Forensic Analysis on Joker Family Android Malware

Chen Shi, Chris Chao-Chun Cheng, and Yong Guan
Department of Electrical and Computer Engineering
NIST Center of Excellence in Forensic Science - CSAFE
Iowa State University
cshi, cccheng, guan@iastate.edu

Abstract—**Android is the most popular operating system among mobile devices and the malware targeted explicitly for Android is rapidly growing and spreading across the mobile ecosystem. In this paper, we propose a hybrid analysis of Android malware to retrieve evidential data, generated from or accessed by such mobile malware, which can be adopted as critical evidence for civil and criminal cases. We target on Android malware from Joker Family where we collected and analyzed 62 recently discovered malicious apps, we found that: 11 apps access and store user's location information, 17 apps track user's SMS text messages and 58 apps send out user personal information to remote servers. Our proposed approach found that, evidence data including location, timestamp, IP address are still able to be identified from the local file system and logging system. Our main contribution in this research is to provide an effective forensic analysis report on Android malware that can extract critical evidence from the local file systems as well as system logs.**

Index Terms—**Android Malware, Mobile Security, Forensic**

I. INTRODUCTION

Smartphones and mobile devices are becoming indispensable as remote learning and working from home have increased during the pandemic, mobile malware is a serious concern and threat for users in terms of security and privacy. According to the report [1], Joker is one of the most prevalent malware families that continually infects Android devices. Google Play store [2] has removed more than 1,700 Joker apps since its initial release, however, despite awareness of this particular malware, it keeps finding its way to bypass Google Play app vetting mechanism by employing changes in its code, execution methods, and payload-retrieving techniques.

Currently, forensics analysis on Android malicious apps is quite limited, considering most of the research works are focus on malware detection. To the best of our knowledge, we are the first work focusing on large-scale forensic analysis on Android malicious apps of the Joker family on the Android platform. First, existing studies [3], [4] could not recover the file paths of the evidential data which is generated by Joker malware. Second, existing studies do not examine the frameworks and libraries which are unitizing by Joker malware which overlook critical evidence. Moreover, existing works only a collected limited number of Joker infected apps on a certain period, the analysis result might be uncompleted. Therefore, we proposed to perform an improved forensic analysis on Android Joker malware.

Our work: We proposed an improved forensic analysis procedure and technique of finding evidence generated by Android malicious apps from the Joker family. Our technique examine both system logs and file system. We have collected 62 Android malicious apps from the Joker family to evaluate our proposed approach. The evaluation result demonstrates that, through our proposed approach, all of the 62 created different types of evidence and stored in local storage.

The contribution of this work is summarized below.

- We identified and presented 72 evidence records with type and file path, which assisting forensic examiners with the digital investigation of Android device.
- Our work demonstrates that 58 apps collect and store user private information including location, timestamp, visited URL, etc.
- To the best of our knowledge, we are the first work that includes framework and libraries in the forensic analysis of Android malware.

II. METHODOLOGY

This section presents our approach to performing forensic analysis toward Android malicious apps. Building a comprehensive analysis result is a key challenge. We take an APK as the input for forensic analysis and de-compile the APK into three components, *dex* files, *ManiFest* & *Resources* files and libraries. We leverage DEX2IR [5] for code lifting from Dalvik byte-code to Smali and use Resources Parser to collect application information. The DCL tracker records the information of dynamically loading code and remotely downloaded files. Through the instruments of two class loader *DexClassLoader* and *PathClassLoader*, it monitors JNI APIs *load()* and *loadLibrary()* and other APIs that are related to file renaming. The Native Info Analyzer receives information from DCL tracker to generate the *Native Method Mapping* and *Native Activity Info*. Taint Engine maintains shadow registers to store the related taint tags and the propagation logicI follows the Union operation.

We note that a large number of static analysis tools–such as CHEX [6], FlowDroid [4] and HornDroid [3] have been developed to detect sensitive data flows between sources and sinks in Android apps, where a source is a method generate or access evidential data and a sink is the method send over the data or save the data in local storage. However, these tools

TABLE I
PROPAGATION RULE FOR ARM/THUMB INSTRUCTIONS

Instruction Format	**Propagation Rule**	
	TaintSet	***Path***
$mov\ R_d, C$	$TaintSet(R_d) \leftarrow \emptyset$	$R_d.Path \leftarrow \emptyset$
$mov\ R_d, R_m$	$TaintSet(R_d) \leftarrow TaintSet(R_m)$	$R_d.Path \leftarrow R_m.Path$
$unary - op\ R_d, R_m$	$TaintSet(R_d) \leftarrow TaintSet(R_m)$	$R_d.Path \leftarrow R_m.Path$
$binary - op\ R_d, R_m, R_n$	$TaintSet(R_d) \leftarrow TaintSet(R_m) \cup TaintSet(R_n)$	$R_d.Path \leftarrow R_m.Path + R_n.Path$
$binary - op\ R_d, R_m$	$TaintSet(R_d) \leftarrow TaintSet(R_m) \cup TaintSet(R_n)$	$R_d.Path \leftarrow R_m.Path + R_n.Path$
$binary - op\ R_d, R_m, C$	$TaintSet(R_d) \leftarrow TaintSet(R_m)$	$R_d.Path \leftarrow R_m.Path$
$store\ R_d, R_n, C$	$TaintSet(mAddr) \leftarrow TaintSet(R_d)$	$R_d.Path \leftarrow R_n.Path$
$load\ R_d, R_n, C$	$TaintSet(R_d) \leftarrow TaintSet(R_n) \cup TaintSet(mAddr)$	$R_d.Path \leftarrow R_n.Path$
$push\ regL, R_n, C$	$TaintSet(mAddr) \leftarrow TaintSet(R_i, R_j)$	$R_d.Path \leftarrow R_i.Path + ... + R_j.Path$
$pop\ regL, R_n, C$	$TaintSet(mAddr) \leftarrow TaintSet(R_i, R_j)$	$R_d.Path \leftarrow R_i.Path + ... + R_j.Path$

could not provide the file paths where the data are written to which is critical information to extract the evidence. For instance, these tools could detect that certain app will collect GPS locations and save them to the file system, but they do not record the file paths where the location information will be written to.

Another significant improvement against approaches mentioned above is to include framework and library analysis which was overlooked in the prior studies of forensic studies. In order to bypass security check, one of the techniques malicious apps utilized is uploading clean app bundled with adware and malicious SDKs, we believe analyze the associated frameworks and libraries are necessary in order to provide a completed evidential data list.

A. Forensic analysis on Android malware

In this work, we focus on four types of evidential data including location, SMS text message, timestamp, and visited URL. To be noted, our source method list is combined from existing tools including FlowDroid [4], and HornDroid [3] which is not limited to these four types of evidential data. However, we found that the combined source methods for evidential data are still not complete, we can update the sources and extend tags if a new source method is found.

Next, we discuss the sources of the four types of evidential data.

1) **Location**: We included 40 source methods for a location type of evidence. Location tracking is considered highly invasive, it tracks users all the movements and who you associate with. Location type of source methods include precise GPS location (e.g, *LocationManager.getLatitude()*) which obtain from satellites within a limited range and coarse-grained location(*e.g, Responder Location.ACCESS_FINE_LOCATION*) determined by WiFi and cellular data. According to our findings, the two main types of tracking technology are GPS and Beacon Bluetooth, which can detect a person's location within 30-40 feet.

2) **Text message**: We included 2 source methods for text messages. Some of the malicious apps also require users to enter their mobile numbers in order to receive alerts and subscription notifications by text message.

3) **Timestamp**: We included 17 source methods for the timestamp. Apparently, timestamps can be used to determine user's activities and served as important evidence in other cases.

4) **Visited URL and IP address**: We included 5 source methods for visited URL and IP address. We would like to gather a list of URL and IP addresses of remote servers where the evidential data sent to, in this way, forensic investigators would be able to retrieve information back from a remote server.

B. Forensic analysis on framework and libraries of Android malware

As we mentioned above, Joker malicious apps utilize dynamic loading techniques and download malicious SDKs at run time to bypass the security check. When building the app, all the resources from the libraries are moved to one `res` folder which is included in the `<package name>.apk`. By checking the `res` folder and `Android Manifest.xml` In the library forensic analysis, we examined the artifacts that may be created in the local file system. The target of the library forensic analysis is to identify the location of evidence and define the rule of taint propagation that can help to generate analysis results.

We apply forward analysis and create a work order with method signature and library file, the Library Function Summary Builder first identifies the code range of the corresponding library function of the method. Then it checks every instrument to propagate the taint tags according to the propagation rules. We create a Library Method Mapping data structure to identify the callee of the Java method in app code, where the key is the Java method signature and the value is the corresponding function name and associated library. Whenever the analyzer encounters any method/function invocation, it will check with the list of source and sink API. If so, it will add taint to the proper argument or record the associated file path where the tainted argument was written to. When the analysis is over, we extract the summary of the callee function with a taint tag related to each argument and return the node to build the summary.

TABLE II
EVIDENCE REPORT OF COM.GOODERS.PDFSCANNER.GP

Package name	**Installs**	**Evidence Type**	**Evidence File Path**	
			Logging system	***File Paths/ Urls***
com.gooders.pdfscanner.gp	50K	TimeStamp	*	$/data/user/0/com.gooders.pdfscanner.gp/shared\_prefs/acra\_criticaldata\_store.xml$
		Visited URL	*	https://logger-dyu6ojodva-uk.a.run.app
		Cookie	*	
		Location		$/data/data/com.gooders.pdfscanner.gp/app\_lib/realm-jni$

III. EVALUATION

In this section, we aim to evaluate the forensic analysis results of Joker infected apps. Case studies are presented to discuss the frequent types of evidence found in our large-scale analysis result, which corresponds to the result of forensic analysis of malicious apps and dynamic loading SDKs respectively. At last, the analysis result of large-scale malicious apps of the Joker family is presented, which may assist forensic practitioners with performing evidence extraction in the real world.

We obtained 62 Joker infected apps from 11 different Android app markets. We leverage Monkey [7] to generate random user events for these apps. Since some apps are larger and take a longer time to be fully analyzed and we aim to generate a completed analysis result, we set an 8-hours timeout for each app.

Table II summarizes our analysis results for each type of evidential data. A reported file could include at least one type of evidential data including location, visited URL, timestamp, and text message. If a file path contains any patterns of the *¡timestamp¿*, *¡UUID¿*, and *¡intent¿*, we will treat it as a dynamic file path, all other kinds of file paths are treated as static file paths.

TABLE III
SUMMARY OF THE ANALYSIS RESULTS FOR 62 JOKER INFECTED APPS

Evidence Type	**App**	**Evidence File Path**	
		Static File Path	***Dynamic File Path***
Location	11	9	2
Time	30	30	2
Text Message	17	19	1
Visited URL	21	20	3
Others	47	45	7

A. Case Study of Analysis Result

We utilize both static and dynamic analysis to discover the artifacts stored in the file system as well as the logging system. In our large-scale evaluation result, we found that 94% of Joker infected apps do leave forensic artifacts at local storage which may be extracted in the future for forensic study. To understand the evaluation result, we use a case study of All Good PDF Scanner version 2.8(`com.gooders.pdfscanner.gp`) as an example to elaborate the detail of the report.

As shown on Table II , two places in the logging system and one place in the file system are reported to have evidences stored. A brief discussion of how these evidences would help forensic practitioners in real cases is presented as followed.

First, the report shows that log messages contain both timestamp and visited URL. If we combine the URL with timestamp together, we could reconstruct the user's browsing history. Since extracting log messages does not require root permission, forensic practitioners can retrieve user browsing history traces on a non-rooted device if this Joker infected app is installed.

Second, the cookie is also reported in the logging system. Note that cookie is a simple key-value pair data sent from the website and each record of cookie associates with a website URL, therefore, the user's browsing history can be obtained by mapping the cookie with its corresponding last access timestamp together.

Third, the report shows that timestamp type of evidence is stored under app's internal storage with the file path of */data/user/0/com.gooders.pdfscanner.gp/shared_prefs/acra_criticaldata_store.xml* and location can be extracted from */data/data/com.gooders.pdfscanner.gp/app_lib/realm-jni*. Both the timestamp and location can be found at local storage, correlate the location with timestamp can provide user's geo-location activity for forensic analysis.

B. Large-Scale Forensic Analysis Result of malicious apps of Joker family

Since Joker infected apps are bundled with different functionalities and have their own specific implementations, the storage location and structure of data that contains evidence might be variable. Therefore, we evaluated the effectiveness of our proposed approach on collected Joker infected apps. We note that the evaluation result can significantly improve the forensic practitioner's work from both time efficiency, precision as well as completeness. By crawling from Google Play Store [2], and 63 other Android app markets, we collected 62 Joker infected app with more than 472,000 download times.

Joker Malware target on user from the specific countries by checking Mobile Country Codes (MCC). After downloading the second stage payload which is the core component, it periodically requests new commands from the remote C&C server. One of the malicious behaviors of Joker malware is the premium subscription which will be billed to the user's wireless account. When the Joker app receives commands, it proceeds to open the URL, retrieve the authentication code to sends an SMS message for the premium service without user consent.

From our findings as shown in Table III, 58 out of 62 malicious apps from the Joker family are identified to store evidential data either in the local file system or logging system. Among these 58 apps, 11 apps store user's location information in app internal storage; 30 apps periodic record device timestamp and store it in both logging system and file system; 17 apps store user's SMS text system and 21 out of 62 malicious apps do not prevent the visited web-page URLs appearing in log messages. Particularly, 58 of them are storing other kinds of evidential data, which provides great chances of extracting evidence for forensic analysis in the real world.

In addition, we notice that 17 Joker infected apps dynamically loading the same dex and SDK, where the timestamp is stored under the same pattern of file path */data/data/package name/shared_prefs/rn_push_notification.xml*. This result determines that if other apps utilize this framework, most likely, they're infected with malicious behaviors and forensic investigators could extract timestamp evidence from the same file path. While most of the timestamp and visited URL are reported to be stored in plain-text format, there are still a few special cases different from that. For instance, app Smart Scanner (*com.rapidface.smart.scanner*) keeps visited URL in ASCII format that can be found at */data/data/com.rapidface.smart.scanner/shared_prefs/react-native-device-info.xml*. Similar situation can be found in the analysis result of app *com.wallpapers.dazzle.gp* and app *com.saying.wallpaper.bb*.

Moreover, we find that app *com.board.picture.editing* write deviceID and timestamp into Android default file manager instead of its own internal storage directory. The full path is */data/data/android.support.v4.content.FileProvider*. Such operations can be achieved either through the public *Content-Provider* or *Inter-component communication* which belongs to Android inter-app communication system.

Overall, the large-scale evaluation result verifies the effectiveness of our proposed approach working on forensic analysis of malicious apps of the Joker family. Framework and library analysis is demonstrated as a significant component of our approach, which is beneficial for apps leveraging the same code base. Moreover, the analysis result of the file system could help forensic investigators quickly locate and extract the evidence.

IV. RELATED WORK

Android malware forensics, a new branch in the digital forensic research community with increasing usage and development of malware. Among the static analysis tools, Flowdroid [4] is a well-known data-flow analysis framework for detecting potential privacy leakage of Android applications. It creates a dummy main method for an app, detects and propagates the taints through a flow and context-sensitive algorithm. However, FlowDroid did not cover the native method invocation and treat all native libraries as black-box settings.

Chex [6] is designed to detect component hijacking problems in the Android platform and is the first static analysis tool considering different types of entry points of an Android app. It is built on top of Wala [8] where it first parses app code and constructs app-split. Each app-splits is a code segment that can be reached from an entry point. Then it builds the data-flow summary for each of the app-split utilizing the data-flow engine from Wala. Finally, it constructs the possible information flows by linking app-splits summaries in all possible permutations.

In conclusion, in order to treat the file system as a type of sink and further discover the file paths where data are written to, we decided to leverage both static and dynamic analysis on the forensic analysis of Android malware.

V. CONCLUSION

This paper conducted a comprehensive forensic investigation on evidence generated by 62 Joker infected apps on the Android platform. The approach is composed of both application and framework analysis, examining evidential data stored both in logging system and file system. Location information, timestamp as well as other kinds of forensic artifacts can be extracted from these malicious apps. Our findings provide meaningful results for forensic practitioners since Joker malware is one of the most active and prevalent malicious apps. Manual verification is made to authenticate the correctness of the analysis result. The case study of sample application presents the detail of forensic artifact findings, the evaluation of large-scale Joker infected apps demonstrate that our proposed approach can be widely applied for evidence discovery.

VI. ACKNOWLEDGMENTS

This work was partially supported by NIST CSAFE under Cooperative Agreement No. 70NANB20H019, NSF under grants No. CNS-1527579, CNS-1619201, CNS-1730275, DEB-1924178, ECCS-2030249, and Boeing Company. We appreciate anonymous reviewers for their valuable suggestions and comments.

REFERENCES

[1] Mobile app store marketing intelligence.

[2] Google play store.

[3] Stefano Calzavara, Ilya Grishchenko, and Matteo Maffei. Horndroid: Practical and sound static analysis of android applications by SMT solving. *CoRR*, abs/1707.07866, 2017.

[4] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *SIGPLAN Not.*, 49(6):259–269, June 2014.

[5] baksmali 2017.

[6] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 229–240, New York, NY, USA, 2012. ACM.

[7] Ui/application exerciser monkey.

[8] Wala - static analysis capabilities for java bytecode and related languages.