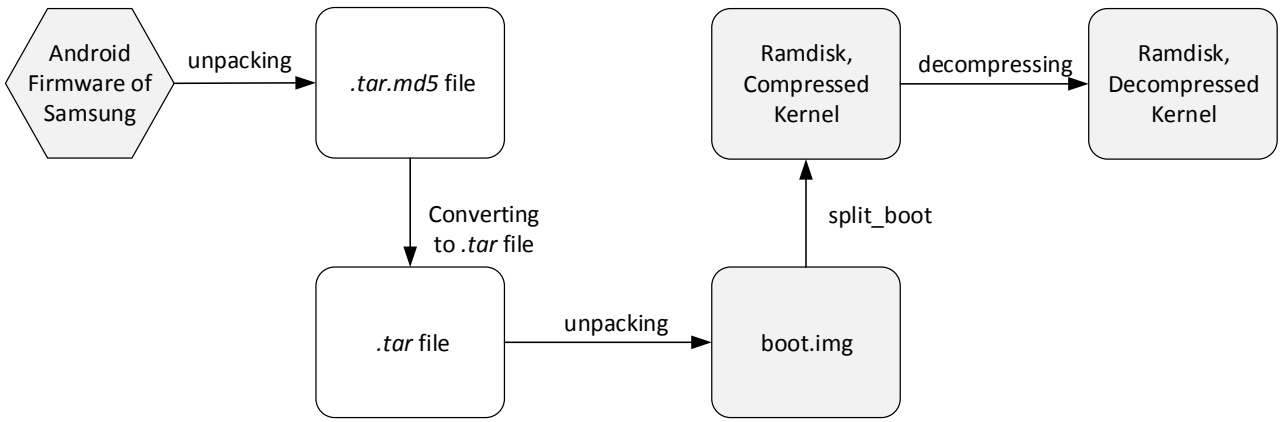

Figure 11. The process is used to extract Linux kernel from Android custom ROMs of Samsung

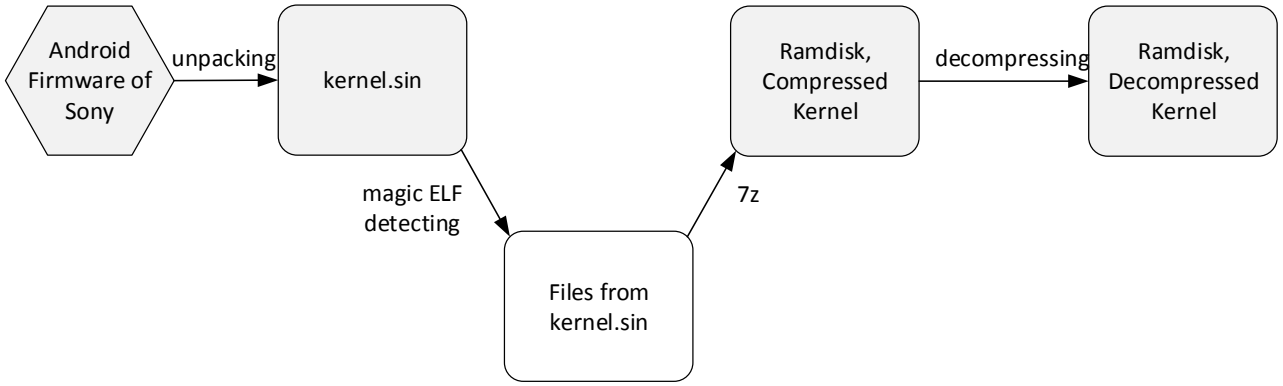

Figure 12. The process is used to extract Linux kernel from Android custom ROMs of Sony

D. Kernel Analyzer

The Kernel Analyzer is a module used to analyze sensitive information leakage risks in the Linux kernel of Android custom ROMs, and the default settings are likely to cause security risks.

In this study, we analyze the default setting such as USB debugging, WiFi, Bluetooth, allow to access with SU Access, checking the existence of abnormal IP addresses */etc/hosts* to detect the URL redirection. Default settings are stored in *default.prop* file in the Ramdisk extracted from Kernel Extractor or other files such as */etc/hosts*. Figure 13 shows the contents of *default.prop* file in the Ramdisk of *ASUS A66A* device.

```
ro.adb.secure=1
persist.sys.usb.config=mtp,mass_storage
persist.service.acm.enable=0
ro.secure=1
ro.allow.mock.location=0
ro.debuggable=0
persist.service.adb.enable=1
ro.zygote=zygote32
ro.mount.fs=EXT4
camera.disable_zsl_mode=1
dalvik.vm.dex2oat-Xms=64m
dalvik.vm.dex2oat-Xmx=512m
dalvik.vm.image-dex2oat-Xms=64m
dalvik.vm.image-dex2oat-Xmx=64m
ro.dalvik.vm.native.bridge=0
```

Figure 13. The contents of *default.prop* file in the Ramdisk of *ASUS A66A device*

In this study, we analyze default settings that listed in Table 1.

Table 1. Default settings is analyzed

No	Default settings	Value
1	SU access	Having SuperSU.apk
2	URL redirection	Having any abnormal IP address in */etc/hosts*
3	USB Debugging	ro.debuggable=1
4	ADB shell via WiFi	service.adb.tcp.port = a port number
5	ADB shell root mode	ro.secure=0

E. Reporter

Module Reporter is a module used for summarizing the analyzed results from modules: Framework analyzer, and Kernel analyzer. This module allows exporting the result of integrated analysis to *JSON* format to provide information to the user and the web application interface layer. Figure 14 shows an example of an analytical result from the Kernel Analyzer that the Reporter module provides to the user and the web interface layer.


```
1 {"_id":{"$oid":"59358695dc9cbe3360f3dd6d"},"hash":
"1da608989267158c4714a5c0c04f36895e200636","name":
"2.UL-Z008-WW-2.20.40.144-user","s01":"\u001b[31mWARNING\u001b[0m","s02":
"\u001b[32mOK\u001b[0m","s03":"\u001b[32mOK\u001b[0m","s04":
"\u001b[32mOK\u001b[0m","s05":"\u001b[31mWARNING - ramdisk, WARNING -
system\u001b[0m","s06":["\u001b[31mWARNING\u001b[0m",
"123.0.0.1\u0009\u0009 ","testdomain "]}
2 {"_id":{"$oid":"59358695dc9cbe3360f3dd6e"},"hash":
"80870ddabc9c3f0a65985d600bddad8d6019de37","name":
"3.UL-ASUS_Z00E-WW-12.8.5.106-user","s01":"\u001b[32mOK\u001b[0m","s02":
"Missing file build.prop","s03":"\u001b[32mOK\u001b[0m","s04":
"\u001b[32mOK\u001b[0m","s05":"\u001b[31mWARNING - ramdisk\u001b[0m","s06"
:["Missing file hosts","",""]}
3 {"_id":{"$oid":"59358695dc9cbe3360f3dd6f"},"hash":
"aa9872bcc7cb02bb0b7dbcee006a3c5e3cc6e6ac","name":
"8.UL-ASUS_T00F-WW-2.22.40.53-user","s01":"\u001b[31mWARNING\u001b[0m",
"s02":"\u001b[32mOK\u001b[0m","s03":"\u001b[32mOK\u001b[0m","s04":
"\u001b[32mOK\u001b[0m","s05":"\u001b[31mWARNING - ramdisk, WARNING -
system\u001b[0m","s06":["Missing file hosts","",""]}
```


Figure 14. An example report from Kernel analyzer

III. EVALUATION

F. Hardware configuration

The proposed system is deployed on the evaluating system that has the hardware configuration showed in Table 2.

Table 2. Hardware configuration of the evatuating system

Component	Value
CPU	Intel Core i7-4720HQ
Memory	16 GB
HDD	1TB, 7200 rpm

G. Dataset

In this study, we propose a dataset. This dataset has 10 Android custom ROMs that contain sensitive information leakage scenarios described in Table 3.

Table 3. Our samples in the proposed dataset.

No	Source Type	Sink Type	Note
1	Contact list	Socket	Leak the contact list via a socket to a service on the Internet.
2	Contact list	SMS	Leak the contact list via SMS.
3	GPS+IMEI	Socket	Leak user location via a socket.
4	GPS+IMEI	SMS	Leak user location via SMS.
5	IMEI	Socket	Leak device ID via a socket.
6	IMEI	SMS	Leak device ID via SMS.
7	Call log	Socket	Leak call log via a socket to a service on the Internet.
8	Call log	SMS	Leak call log via SMS.
9	Contact list	Internet	Leak the contact list via HTTP.
10	Call log	Internet	Leak call log via HTTP.

H. Experimental results

In this study, we tested the proposed system with the proposed dataset and 290 Android custom ROMs downloaded from the Internet. The experimental results show that the system accurately detects cases of sensitive information leakage at the Framework layer in our Android custom ROMs. Listing 4 describes an analysis result from the proposed system for the fifth sample in Table 3. The results of this analysis indicate that the system correctly detects the sensitive information leakage.

Listing 4. An example of the Framework layer analysis results of Android custom ROMs.

```
{
  "analyzeTime": {
    "date": {
      "year": 2017,
      "month": 4,
      "day": 27
    },
    "time": {
      "hour": 14,
      "minute": 33,
      "second": 50,
      "nano": 60000000
    }
  },
```

```
      "totalDuration": {
        "seconds": 19,
        "nanos": 671000000
      },
      "frameworkName": "Leak_IMEI_Socket.jar",
      "frameworkHash":
"1216c927c6d7206d1007fef5bd269f06e059a5740b667eeba129cd25f353fec7",
      "sizeInByte": 6576366,
      "numberOfAnalyzeMethods": 6,
      "analyzeMethods": [
        "<android.hardware.camera2.CameraManager:
android.os.PowerManager$WakeLock -
get0(android.hardware.camera2.CameraManager)>",
        "<android.hardware.camera2.CameraManager$1: void
onTorchModeChanged(java.lang.String,boolean)>",
        "<android.hardware.camera2.CameraManager$1: void
onTorchModeUnavailable(java.lang.String)>",
        "<android.hardware.camera2.CameraManager$1: void
<init>(android.hardware.camera2.CameraManager)>",
        "<android.hardware.camera2.CameraManager: void
<init>(android.content.Context)>",
        "<android.app.Activity: void onCreate(android.os.Bundle)>"
      ],
      "numberOfFlow": 1,
      "flows": [
        {
          "source": "$r9 = virtualinvoke
$r19.<android.telephony.TelephonyManager: java.lang.String getDeviceId()>()",
          "sink": "virtualinvoke $r2.<java.io.DataOutputStream: void
writeUTF(java.lang.String)>($r9)",
          "sourceMethod": "<android.telephony.TelephonyManager:
java.lang.String getDeviceId()>",
          "sinkMethod": "<java.io.DataOutputStream: void
writeUTF(java.lang.String)>",
          "pathLength": 2,
          "path": [
            "virtualinvoke $r2.<java.io.DataOutputStream: void
writeUTF(java.lang.String)>($r9)",
            "$r9 = virtualinvoke $r19.<android.telephony.TelephonyManager:
java.lang.String getDeviceId()>()"
          ]
        }
      ]
    }
```

In this study, we don't conduct sensitive data flow at Linux kernel layer of Android custom ROMs yet. However, we succeed in the Linux kernel extraction task. This task is very important to conduct sensitive data flow at Linux kernel layer in the future. The experimental results of this task are showed in Table 4.

Table 4. The experimental results of Kernel extractor

Vendor	Number of Android custom ROM	Number of Android custom ROM are detected correctly	The number of Linux kernel are extracted correctly	Time (second)
Asus	6	6	4	1.204
LG	8	6	7	140.326
Motorola	7	7	4	7.485
Nexus	6	0	6	21.42
Samsung	20	6	15	7.861
Sony	25	7	19	0.995
Others	151	-	138	4.515

The experimental results in Table 4 show that the proposed system detects correctly the vendor name of the corresponding Android custom ROMs with 44.5%. The success rate of Kernel extraction is 96.5%. The results show that Kernel Extractor module is effective in extracting Linux Kernel to analyze sensitive data flow by Kernel analyzer module.

Table 5 shows the results related to analyzing default settings of Android custom ROM in the wild. The results show that there are many Android custom ROMS contain SuperSU application and allow end user accesses to the system shell with root privilege. This finding leads to a conclusion that there are many security threats that are existing in Android custom ROMs in the wild.

Table 5. The results of analyzing default settings of Android custom ROMs in the wild

No	Type of default setting	Number of Android custom ROMs	The rate of ROMs contain security threat
1	Contain SuperSU.apk	23	7.9%
2	Enable "ADB shell over WiFi"	3	1.0%
3	Enable "USB Debugging"	15	5.2%
4	Enable "ADB shell root mode"	14	4.8%
5	*/etc/hosts* file contains any abnormal IP address	6	2.1%

IV. RELATED WORKS

Until now, there are some findings of researches on analyzing security risks can be hidden in Android firmware [5, 6, 11, 12]. DroidRay [5] is proposed to analyze vulnerable gaps on Android firmware by conducting scanning process on pre-installed applications and default settings in Android firmware. At app-level, they extract all apk files in Android firmware, then uploading these files to VirusTotal for online malware scanning. In DroidRay, they don't analyze data flows in Android applications, it just uses results of VirusTotal detect security threat in apk files. Beside that, they analyze default settings of custom ROMs such as information of */etc/hosts* or default shell mode.

Michael Grace et al. proposed Woodpecker [6] to detect sensitive data leakage in pre-installed apps of Android firmware. They used Soot [13] to analyze the sequences of calling functions in Android apps to address the data path from initiative functions in *AndroidManifest.xml* to sensitive functions, such as *sendTextMessage*. Woodpecker does not analyze inter-app communications, so it can not detect sensitive data flows through multiple applications. They also don't analyze security at Application Framework layer.

Aafe et al. proposed DroidDiff [11] to evaluate security threat in Android firmware by considering firmware configuration to determine its inconsistency from manufacturer's specification documents. DroidDiff does not analyze sensitive data flows in applications and Application Framework layer.

Zhou et al. proposed ADDICTED [12] to detect dangerous changes of device manager component in Android firmware. They use dynamic analysis technique to record calling systems in standard firmware (Android Open Source Project) [14] and custom ROMs. In the next step, they find the differences among firmware to determine what security threat containing in Android custom ROMs. They don't analyze sensitive data flows at application layer and Application framework layer.

V. CONCLUSION

In this study we proposed a method to detect sensitive data leakage in Android custom ROMs by analyzing sensitive data flow in Android Framework layer and default settings. The experimental results show that the system,

uitXFROID, of the proposed method accurately detects cases of sensitive information leakage at the application framework layer in our Android custom ROMs. It also detects security threats in several Android custom ROMs in the wild by analyzing their default settings. The experimental results also show that the proposed system can accurately extract Linux kernel of Android custom ROMs in the wild to support for sensitive data flow analysis at Linux kernel layer in the future.

ACKNOWLEDGEMENT

This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number B2016-26-01

REFERENCES

[1] Symantec. (2017, June 01). *2017 Internet Security Threat Report*. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[2] IDC. (2017, July 10). *Smartphone OS Market Share, 2016 Q3*. Available: http://www.idc.com/promo/smartphone-market-share/os

[3] Semantec. (2015, May 10). *2015 Internet Security Threat Report, Volume 20* Available: http://www.symantec.com/security_response/publications/threatreport.jsp

[4] (2016, 10/10). *Android Platform Architecture*. Available: https://developer.android.com/guide/platform/index.html

[5] M. Zheng, M. Sun, and J. C. S. Lui, "DroidRay: a security evaluation system for customized android firmwares," presented at the Proceedings of the 9th ACM symposium on Information, computer and communications security, Kyoto, Japan, 2014.

[6] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *The 19th Annual Network & Distributed System Security Symposium*, 2012.

[7] Virustotal.com. (2016). *Virustotal*. Available: https://www.virustotal.com

[8] (2017). *FlashTool*. Available: https://github.com/Androxyde/Flashtool

[9] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, *et al.*, "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," presented at the Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, Edinburgh, United Kingdom, 2014.

[10] S. Rasthofer, S. Arzt, and E. Bodden, "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks," 2014.

[11] Y. Aafer, X. Zhang, and W. Du, "Harvesting Inconsistent Security Configurations in Custom Android ROMs via Differential Analysis," presented at the USENIX SECURITY, 2016.

[12] X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations," presented at the Proceedings of the 2014 IEEE Symposium on Security and Privacy, 2014.

[13] E. B. Patrick Lam, Laurie Hendren. *Soot: a Java Optimization Framework.* Available: https://sable.github.io/soot/

[14] Google. (2017, March 10). *Android Open Source Project*. Available: https://source.android.com/

PHÁT HIỆN NGUY CƠ BẢO MẬT TRONG CÁC ROM ANDROID TÙY BIẾN BẰNG CÁCH PHÂN TÍCH THÀNH PHẦN APPLICATION FRAMEWORK VÀ CÁC CẤU HÌNH MẶC ĐỊNH

Nguyễn Tấn Cầm, Phạm Văn Hậu, Nguyễn Anh Tuấn

TÓM TẮT: *Hệ điều hành Android là hệ điều hành phổ biến nhất trong các năm gần đây. Đánh giá nguy cơ bảo mật trong hệ điều hành Android trở nên cấp thiết. Các nghiên cứu trước chủ yếu phân tích các ứng dụng trên thiết bị Android. Tuy nhiên, nguy cơ bảo mật có thể tồn tại trong các thành phần khác của hệ điều hành Android chứ không chỉ riêng các ứng dụng. Trong nghiên cứu này, chúng tôi đề xuất hệ thống uitXFROID cho phép phát hiện rò rỉ thông tin nhạy cảm trong thành phần applications framework và nguy cơ bảo mật từ cấu hình mặc định của các ROM Android tùy biến. Kết quả thử nghiệm cho thấy hệ thống phát hiện chính xác các trường hợp gây rò rỉ thông tin nhạy cảm trong các mẫu thử được đề xuất. Bên cạnh đó, hệ thống làm việc tốt với các ROM Android tùy biến được tải từ Internet.*