www.ijiarec.com

ISSN: 2348-2079

MAR-2015

International Journal of Intellectual Advancements and Research in Engineering Computations

DIFFERENT SECURITY POLICIES IN ANDROID BASED SMARTPHONES

[1]K.Rajamurugan, [2]Prof.K.Gunasekar ME.,(PhD)

ABSTRACT

Nowadays Smartphone is an effective tool for increasing the productivity of business user. With their increasing computational power, storage and capacity of smartphones allow end users to perform several tasks and always be updated while they on the move. Companies are eager to footing employee-owned smartphones because of the growing productivity of their employees. Still, security responsibilities about data distribution, leakage and loss have blocked the adoption of smartphones for corporate use. In MOSES, a policy-based framework for implementing software isolation of operations and data on the Android platform and it's possible to specify specific Security Profiles within a single smartphone. Each security profile is identified with a set of policies that control the access to operations and data. Profiles are not predefined or hard coded and they can be named and tested at any time. One of the main aspects of MOSES is the dynamic switching from one security profile to another. We run a accurate set of analysis using our full performance of MOSES. The results of the analysis certify the feasibility of our proposal.

Intex Terms-- Android security, context policy, smartphone security.

INTRODUCTION

The fast growth of smart phones has lead to be a rebirth for mobile services. Go-anywhere operations guide a wide array of social, commercial, and trade services for any user with a cellular data plan. Application markets such as Apple's App Store and Google's Android Market provide point and click access to hundreds and thousands of paid and free applications. Markets unify software purchasing, installation, and update—therein creating low limits to bring applications to market, and even lower limits for users to access and handle them.

Evaluating Android Security

Our Android application study consisted of a broad range of tests focused on three kinds of analysis: a) exploring issues uncovered in previous studies and malware advisories, b) searching for general coding security failures, and c) exploring misuse/security failures in the use of Android framework. The following discusses the process of identifying and encoding the tests.

Security Concern	All Smartphones	Android	iPhone
Having credit card information stolen	43%	46%	38%
Unknowingly having my activities monitored	43%	45%	42%
Hackers accessing my files or personal information	42%	46%	37%
Device theft/loss	41%	42%	45%
Acquiring viruses/spyware	40%	43%	37%
Downloading apps that might harm my device	38%	43%	34%
Receiving Unwanted or dangerous spam email or text messages	38%	39%	38%
Unwanted personal location tracking	37%	38%	34%

Fig1.1 Classification security concern

Phone Misuse

This section explores misuse of the

Author for Correspondence:
[1]ME-CSE-Final Year, Nandha Engineering College, Erode, Tamilnadu, India.
[2]HoD – CSE Department, Nandha Engineering College, Erode, Tamilnadu, India.

smartphone inter-faces, including telephony services, background recording of audio and video, sockets, and accessing the list of installed applications.

TELEPHONY SERVICES

Smartphone malware can provide direct compensation using phone calls or SMS messages to premium-rate numbers. We defined three queries to identify such malicious behavior: (1) a constant used for the SMS destination number; (2) creation of URI objects with a "tel:" prefix (used for phone call intent messages) and the string "900"; and (3) any URI objects with a "tel:" prefix. The analysis informs the following findings.

Finding 9 - Applications do not appear to be using fixedphone number services. We found zero applications using a constant destination number for the SMS API. Note that our analysis specification is limited to constants passed directly to the API and final variables, and there-fore may have false negatives. We found two applications creating URI objects with the "tel:" prefix and containing the string "900". One application included code to call "tel://0900-9292", which is a premium-rate number (e0.70 per minute) for travel advice in the Netherlands. However, this did not appear malicious, as the application (com.Planner9292) is designed to provide travel advice. The other application contained several hard-coded numbers with "900" in the last four digits of the number. The SMS and premium-rate analysis results are promising indicators for non-existence of malicious behavior. Future analysis should consider more premium-rate prefixes.

Finding 10 - Applications do not appear to be misusing voice services. We found 468 URI objects with the "tel:" prefix in 358 applications. We manually inspected a sample of applications to better understand phone number use. We found: (1) applications frequently include call functionality for customer service;

(2) the "CALL" and "DIAL" intent actions were used equally for the same purpose (CALL calls immediately and requires the CALL_PHONE permission, whereas DIAL has user confirmation the dialer and requires no permission); and (3) not all hard-coded telephone numbers are used to make phone calls, e.g., the Ad Mob library had a apparently unused phone number hard coded.

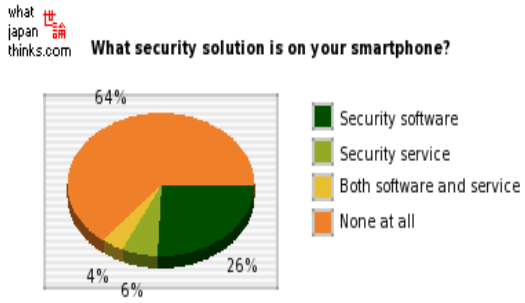

Fig1.2 Protection Chart for Smartphone

BACKGROUND AUDIO/VIDEO

Microphone and camera eavesdropping on smartphones is a real concern [41]. We analyzed application eaves-dropping behaviors, specifically:

(1) recording video without calling setPreviewDisplay() (this API is always required for still image capture);

(2) AudioRecord.read() in code not reachable from an Android activity component; and (3) MediaRecorder.start() in code not reach-able from an activity component.

Finding 11 - Applications do not appear to be misusingvideo recording. We found no applications that recordvideo without calling setPreviewDisplay(). The query reasonably did not consider the value passed to the pre-view display, and therefore may create false negatives. For example, the "preview display" might be one pixel in size. The Media Recorder.start() query detects audio recording, but it also detects video recording. This query found two applications using video in code not reachable from an activity; however the classes extended Surface-View, which is used by setPreviewDisplay().

INSTALLED APPLICATIONS

The list of installed applications provides valuable marketing data. Android has two relevant APIs types:

(1) a set of get APIs returning the list of installed applications or package names; and (2) a set of query APIs that mirrors Android's runtime intent resolution, but can be made generic. We found 54 uses of the get APIs in 45 applications, and 1015 uses of the query APIs in 361 applications. Sampling these applications, we observe:Finding 15 - Applications do not appear to be harvesting information about which applications are in-stalled on the phone.

In all but two cases,the sampled applications using the get APIs search the results for a specific application. One application defines a method that returns the list of all installed applications, but the results were only displayed to the user. The second application defines a similar method, but it only appears to be called by unused debugging code.

Our survey of the query APIs identified three calls within the Ad-Mob library duplicated in many applications. These uses queried specific functionality and thus are not likely to harvest application information. The one non Ad-Mob application we inspected queried for specific functionality, e.g., speech recognition, and thus did not appear to attempt harvesting.

Fig1.3 Applications in smartphone

CRÊPE: A SYSTEM FOR ENFORCING FINE-GRAINED CONTEXT-RELATED POLICIES ON ANDROID

Current smart phone systems grant the user to handle only somewhat contextual knowledge to indicate the behavior of the applications: this hinders the wide adoption of this technology to its full potential. In this paper, we fill this inconsistency by proposing CRêPE, an elegant Context-Related Policy Enforcement System for Android. At the same time the approach of context-related access control is not new, this is the first work that carries the concept into the Smartphone environment. In appropriate, in our task, a context can be defined by: the status of variables sensed by physical (low level) sensors, like time and spot; Increased processing on these information via software (high level) sensors; or specific interactions with the users or third parties. CRêPE grants context-related policies to be set (even at runtime) by both the user and authorized third parties locally (via an application) or casually (via SMS, MMS, Bluetooth, and QR-code). An accurate set of experiments shows that our full implementation of CRêPE has a inconsequential overhead in terms of energy consumption, time, and space, making our system ready for a production environment.

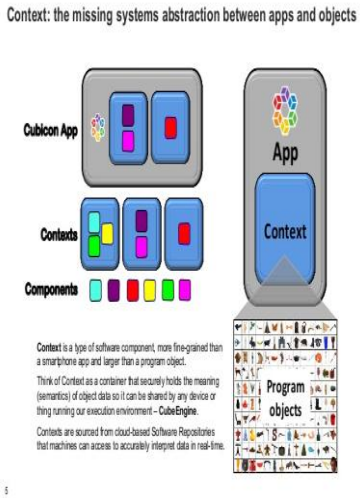

Fig1.4 Abstraction between apps and objects

MOCK DROID: TRADING PRIVACY FOR APPLICATION FUNCTIONALITY ON SMART PHONES

Mock Droid is a altered form of the Android operating system which grants a user to `mock' an application's access to a resource. This resource is finally reported as empty or unavailable at any time the application requests access. This approach grants users to remove access to particular resources at run-time, supporting users to consider the trade-off between performance and the exposure of personal data whilst they use an application.

Current applications continue to work on Mock Droid, possibly with reduced performance, since current applications are already written to accept resource failure, such as network unavailability or loss of a GPS signal. We indicate the concern with actual use of our approach by successfully running a random sample of twenty-three popular applications from the Android Market.

FLEXIBLE AND FINE-GRAINED MANDATORY ACCESS CONTROL ON ANDROID FOR DIVERSE SECURITY AND PRIVACY POLICIES

The threat of providing common security architecture for the Android OS that can deliver as a flexible and effective ecosystem to instantiate different security solutions. In contrast to previous work our security architecture, termed *Flask Droid*, provides main approach control together on both Android's middleware and kernel layers. The order of policy enforcement on these two layers is non-trivial due to their completely different definitions. We present an skillful policy language (inspired by SE Linux) tailored to the specifics of Android's middleware definitions. We show the flexibility of our architecture by policy driven instantiations of preferred security models such as the current work *Saint* as well as a modern privacy preserving, user-defined and fine-grained per-app access control model. Other possible instantiations cover phone booth mode, or dual persona phone. Finally we appraise our implementation on SE Android 4.0.4 Illustrating its ability and performance.

SECURITY ENHANCED (SE) ANDROID: BRINGING FLEXIBLE MAC TO ANDROID

The Android software stack for mobile devices represents and requires its own security model for apps through its application-layer permissions model. However, at its base, Android relies at the time of the Linux kernel to protect the system from malicious or improper apps and to separate apps from one another. At present, Android support Linux discretionary access control (DAC) to accomplish these guarantees, against the known shortcomings of DAC. In this paper, we motivate and define our work to bring flexible mandatory access control (MAC) to Android by permitting the active use of Security Enhanced Linux (SELinux) for kernel-level MAC and by growing a set of middleware MAC increases to the Android permissions model. We then establish the benefits of our security improvements for Android through a article analysis of how they reduce a number of previously published exploits and vulnerabilities for Android. Finally, we appraise the overheads imposed by our security enhancements.

DR. ANDROID AND MR. HIDE: FINE-GRAINED PERMISSIONS IN ANDROID APPLICATIONS

Google's Android platform includes a permission model that secures access to sensitive potentials, such as Internet access, GPS use, and telephony. While permissions bring an important level of security, for many applications they grant large access than actually required.

In this paper, we introduce a unique plan that addresses this problem by enumerate finer-grained permissions to Android. Basically our framework is a taxonomy of four major groups of Android permissions, each of which accepts some common techniques for deriving sub-permissions. We used these techniques to examine fine-grained versions of five of the most common Android permissions,

including entry to the Internet, user contacts, and system locations.

We then established a suite of tools that grant these fine-grained permissions to be fixed on existing apps; to be required by developers on their own apps; and to be back fitted by users on existing apps. We checkout our tools on a set of top apps from Google Play, and launch that fine grained permissions are suitable to a wide range of apps and that they can be back fitted to rise security of existing apps without changing functionality.

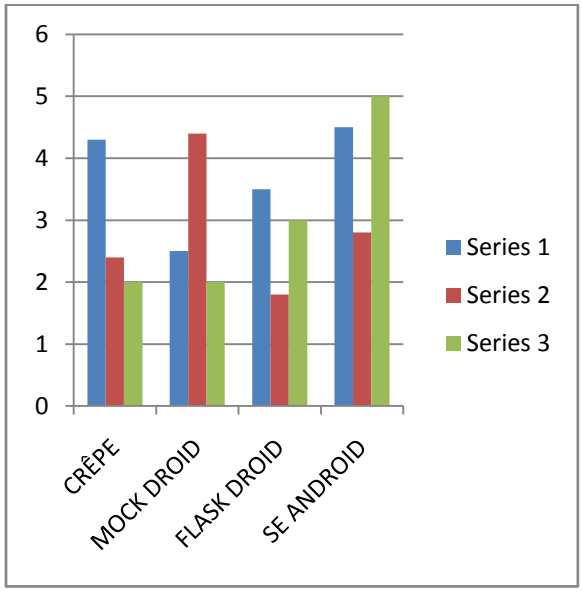

Fig1.5 Comparison chart for different security policies

REFERENCES

[1]. Y. Xu, F. Bruns, E. Gonzalez, S. Traboulsi, K. Mott, and A. Bilgic, "Performance Evaluation of Para-Virtualization on Modern Mobile Phone Platform,"Proc. Int'l Conf. Computer, Electrical, and Systems Science and Eng. (ICCESSE '10), 2010.

[2]. J. Andrus, C. Dall, A.V. Hof, O. Laadan, and J. Nieh, "Cells: A Virtual Mobile Smartphone Architecture,"Proc. 23rd ACM Symp. Operating Systems Principles (SOSP '11), pp. 173-187, 2011.

[3]. M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints,"Proc. Fifth ACM Symp. Information, Computer and Comm. Security (ASIACCS '10), pp. 328-332, 2010.

[4]. A.R. Beresford, A. Rice, and N. Skehin, "Mock Droid: Trading Privacy for Application Functionality on Smartphones,"Proc. 12th Workshop Mobile Computing Systems and Applications (Hot Mobile '11), pp. 49-54, 2011.

[5]. Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming Information Stealing Smartphone Applications (on Android)," Proc. Fourth Int'l Conf. Trust and Trustworthy Computing (TRUST '11), pp. 93-107, 2011.

[6]. P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These Aren't the Droids You're Looking for': Retroffiting Android to Protect Data from Imperious Applications," Proc. 18th ACM Conf. Computer and Comm. Security (CCS '11), pp. 639-652, 2011.

[7]. S. Smalley and R. Craig, "Security Enhanced (SE) Android: Bringing flexible MAC to Android,"Proc. 20th Ann. Network and Distributed System Security Symp. (NDSS '13), 2013.

[8]. R. Xu, H. Sa€ıdi, and R. Anderson, "Aurasium: Practical Policy Enforcement for Android Applications,"Proc. USENIX 21st USENIX Conf. Security Symp. (Security '12), p. 27, 2012.