EasyChair Preprint

№ 186

Andrologger: Collecting and Correlating Events to Identify Suspicious Activities in Android

Pradeep Tiwari and T Velayutham

May 30, 2018

Andrologger: Collecting and Correlating Events to Identify Suspicious Activities in Android

Pradeep Kumar Tiwari
Central Research Laboratory
Bharat Electronics Limited
Bengaluru, India
pradeepkumart@bel.co.in

T Velayutham
Central Research Laboratory
Bharat Electronics Limited
Bengaluru, India
tvelayutham@bel.co.in

***Abstract*— With the tremendous increase in android smartphone users and easy availability; industry, government and enterprises are trying to tap into these device's usage possibility for organization's work purpose. This could significantly reduce the costs and add capabilities earlier un-existed for the enterprises. However, organizations should be prepared to deal with the risk associated with it. These devices will contain plethora of information and data regarding work which when compromised can pose significant challenge to internal investigations comprising policy violations, data theft, intellectual property theft, sabotage, social engineering attacks. In android forensics the earlier approaches and capabilities usually are limited to physical access with forensic tools, though useful but not exposed to full potential. In this paper, we propose a tool, Andrologger which has the capability of automatically collecting data and user events from the device and sent to enterprise server for monitoring and analysis. This data can then be co-related with various activities to suggest a suspected user, beforehand. Andrologger will help investigators and analysts with the real data from the user and their activities and can also be used for user behavior analysis during work-hours by the organizations.**

Keywords—Andrologger; Data Collection; Suspiciousness; Enterprise; Android Forensics; BYOD

I. INTRODUCTION

Android's availability in abundance [12] and flexibility to tailor its use has opened a lot of opportunity to the enterprises to adopt it in their environment. Some organizations have already adopted it as part of their BYOD policies [Citrix, 2014] [11] and at the same time other government, enterprises and organizations are forced to adopt it in order to effectively survive in the competitive environment. Bring Your Own Device (BYOD) brings several benefits [10] to the organization including cost effectiveness in implementing certain policies, technology familiarity as users are adept to it. However at the same time it poses certain challenges to deal with. The challenges include physical security of the device, data theft from the device including intellectual property theft, sabotage, espionage and social engineering attacks.

Android OS uses custom linux [13] flavor for their platform, which uses sandboxing technique to prevent inter application data leakage and uses permission based protection [14] to the critical resources of the hardware. However there exists literatures [15] [16] suggesting data leakage from android devices through the use of android permissions. Ensuring the security of data lying on the phones becomes major concern for today's user through strict enforcement of MDM policies. There exist few Mobile Device Management (MDM) systems for the management of devices in enterprise environment [Pettey, 2012] [17] and it is expected that 75% of enterprises will be going to adopt third party MDMs in their premises. Researches on existing MDMs show that they lack many features at the enterprise level. According to the 2014/2015 survey from Computer Security Institute Computer Crime and Security [18], 61.5% of the respondents reported that internal audits were carried out as part of security mechanism in their organizations. Moreover, 44% of the partner companies reported that there were user-content monitoring programs and data-loss prevention programs in place (Richardson, 2015) [18]. These indicators show that the organizations are aware of internal threats and also taking measures to mitigate them. However, they lack the technology to monitor them automatically and continuously. This unavailability of technological option for its solution has led to the project of Andrologger.

The research work makes the following contributions:

- Automated data collection tool covering user activities over a period of time with customization options.
- Extending the list of data sets to include events from Bluetooth, MMS, Gallery and their combinations.
- Detecting Suspiciousness through conjunction of various data sets through an UI provided at the server as well as on user device.
- Tested the application in a team of 5 for 10 days and identified the general usage trends gaining insights into the user behavior and later classifying them.

The andrologger application though asks for exhaustive list of permissions from android system in order to work, but at the same time it gives the user transparency over the data

collection. The data was collected from the Moto G4 Plus android phone running with Marshmallow version of android. It was then sent to the central server running Ubuntu 14.04 installed with MySql 5.7, Apache 2.4.17 and PHP 5.0. The application gives support from version 4.0 having 97.4% users [9] to android 7.0. The andrologger application is available on github [19] for its full-fledged functionality testing.

II. Background and Related Work

From android's framework, in order to get certain values it has to go through the layered architecture which protects the resources using permission. For example, In order to get GPS latitude and longitude the API *LocationManager* class from application framework level should interface itself with C library *libgps.so* at the middleware level, which will finally call the kernel driver responsible for getting the current location. The sequences of events are as follows:

Application → Application Framework → Middleware Library → Kernel Driver

Android security framework at different levels is describes below:

a) **At Application Level:** Permissions are used for protection of resources of device as well as resources of the application itself. For example, if two applications A1 and A2 with permissions P1 and P2 respectively want to use each other's resources, A1 has to declare in its manifest file permission P2 and P1 in case of A2. Hence protecting resources at application level.

b) **At Framework Level:** All applications run in sandboxed environment. The inter process communication is implemented using the component called Binder which is different from traditional linux. It is optimized for low powered devices.

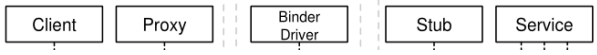

Fig1: Binder Communication Model[1]

c) **At Linux Kernel Level:** At kernel level the applications are sandboxed with uids and gids (User ID and Group Ids). In android each application is considered as a user in OS perspective. OS makes fair separation of privileges between resources at the application level.

There was not much literature, we could find in the area of android forensics. However the book [3] by A Hoorg on android forensics gave real insight into the techniques and approaches. One research related to android forensics, carried out by Timothy Vidas et. al. [2] on android forensics concentrates on taking the image of the device through special boot and collect data for investigation, which is useful in more sensitive cases, but can't be pertained to enterprise environment. Another interesting study on whatsapp forensics [4] from InfoSec could increase the current work and facilitate with more data. Other approaches [5] [6] focused on anti-forensics techniques deployed on android smartphones to examine their effectiveness. Studies [7] [8] majorly focused on data collection from physical devices or from specific application.

III. Proposed Design and Implementation

A. System Architecture

The proposed system architecture can be referred from following diagram where each layer gives an abstraction. The andrologger service starts running after user accepts the consent of being monitored in the enterprise environment. Once user consent banner is accepted, andrologger's WatcherService starts executing in the background performing storage, collection and transfer events. The service keeps an eye on all the events defined in the schema of the application. The service keeps collecting the data and storing in local sqlite database, which is flushed out once transferred to the enterprise server.

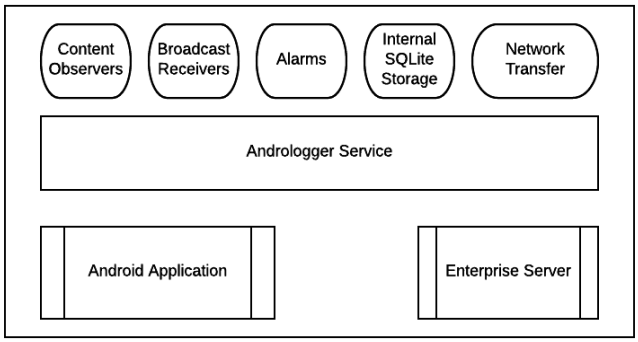

Fig2: Andrologger System Architecture

Table I lists the watchers of the andrologger application which extends the earlier work on monitoring discussed in the related work.

TABLE I. Components Used in Andrologger

	Data Set	*Collection Components Used*		
		Alarms	***Content Observers***	***Broadcast Receivers***
1.	Device Id (e.g. IMEI)	✓		
2.	GPS Location	✓		✓
3.	GPS Settings (On/Off)	✓		✓
4.	Device Accounts*			
5.	Application Install/Uninstall			✓
6.	Blutooth Status			✓
7.	Browser Navigation History	✓		
8.	Browser Searches	✓		
9.	Calender Events	✓	✓	
10.	Call Logs (Incoming/Outgoing/Missed)		✓	
11.	SMS (Incoming/Outgoing)		✓	✓

	Data Set	Collection Components Used		
		Alarms	*Content Observers*	*Broadcast Receivers*
12.	Screen Lock Status (e.g Unlocked)			✓
13.	Third party Application Logs (logcat)	✓		
14.	MMS Status	✓		✓
15.	Pictures Added		✓	

** To get device accounts information there exists direct APIs to access them, there was no need to register any alarm or content observer for the same.*

Alarms are used for time-based operations outside lifetime of the application. For example, an alarm is used to start a long-running task, such as running a service once a day to download a weather forecast. They help fire an activity at the said time. With the help of alarms we achieved tracking of the events periodically.

Content Observers listens to the changes in the data. So whenever there is change in data of either of the client, it records that.

Broadcast Receivers attends to system wide notifications sent by the system. For example, the battery status is sent to each and every application to respond accordingly.

B. The Approach

The above approach diagram gives the flow of events from one activity to another. The consent banner is popped and depending on the input from the user the application andrologger responds to the activity. If the consent banner is accepted, andrologger runs in the background traversing for the Call logs, Browser, Contacts, GPS, Calender, Device Account, Package Manager, Messaging, Screen On/Off activities and store them in a local sqlite database. Same time the inference engine (Suspiciousity Watchdog) runs for the detection of suspicious activities. If a particualar event/activity is found to be suspicious, it is flagged to enterprise server/user. And periodically the stored log database is flushed to the enterprise server and removed from local database to avoid the memory overhead in local system.

Some rules for suspiciousness are as follows:

a) From ContactWatcher, if a new contact is added, which was not existing earlier in phonebook, provided with the phone, may pose a concern for the investigator.

b) CallWatcher which monitors incoming and outgoing calls from the device can help in identifying calls made to unknown or unidentified numbers through a normal search.

c) Similar to CallWatcher, SMS and MMS detectors identifies messaging communication, which could be very helpful for investigators, as this module also gives the body of message and its content as well. However sometimes, the timestamp of sender may not match with the local timestamp. So the content with time zone will form a unique event.

d) One of the most important event watcher is LocationWatcher which records changes in GPS co-ordinates. For example, two events, one a picture has been taken and, second the GPS co-ordinates are also recorded. These two events can help in identifying from which place the picture was taken. If found in a confidential zone, immediately reported to the investigator.

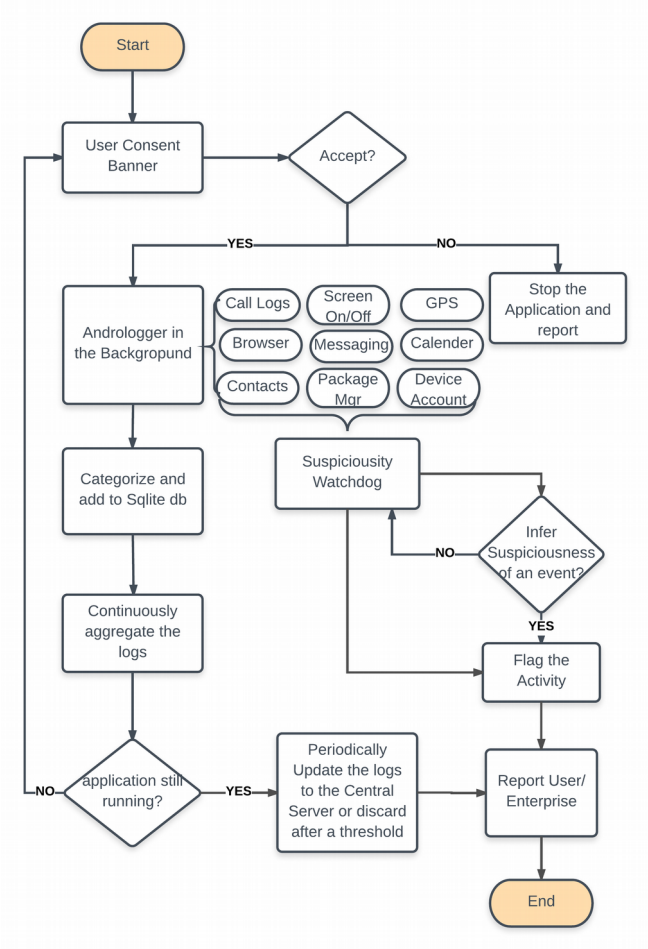

Fig3: Proposed Methodology and the flow chart

1) User Consent

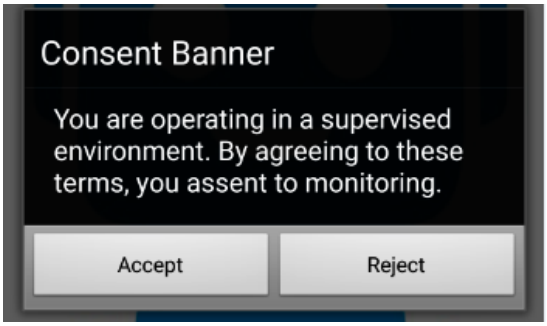

Fig4: Consent Monitoring Banner

User consent dialog box is being implemented with AlertDialog builder of android API. Once user accepts the

1. Zhauniarovich, Yury. "Android Security (and Not) Internals." Web,

consent banner, it is logged and WatcherService starts running in the background as shown in the code below.

```
Log.v(TAG, "User Consent Banner - Accepted"); // Close GUI
finish();
//Start the service
if (!isServiceRunning()) {
    Intent serviceIntent = new Intent("com.andrologger.WatcherService"
    startService(serviceIntent);
}
.setNegativeButton("Reject", (dialog, whichButton)  › {
        finish();
        if (isServiceRunning()) {
            Intent serviceStopIntent = new Intent("com.andrologger.WatcherService"
            stopService(serviceStopIntent);
        }
```

Fig5: Consent Banner Accept and Reject Options

And if user rejects the consent agreement the service WatcherService, if running, is stopped and logged to the administrator.

2) Data Collection Components Stored/Transferred

a) **Device Id and Accounts* (e.g. IMEI/ IMSI):** The *getDeviceId()* method from the TelephonyManager API class was used to directly access phone's device ID. And the *AccountManager.get().getAccounts()* method was used to get the associated accounts on the phone. In order to get the accounts information *android.permission.GET_ACCOUNTS* and *android.permission.READ_PHONE_STATE* permission was to be declared in the *AndroidManifest* file.

b) **GPS Location:** The GPS location of the phone was retrieved using *LocationManager* API of android framework. The *getLastKnownLocation()* method of the API class which runs every hour, by default give GPS co-ordinates periodically. An alarm is to be registered for the same. In order to access these values *android.permission.ACCESS_FINE_LOCATION* permission was to be declared in the *AndroidManifest* file of the application.

c) **GPS Settings (On/Off):** In android system, whenever a setting related to GPS is changed manually (enabled/ disabled), a broadcast notification is sent to whole system, which is recorded by *LocationWatcher*.

d) **Application Install/Uninstall:** Whenever an android application is installed or uninstalled a system-wide broadcast notification is sent. There is no need to declare permissions in the manifest file. Although it should register a broadcast receiver and intent filter for *Intent.PACKAGE_ADDED* and *Intent.PACKAGE_REMOVED* events.

e) **Bluetooth Status:** To get the status of Bluetooth, *BlutoothAdapter* API should be implemented. It indicates status and support for the hardware. And the API should be accompanied with the permission, *android.permission.BLUETOOTH.*

f) **Browser Navigation History:** From android's built in browser, URLs are accessed through an alarm, registered to its API. To avoid battery drainage and power saving, the content provider accesses the bookmarks history through *android.provider.Browser.BOOKMARKS_URI* URI every six hours, by default. To gain access to this data permission, *android.browser.permission.READ_HISTORY_BOOKMARKS* need to be declared in the manifest file of the application.

g) **Browser Searches:** Browser searches are not the URLs, they are searches made with the keywords into the inbuilt browser. There is no data set available for this to be retrieved with ContentObservers. Browser searches can be collected through an alarm registered as process running every six hours. The same permission *android.browser.permission.READ_HISTORY_BOOKMARKS* is needed to get access to browser searched.

h) **Calendar Events:** Android gave support to access calendar events from API level 19 onwards (Ice-cream Sandwich). An alarm was configured to scan content provider URI, *content://com.android.calendar/event_entities* every twelve hours. Since this content provider doesn't give direct access to calendar database, we created a separate table to collect the same, which is populated with existing events from the google calendar during its first run and keeps itself updated against new events. In order to get access to calendar data, permission, *android.permission.READ_CALENDAR* need to be declared in the AndroidManifest file.

i) **Call Logs (Incoming/Outgoing/Missed):** Telephony related details can be retrieved using two methods, first, via broadcast notification and second, using content providers. Whenever there is change in state of the phone (idle, ringing or vibrating) a system wide broadcast notification is sent which can be retrieved through broadcast receivers. However, in few cases of missed calls, broadcast receiver couldn't identify the change of state and went unnoticed. So, to ensure all the logs are captured and avoid false positives content URI, *android.provider.CallLog.Calls.CONTENT_URI* was used to get access all the logs Incoming, Outgoing, Missed and even the timestamp information of calls. To get access the call logs database, *android.permission.READ_CONTACTS* permission was needed to be declared in the AndroidManifest file.

j) **SMS (Incoming/Outgoing):** For incoming messages, a broadcast receiver was registered to retrieve them. The incoming messages contain extra information such as sender's address, timestamp and its content. And for outgoing messages an implicit content observer was registered with the URI, *content://sms/out* and the change notifications are

observed from “draft” to “pending” to “sent”. Andrologger records only sent message statuses to avoid duplicate entries. In order to get the ability to collect the SMS information, permissions *android.permission.RECEIVE_SMS*, *android.permission.READ_SMS*, and *android.permission.READ_CONTACTS* needed to be declared in the AndroidManifest file of the Andrologger application.

k) **Screen Lock Status (e.g Unlocked):** To get screen lock status (e.g. Locked, Unlocked, or “off” states) no extra permission was required. The intent-filters should be made to handle *Intent.ACTION_USER_PRESENT, Intent.ACTION_SCREEN_ON, and Intent.ACTION_SCREEN_OFF* intents through broadcast receivers.

l) **Third party Application Logs (logcat):** Android has a built in logging system in its SDK, *logcat* in *android.util.Log*. Logcat temporarily stores the logs in */dev/log/main* which is discarded as soon as a new log is available to avoid overhead on the system. Andrologger’s logging system for third party application (any application which is not registered in the Package Manager) was made sophisticated to handle logs efficiently with the help of tags, debug mode disabled logging etc. An alarm was registered to collect logs every hour. Andrologger required the permission, *android.permission.READ_LOGS* in AndroidManifest file of the application to get access of logs.

m) **MMS Status:** This is challenging task in android to extract multimedia messages from the device. For incoming MMS messages, it was easy to get information about through broadcast receivers. However, for outgoing MMS messages, when tried with content observers URI *content://mms* activity got crashed, but gave information about sender, receiver and timestamp.

n) **Pictures Added:** Pictures can give a lot of information about an event. They contain timestamp, location, description, tagged people etc. Usually short information can be gathered using broadcast receivers, as a notification is sent throughout system when a picture is clicked. However to get more information a content observer is registered to the android gallery through URI, *android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI*. Whenever a change notification is sent, gallery is scanned for changes and recorded in the events table of sqlite database. There is no need to declare any additional permission in the application’s manifest file.

C. Data Flow Diagram

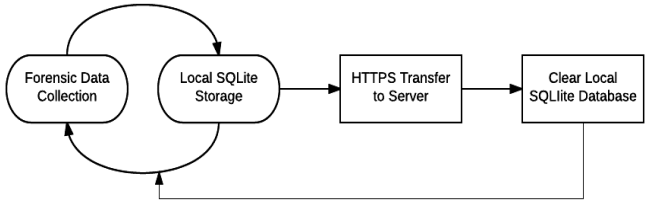

Fig6: Data Flow Diagram for Andrologger

In the enterprise environment, regular update to server is equally important. The forensic data is being collected and stored locally in sqlite database, which is updated periodically as set by *andrologger.properties* file in app’s assets directory.

```
server_url=http://192.168.102.187/uploads/upload.php
# SSL Cert
ssl_cert_name=server.crt
# App Install/Removals
app_install_removal_interval=1
# Calendar Events Added
calendar_interval=43200000
# Call Logs
call_log_interval=1
# Contacts Added
contacts_interval=1
```

Fig7: Setting Intervals and Options for Data Collection (andrologger.properties)

Once the application is configured, the local database is sent over secure http protocol to the server, specified in the url. And once the acknowledgement is received from server, local database file is flushed and new file is created.

IV. Evaluation and Analysis

A. Suspicious Communication and Contacts

In this research, we pointed the suspicious communication if one of following occurs:

a) Andrologger’s ContactWatcher detecting newly added contacts, which were not added already in the enterprise list, could pose a threat. A flag can be set to such contacts for further investigation.

b) CallWatcher of Andrologger could detect a suspicious call from known bad individual outside the organization or external to organization’s phonebook.

c) SMS and MMS Detectors are very helpful for investigators as Andrologger can collect the body of the message. Specific text or keywords could be tracked for suspicious activity to gain further insight into an event. However there were few limitations with MMS like few a times there was no text available in MMS body and sometime the timestamp got changed.

d) Combination with Location, Gallery and Contacts could give further insight for events related to data

leakage or suspicious camera usage. E.g. A picture attached with MMS being sent to a particular contact. This event can lead to identification of contact which is secretly snooping into the enterprises confidential assets.

B. *General Usage Trends*

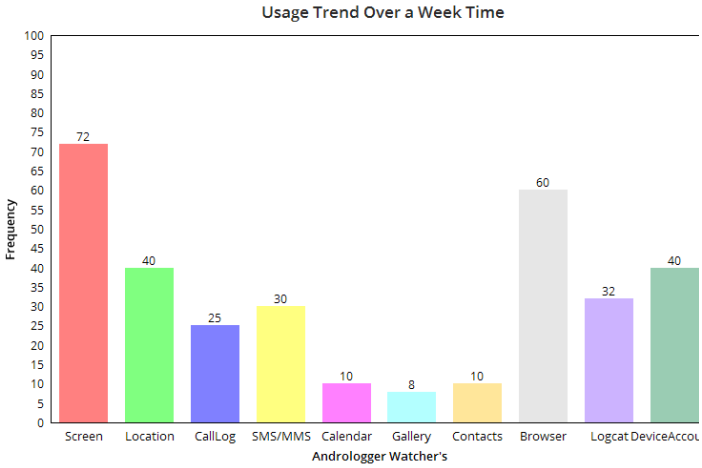

Fig8: Usage Trends of User over a Week Time

The general trend of user activity shows that user in general turns on and off the screen, sometimes without doing any activity. And the second important user behavior that we identified is the intense use of web browser for searches and visiting URL's. However, as Andrologger in each run checks for new Contacts, new Calender events and new device accounts, the activities related to the same seems to be minimal. The usage for the purpose of calling and messaging seems to be optimal.

C. *Location*

User's last known location can help in various ways to the investigators to locate the device, track the movement of the user. The alarm set to listen for GPS changes, though being set very optimally to conserve battery, sometime result in inaccurate location. It happened because sometimes user manually turned the GPS OFF or ON, resulting in data loss. As android suggests, on each reboot the last known location data is lost, resulting the device untraceable unless the location is turned ON again. The LocationWatcher of Andrologger collects latitude, longitude, deviceID and capture time from one data capture event.

RootWatcher	1484217092		1484217092331	Not Rooted
LocationWatcher	1484217112	LastKnownLocation Received	1484217112000	Lat:13.05964544; Lng:77.5534881
LocationWatcher	1484217113	LastKnownLocation Received	1484217113000	Lat:13.05964386; Lng:77.55352027
LocationWatcher	1484217114	LastKnownLocation Received	1484217114000	Lat:13.0596813; Lng:77.55347188
LocationWatcher	1484217118	LastKnownLocation Received	1484217118000	Lat:13.05968334; Lng:77.55342191

Fig9: LocationWatcher from Andrologger

In a week's usage only four significant locations could be traced, which may have happened due to not much usage of any navigation application. If any navigation app like Google Maps would have been used, data capture would have been more. Also from calendar events (a separate table in andrologger's database) tagged with location would help investigator in finding user's appointments, meetings, scheduled holidays and leaves.

D. *Browser Activities*

Andrologger tracks and records all browser related activities and being stored in the events table of the database. An investigator could detect the improper usage of internet in the organization and also can find the intention of user from the browser searches keywords. Also, if there is suspicion of intellectual property data theft, it could be tracked down to searches made to upload data to the third party websites. An event from internet history includes deviceID, timestamp, keyword or URL.

BrowserWatc...	1466411104	Browser Navi...	1466348293335	YouTube	https://www....
BrowserWatc...	1466411104	Browser Navi...	1466353904030	Online Recharge | Mobil...	https://paytm...
BrowserWatc...	1466411104	Browser Navi...	1466322477004	KAT - Kickass Torrents	https://kat.cr/
BrowserWatc...	1466411104	Browser Navi...	1466347776878	Facebook	http://www.f...

Fig10: BrowserWatcher from Andrologger Identifying Improper Usage

E. *Malicious Application*

Application installation and removal from the device are of utmost importance to the investigators. These applications could be malware, or data stealing applications. Andrologger puts a clear watch on each application installs and uninstall. AppWatcher records package name, install time, and the operation on the application as seen in below picture.

AppWatcher	1484215634	Package Removed	1484215634968	com.app.abhibus
AppWatcher	1484215635	Package Installed	1484215635184	com.app.abhibus
AppWatcher	1484215678	Package Removed	1484215678656	com.google.android.inputmethod.latin
AppWatcher	1484215678	Package Installed	1484215678939	com.google.android.inputmethod.latin

Fig11: AppWatcher indicating Application Change Status

There is also logging for all the third party application with proper filtering mechanisms in place using logcatWatcher. The filters include logs generated from *com.android*, *com.google* and *com.sec.android*. More filtering mechanisms could also be applied on the same.

V. Challenges: Anti-Forensics

As a point to note that Andrologger is susceptible to many attacks such as root detection, application uninstalls and process termination by the user itself. We could apply third party root enforcement techniques to ensure no root attacks are performed. However, still it poses the risk of evidence alteration. The data directory of the application can be accessed through android debug bridge and destroyed by the attacker or replaced with other or false evidence. Even during the logging of events an intent-filter registered with com.andrologger could get access of the logs and can be altered.

So to enforce policy level agreements from enterprise, the application should run at the system level like the com.google.android processes which runs in higher privileged mode.

VI. Conclusion and Future Scope

We tried to cover most of the useful events happening on a device and helped investigators and enterprise monitoring the organization efficiently; however there still exists space to collect additional data such as user pattern of using the device over a longer period to categorize user into threat or Suspect or Friend category. Also the application usage frequency could help in deciding that on what applications user is spending more time. This could help the investigator in identifying the intent of the users from a particular category.
Another area to look into for future scope is on how to avoid the application from becoming victim of tampering mechanisms. And in longer term the integration of such application with the OEM itself would channelize the work into the wheel.

Acknowledgment

We would like to acknowledge the contribution and support given from our organization, Bharat Electronics Ltd, Bangalore in order to carry out this work efficiently. We would also like to acknowledge our friends who equally helped us in doing the same.

References

[1] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011, August). A Study of Android Application Security. In USENIX security symposium (Vol. 2, p. 2).

[2] Vidas, Timothy, Chengye Zhang, and Nicolas Christin. "Toward a general collection methodology for Android devices." digital investigation 8 (2011): S14-S24.

[3] Hoog, Andrew. Android forensics: investigation, analysis and mobile security for Google Android. Elsevier, 2011.

[4] Sahu, Shubham. "An Analysis of WhatsApp Forensics in Android Smartphones." International Journal of Engineering Research 3.5 (2014): 349-350.

[5] Lee, Xinfang, et al. "Design and implementation of forensic system in Android smart phone." The 5th Joint Workshop on Information Security. 2009.

[6] Distefano, Alessandro, Gianluigi Me, and Francesco Pace. "Android anti-forensics through a local paradigm." digital investigation 7 (2010): S83-S94.

[7] Mahajan, Aditya, M. S. Dahiya, and H. P. Sanghvi. "Forensic analysis of instant messenger applications on android devices." arXiv preprint arXiv:1304.4915 (2013).

[8] Lessard, Jeff, and Gary Kessler. "Android Forensics: Simplifying Cell Phone Examinations." (2010).

[9] https://developer.android.com/about/dashboards/index.html, accessed January 2017.

[10] Scarfo, Antonio. "New security perspectives around BYOD." Broadband, Wireless Computing, Communication and Applications (BWCCA), 2012 Seventh International Conference on. IEEE, 2012.

[11] https://www.citrix.com/glossary/byod.html, accessed January 2017.

[12] https://www.statista.com/topics/876/android/, accessed January 2017.

[13] Rogers, Rick, et al. Android application development: Programming with the Google SDK. O'Reilly Media, Inc., 2009.

[14] Enck, William, Machigar Ongtang, and Patrick McDaniel. "Understanding android security." IEEE security & privacy 7.1 (2009): 50-57.

[15] Tiwari, Pradeep Kumar, and Upasna Singh. "Android Users Security via Permission Based Analysis." International Symposium on Security in Computing and Communication. Springer International Publishing, 2015.

[16] Wei, Fengguo, Sankardas Roy, and Xinming Ou. "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps." Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014.

[17] http://www.gartner.com/newsroom/id/2010217, accessed January 2017.

[18] https://cours.etsmtl.ca/gti619/documents/divers/CSIsurvey2010.pdf, accessed January 2017.

[19] https://github.com/pradeeptewary/andrologger/, accessed June 2017