International Journal of Latest Trends in Engineering and Technology
Special Issue April-2018, pp. 057-062
e-ISSN:2278-621X

DETECTION OF ZOMBIE APPS USING ZAPDROID

R.Suresh[1], C.Merlin Langes[2]

Abstract— Android is an open source and linux based operating system for mobile devices such as smart phones and tablet computers. Android provides a unified approach to develop applications where applications developed in android can be run in different devices powered by android. In most cases users interact with the application infrequently and some them forgot that why they installed that application. Users thought that these applications are dormant, but the application still runs in the background without the knowledge of the user which results on the loss of sensitive data and exploitation of resources. Such types of applications are marked as zombie apps. Zapdroid potentially identifies and quarantines such zombie apps and confines them to an inactive state. Zapdroid continuously monitors all the applications with optimized resource utilization and maintains the quarantined application in the sleep state unless the user intentionally awakes it. Zapdroid also provides data security and improves the performance of the device.
Keywords—Android, Zapdroid, Security, Quarantine, Restore.

1. INTRODUCTION

Android is open sourced and Linux-based Operating System for mobile devices (smart phones and tablet computers).. The snippet for Android is available under free source software licenses. Android applications are mostly developed in the Java language using the Software Development Kit. Android is used in millions of mobile devices in all countries around the world. It's the biggest installed base of mobile platform and growing fast. Growing at a rate of more than 1 million new Android devices are activated worldwide every day.

Statistics indicate that for a typical app, less than half of the people who downloaded it use it more than once ; further, 15 percent of the users never delete a single app that they download. These apps, which are seemingly considered dead by the user, continue to operate in the background long after the user has stopped interacting with them. Such background activities have significant negative impacts on the user, e.g., leaking private information or significantly taxing resources such as the battery or network. Unfortunately, the user is completely unaware of these activities. We call such apps, which are dead from the perspective of the user, but indulge in undesired activities, as “zombie apps”.

Fig 1.1 Android Core Building Blocks

In this paper, we seek to facilitate effective identification and subsequent quarantine of such zombie apps towards stopping their undesired activities. Our application continuously monitors all the applications with optimized resource utilization and maintains the quarantined application in the sleep state unless the user intentionally awakes it. Zombie apps are active in the background and hence, we need an efficient mechanism to track the continuous monitoring of apps has to be done with less resource utilization. Once the zombie app is quarantined we must ensure that it does not becomes active unless user calls it. Zapdroid must be able to restore the quarantined application quickly and with the same state.

[1] Department of Computer Science Engineering, Rajiv Gandhi Engineering college, Nemili, Sriperumbattur, India.
[2] Departmrnt of Computer Science Engineering, Rajiv Gandhi Engineering college, Nemili, Sriperumbattur, India.

Towards achieving our goal and addressing the above challenges, we design and implement Zap-Droid. Zapdroid identifies candidate zombie apps, exports appropriate information to the user to allow users to choose if he/she wants to quarantine any of them, and based on the input provided, silos a zombie app appropriately. In addition, Zapdroid also seeks to ensure that an app will execute again (in the state prior to quarantine) if the user chooses to invoke it, that the app does not crash, or that unwanted error messages do not pop up when an app is quarantined.

2. RELATED WORKS

A security framework for practical and lightweight domain isolation on Android to mitigate unauthorized data access and communication among applications of different trust levels (e.g., private and corporate).We present the design and implementation of our framework, TrustDroid, which in contrast to existing solutions enables isolation at different layers of the Android software stack:(1) at the middleware layer to prevent inter-domain application communication and data access, (2) at the kernel layer to enforce mandatory access control on the file system and on Inter-Process Communication (IPC) channels, and (3) at the network layer to mediate network traffic. For instance, (3)allows network data to be only read by a particular domain, or enables basic context-based policies such as preventing Internet access by untrusted applications while an employee is connected to the company's network. Our approach accurately addresses the demands of the business world, namely to isolate data and applications of different trust levels in a practical and lightweight way. Moreover, our solution is the first leveraging mandatory access control with TOMOYO Linux on a real Android device(Nexus One). Our evaluation demonstrates that TrustDroid only adds a negligible overhead, and in contrast to contemporary full virtualization, only minimally affects the battery's life-time.

Android’s permission system is intended to inform users about the risks of installing applications. When a user installs an application, he or she has the opportunity to review the application’s permission requests and cancel the installation if the permissions are excessive or objectionable. We examine whether the Android permission system is effective at warning users. In particular, we evaluate whether Android users pay attention to, understand, and act on permission information during installation. We performed two usability studies: an Internet survey of 308 Android users, and a laboratory study wherein we interviewed and observed 25 Android users. Study participants displayed low attention and comprehension rates: both the Internet survey and laboratory study found that 17% of participants paid attention to permissions during installation, and only 3% of Internet survey respondents could correctly answer all three permission comprehension questions. This indicates that current Android permission warnings do not help most users make correct security decisions. However, a notable minority of users demonstrated both awareness of permission warnings and reasonable rates of comprehension. We present recommendations for improving user attention and comprehension, as well as identify open challenges.

3. PROPOSED WORK

In proposed model, we seek to facilitate effective identification and subsequent quarantine of such zombie apps towards stopping their undesired activities. Explains about different scenario of attacks which can steal ourcontacts, login credentials, text messages, or maliciously subscribing the user to costly premium services. Modification is our implementation; first of all we need to design multiple applications. We also create an Anti virus like Application so as to monitor. The virus behaviors are like, Sending SMS, Storage of call logs in the Cloud server, Crashing the Gallery, Drain of Battery, Slowness of Mobile by reducing the RAM. All these misbehavior activities are monitored and quarantined successfully with the acknowledgement of the user by our Application. If any application is draining the battery level those application will shows on user mobile and it will be automatically uninstalled from user mobile and send it to recycle bin. User can install that application whenever they need.

In this paper we present a novel multi-level and behaviour based, malware detector for Android devices called MADAM(Multi-Level Anomaly Detector for Android Malware). In particular, to detect app misbehaviours, MADAM monitors the device actions, its interaction with the user and the running apps, by retrieving five groups of features at four different levels of abstraction, namely the kernel level, application-level, user-level and package-level. For some groups of features MADAM applies an anomaly based approach, for other groups it implements a signature based approach that considers behavioural patterns that we have derived from known malware misbehaviours. In fact, MADAM has been designed to detect malicious behavioural patterns extracted from several categories of malware. This multi-level behavioural analysis allows MADAM to detect misbehaviours typical of almost all malware which can be found in the wild. MADAM also has shown efficient detection capabilities as it introduces an 1.4% performance overhead and a 4% battery depletion. Finally, MADAM is usable because it both requires little-to-none user interaction and does not impact the user experience due to its efficiency.

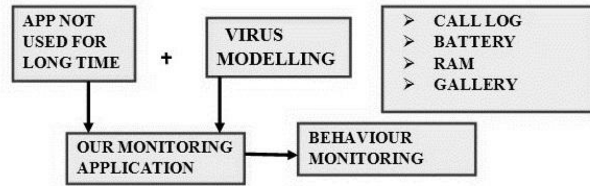


Fig 1.2 Architecture Diagram

Modules used in the application are

1. Android Deployment
2. Server
3. Modeling Virus
4. Monitoring Call Logs & Crashing Gallery

3.1 Module-Client

Mobile Client is an Android application which created and installed in the User's Android Mobile Phone. So that we can perform the activities. The Application First Page Consist of the User registration Process. We'll create the User Login Page by Button and Text Field Class in the Android. While creating the Android Application, we have to design the page by dragging the tools like Button, Text field, and Radio Button. Once we designed the page we have to write the codes for each. Once we create the full mobile application, it will generated as Android Platform Kit (APK) file. This APK file will be installed in the User's Mobile Phone an Application.

3.2. Module-Server

The Server is Server Application which is used to communicate with the Mobile Clients. The Server can communicate with their Mobile Client by GPRS or Bluetooth Technology. In the Project we are using Bluetooth technology to access with the Client. The Server Application can be created using Java/ Dot Net Programming Languages. The Server will monitor the Mobile Client's accessing information and Respond to Client's Requested Information. The Server will not allow the Unauthorized User from entering into the Network. So that we can provide the network from illegitimate user's activities. Also the Server will identify the Malicious Nodes activities.

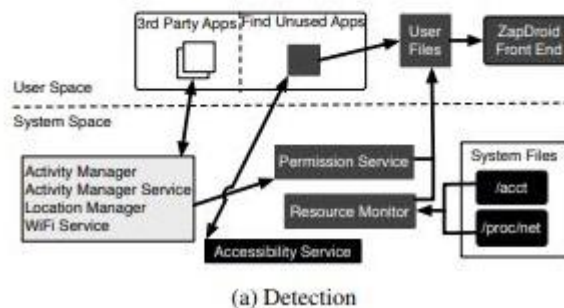


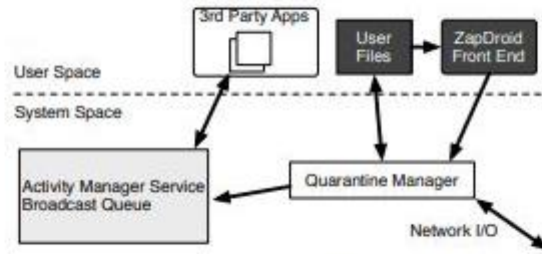
Fig 1.3

3.3. Module-Modeling virus

In this Module we will create the Mobile Virus which is malicious code that will perform malicious activities in the User's Mobile Phones. In this Project we are creating a New Folder Virus which will create a Folder inside the Folder virus by developing malicious codes. So that we can generate the Mobile Virus. Once the attackers created the Virus, they will spread the Virus via Bluetooth or SMS technique, So that the virus will be spread to other Users Mobile Phones. While sending via Bluetooth technique, the User's has to be present within the communication range. The Attacker can send the virus file via Mobile Application that was installed in their Mobile Phones.

3.4. Module-Call logs & Crashing gallery

In this module we will create the mobile virus and spread the mobiles, call logs are stored in cloud server like missed call and dialed calls. Also spread the gallery so all data's are changed in encrypted format. So does not view the gallery.



(b) Quarantine and Restoration

Fig 1.4

Measuring resource consumption: ZapDroid seeks to monitor the resources consumed by apps if they are unused by the user for a short duration (default of 2 days) to determine if they are candidates for being resource heavy zombie apps. It seeks to monitor app-level resource consumption in terms of CPU usage, network traffic, battery, and storage in a lightweight manner. The measured values should be periodically recorded and made available to the user via ZapDroid's front end. Below, we discuss how ZapDroid achieves these monitoring goals in more detail.
Monitoring network usage: ZapDroid uses the Android-3.0 Linux kernel netfilter module qtaguid to track network traffic based on the UID of the owner process (which is the app's unique identifier). The output of this module is written to the file (/proc/net/ xt qtaguid/stats). ZapDroid's Resource Manager component checks this file every 24 hours (it uses Android's Alarm Manager to set the timers) or whenever the device is rebooted (ZapDroid is notified by the Broadcast Receiver) and copies the relevant content with regards to unused apps to a user file. Specifically, it records the total numbers of bytes sent and received by each unused app in that period.
Monitoring CPU usage: To monitor CPU usage, ZapDroid leverages Android's cgroups (Control Groups). To determine the CPU ticks consumed by a particular app ZapDroid reads the file /acct/uid//cpuacct.usage. Note that there is a file per UID in this case; the single entry in each of these files is then copied to the user file discussed earlier.
Determining battery consumption: To compute the energy consumed by an unused app, the Resource Monitor uses the linear scaling model (based on CPU ticks consumed and network traffic transferred) used in Android's Fuel Gauge but at coarser time periods of 24 hours. The results are then recorded in the user file.
Determining storage: The Resource Monitor in ZapDroid calls the du command to measure the disk usage of an unused app. It essentially measures the space consumed by three folders (recursively): 1) /data/data/ / (the application's data on the flash), 2) /data/data//lib which is a soft link to the app library (the app libraries), and 3) /Android// (any data on external storage such as an SD card).
Auxiliary goals of ZapDroid: Next, we list some of the auxiliary goals that we seek to achieve when we design and implement ZapDroid.
Lightweight: ZapDroid should not consume significant resources on the user's smart phone. Otherwise, it defeats the purpose of improving user experience (which is one of the primary reasons why ZapDroid detects and quarantines zombie apps). The quarantine process should not impact device performance (e.g., offload to cloud when good WiFi connectivity is available), and the process of restoration should be quick.
Compatibility: ZapDroid should be usable on most Android devices, if not all. In other words, it must not be limited to working on a specific vendor's device(s) or on a particular version of Android.
Security: ZapDroid should not result in an escalation in privileges for any application. If such escalations are allowed, potentially new unknown vulnerabilities could arise; to avoid such cases, we set this as an implicit design requirement

3.5 Quarantining Zombie Apps

Zombie apps can belong to one of two categories depending on user input viz., "likely to restore" (Category L) or "unlikely to restore" (Category U). For Category L, quarantine involves revoking permissions from the zombie apps and preventing them from consuming resources; the apps are however, retained on the user's smartphone to ensure a quick restoration when needed. For Category U, primarily to save on storage in addition, ZapDroid moves the app binary and any associated data from the smartphone to remote storage. These functions of ZapDroid are performed by a Quarantine Manager which is implemented as an unprivileged Android system service. User Input: When a user indicates that a zombie app should be quarantined using ZapDroid's front end (therein, also indicating its category), ZapDroid's quarantine module is invoked. The user, using the front end, can also indicate the external storage to which a zombie app is to be moved if it belongs to Category U. Quarantining "likely to restore" zombie apps: The Quarantine Manager of ZapDroid essentially kills a Category L process that is executing in the background. It also prevents other apps or processes from communicating (and thus re-initializing or waking up) this app.
Killing the zombie app: ZapDroid leverages Android's Activity Manager's "am force -stop" command to kill the chosen zombie app. Keeping the zombie app in the inactive state: Inactive apps can potentially be activated by messages from deputy

apps or cloud services like GCM (Google Cloud Message service). In an extreme case, an app may itself try to overcome being force stopped by signing up for such activation messages.

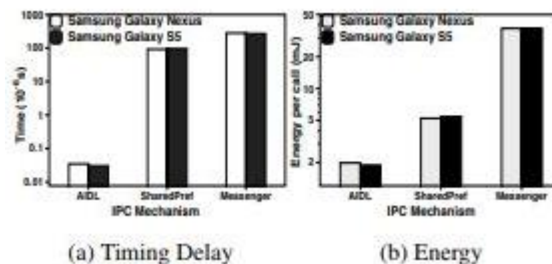
3.6 Restoring Zombie Apps

The restoration of a previously-quarantined zombie app, is also primarily handled by the Quarantine Manager. The goal here is to return the quarantined zombie app to the same state it was in (or an upgraded version if a binary is restored in the case of Category U zombie apps), prior to the quarantine. In a nutshell, the Quarantine Manager simply reverses the procedures that were invoked during quarantine. User Input: When a user seeks to restore an app, she visits ZapDroid's front end, and explicitly removes the chosen app from the list of apps to be quarantined. This automatically triggers the restoration functions within ZapDroid. Restoring "likely to restore" zombie apps: With Category L zombie apps, the Quarantine Manager, essentially invokes the call back function to inform both the hooks implemented in the BQ and in the AMS, that the zombie app to be restored must be removed from their local records. This essentially now causes the BQ and AMS to forward broadcast messages and intents respectively, to the restored app. Note that the user input, will automatically launch the app. Restoring "unlikely to restore" zombie apps: With regards to Category U zombie apps, ZapDroid checks for connectivity to both the external store and the Google Play store. If the check passes, it retrieves the URI associated with the zombie app from the user file (where it had stored the information before). The URI is then passed to the external store, to retrieve the associated objects. The binary is downloaded from the Google Play Store and installed, in parallel. After the install is completed, the user data is placed in the appropriate directories (this information was implicitly recorded during quarantine).

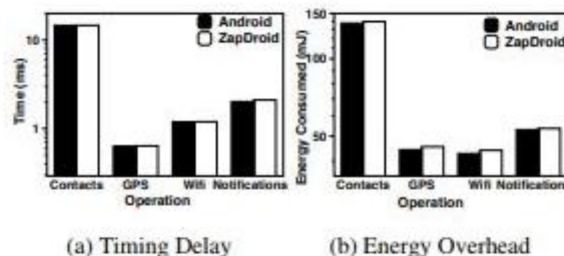
4 .RESULT DISCUSSION

4.1 Evaluation

Validating the choice of AIDL to implement IPC: The Permission Service receives inputs from components such as the WiFi Service or the Location Manager. These inputs indicate the permissions accessed by an unused app with respect to that service. IPC (message passing) is used to deliver these inputs. The IPC mechanisms have to be lightweight; they must not introduce delays in the sending process nor result in high CPU usage. We find that using Android's Binder framework, which is based on AIDL (the Android Interface Description Language) best satisfies these requirements. We compare the performance of the above approach (labelled AIDL), with the following alternative approaches: (i) using the Android's messenger framework (built on top of the Binder framework) and (ii) using shared preferences, wherein the Permission Service is notified whenever one of the aforementioned services makes a change to an underlying shared file (stored on the SD card). In Figures we depict the delays incurred in delivering the message (referred to as "timing delay"), and the energy overheads with the different approaches. We conduct these experiments on the Samsung Galaxy phones since with these



phones, we were able to explicitly remove the battery and measure the energy using a Monsoon power meter. The results demonstrate that the AIDL approach provides a three orders of magnitude advantage in terms of timing delay, and more than 60% less energy per call, compared to the alternatives. This is because, with the alternatives, there is either the overhead of using an abstraction built on top of AIDL or the overhead of using the shared media (SD card) as the medium for implementing message passing.



5. CONCLUSION

Typical smart phone users install third-party applications on their smartphones, but end up not using them in the long run. Many such apps execute in the background consuming smartphone resources (e.g., network bandwidth, energy) and/or accessing private information. We conduct an IRB-approved user study on Amazon's Mechanical Turk to identify such apps, and showcase their negative impact. As our main contribution, we design and implement ZapDroid, which detects these unused apps, and identifies those that exhibit the above undesired behaviours. These so-called zombie apps are then quarantined in order to curb these behaviours. If the user so desires, ZapDroid can later restore the quarantined app quickly and efficiently, to the same state that it was in prior to the quarantine. Evaluations on our prototype of ZapDroid demonstrate that it is lightweight, and can more efficiently thwart zombie app activity as compared to state of the art solutions that target killing undesired background processes.

6. REFERENCES

[1] "Global mobile statistics 2014 part a: Mobile subscribers; handset market share; mobile operators," http://mobiforge.com/research-analysis/global-mobile-statistics-2014-part-a-

[2] "Sophos mobile security threat reports," 2014, last Accessed: 20 November 2014. [Online]. Available: http://www.sophos.com/en-us/threat-center/mobile security-threat-report.aspx

[3] M. G. Christian Funk, "Kaspersky security bullettin 2013," December 2013. [Online]. Available: http://media.kaspersky. com/pdf/KSB 2013 EN.pdf

[4] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," EuroSec, April, 2013.

[5] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and \B. Shastry, "Towards taming privilege-escalation attacks on android,"in 19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012, 2012.

[6] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in Proceedings of the 4th International Conference on Trust and Trustworthy Computing, ser. TRUST'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 93–107. [Online]. Available: http://dl.acm.org/citation.cfm?id=2022245.2022255

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: dl.acm.org/citation.cfm?id=1924943.1924971

[8] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on android," in Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 51–62. [Online]. Available:doi.acm.org/10.1145/2046614.2046624

[9] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner,"Android permissions: user attention, comprehension, and behavior," in Symposium On Usable Privacy and Security, SOUPS '12, Washington, DC, USA - July 11 - 13, 2012, 2012, p. 3.