2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications

WallDroid: Cloud Assisted Virtualized Application Specific Firewalls for the Android OS

Caner Kilinc, Todd Booth, and Karl Andersson
Pervasive and Mobile Computing Laboratory
Luleå University of Technology
SE-931 87 Skellefteå, Sweden

***Abstract*—Security is becoming an increasingly important feature of today's mobile environment where users download unknown apps and connect their smartphones to unknown networks while roaming. This paper proposes and evaluates an enhanced security model and architecture, WallDroid, enabling virtualized application specific firewalls managed by the cloud. The WallDroid solution can be considered as an Android Firewall Application but with some extra functionality. Key components used by the solution include VPN technologies like the Point to Point Tunneling Protocol (PPTP) and the Android Cloud to Device Messaging Framework (C2DM). Our solution is based on the cloud keeping track of millions of applications and their reputation (good, bad, or unknown) and comparing traffic flows of applications with a list of known malicious IP servers. We describe a prototype implementation and evaluate our solution.**

Keywords: Android OS; Security; Mobility; Cloud Computing

I. INTRODUCTION

The number of smart mobile devices has increased rapidly, due to users desire to have Internet access anywhere and at any time. Another driving force has been the steep decrease in cost, for smart model devices. There has also been a steep decrease in cost, of mobile device Internet access. Millions of users are using Android applications, on a daily basis. There have been over ten billion application downloads, from the Android market in 2010 [1]. More than 250 000 applications have been downloaded with malware [2][3]. There is a steep increase in the number of Android users who have been infected with malware. This increase in malware trend is expected to continue. This paper is an attempt to reverse this increase in malware trend.

The Android application market has not been designed to properly reject newly uploaded applications, which contain malware. Google removed 17 applications containing malware in March 2011 [4]. However these malware applications were not removed until long after the malware applications had been downloaded thousands or millions of times. So the removing of malware applications from the Android market after they are downloaded will, in general, always be too late. Another problem is that even if the Android market had been designed to reject uploaded malware applications, this is simply not possible. It is impossible to always identify a malware application, after analyzing only the application. Sometimes, the application can't be identified as malware until after it is run on users' Android devices, in a real world scenario. This paper is an attempt to allow potential malware applications to run, in a real world scenario, but in a tightly controlled environment. In this paper, the tight controls are only based on the potential malware application's IP traffic. In addition to having these tight IP controls, this paper provides a solution, where anti-malware providers can also obtain detailed IP traffic statistics, on any and all potential malware applications. This paper also addresses the following issue. There are many applications which are not malware. However, if these non-malware applications are not designed with the proper security in mind, malware applications can use these non-malware applications in improper ways, to give malware applications additional access. For example, a malware application which is not granted Internet access, can obtain Internet access via a non-malware application (which has not been implemented properly). This paper also addresses this latter issue.

Nowadays, anyone can implement Android applications without having strong programming skills. So the cost of developing Android applications is very low. Most companies and developers do not have the proper security skills, to create secure Android applications. Therefore, the developers sometimes do not consider all security issues or more often, they are simply not skilled enough to be aware of all vulnerabilities.

It is the developer who specifies which permissions the application requires. Then, when the user installs the application, the user is presented with a list of the developer's requested permissions. The user must grant all permissions, otherwise, the application will not install. The allowed permissions cannot be changed at run time. Once the application is installed, it may obtain or give other applications sensitive data. Applications can also obtain sensitive data, by interacting with the user. The sensitive data might be shared between a normal application and a malware application. Then the malware application may transmit that sensitive data via the Internet directly. Again, if the malware application does not have direct access to the Internet, if may access the Internet indirectly, via a normal application. Malware applications and even normal applications may communicate sensitive information via Internet servers or via SMS/MMS without notifying the user.

978-0-7695-4745-9/12 $26.00 © 2012 IEEE
DOI 10.1109/TrustCom.2012.298

IEEE computer society

Facebook, Twitter, and Google Apps (Calendar, Contacts, and Picasa) are a few examples, of non-malware applications which transmit private data as clear text [5], without the knowledge of most users.

Sending sensitive data without encryption over networks triggers a number of critical issues. When a smartphone establishes an Internet connection via WLAN, it is often possible to capture all traffic, including user IDs and passwords. Even if the WLAN is encrypted, with the most recent WLAN IEEE 802.11i WPA2 security, there is a serious vulnerability (Hole196). Our solution also addresses this issue.

To prevent leaking personal data and react fast, we suggest a framework, which aims to provide secure connections by normal and even malicious applications.

The rest of the paper is organized the following way: Section II surveys related work, while Section III gives further background, while Section IV presents the proposed solution. Section V describes evaluations and experiments performed, while results, conclusions and future are indicated in Section VI.

II. Related Work

There are lots of research projects going on to prevent leaking of personal data and malicious apps solutions for Android OS. One of the most commonly used approaches is a security-based permission model.

Tang et al. [6] highlights that Android Security System and treatment are too weak and proposed ASESD to prevent malware. Ongtang et al. [7] proposes the Saint framework, which grants permissions policies to overcome Android security weaknesses. Rassameeroj et al. [8] demonstrated detecting malware by distinguishing APKs' permission request from others, based on their functionality. Barrera et al. [9] overviewed iOS, Android, BlackBerry, and Symbian security frameworks and classified third-party-application installation models.

However, obviously the best and easiest solution is to prevent spreading the malicious applications from the Google Android Market rather than restricting permissions and defining new different permission levels for all applications on the phone. According to [10], the Google Android Market should be able to check security vulnerabilities and those authors even want Google to have that responsibility. Google have removed dangerous applications from their markets and even remotely from phones. Remote app uninstallation, also called a kill switch [11]. Kill switches let the vendor remotely uninstall (or deactivate) an application on a user's smartphone. Kill Switch and removing applications from market are solutions but these solutions often performed too late. Our solution is designed to take action much earlier than these solutions.

III. Background

A. Android OS

Android is a software stack (see fig. 1), which includes an operating system, middle-ware and core applications.

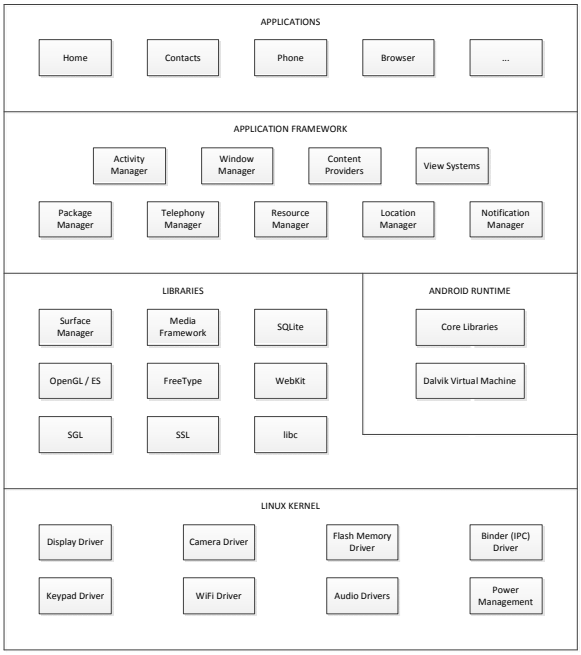

Figure 1. Overview of the Android OS

Android architecture consists of four different layers. The first layer is the Linux Kernel, the second layer is composed of Libraries and the Run Time Environment, the third layer is the Application Framework, and finally the Application layer has been placed on the top.

Android applications are developed with the java programming language. All applications must be digitally signed with a certificate. A vendor can sign their application updates with the same certificate. A vendor can also sign multiple applications with the same certificate. All applications and updates with the same certificate are considered as the same application and assigned the same locally unique User-ID. Applications with different certificates are assigned different and unique User-IDs. Each application also runs in its own Dalvik VM which is in a separate process and by default, can access only its own application files. Therefore applications with different User-IDs are isolated from each other and this structure is called a kernel-level Application Sandbox. With the default settings, just a few core applications can run with root level permissions. Each application consists of four components; Activities, Services Broadcast receives and Content Providers. All components except Content Provider provide communications between applications. Access to these communication features are allowed, based on the application's requested and granted permissions by the Intent Message Passing System [12][13][14]. Most Android built-in services have been implemented as components, for example; Phone Book and device-based functions. Inter-Process Communication (IPC) mechanisms provide interactions between these components. Therefore an installed malicious application can use built-in services and expose private data easily [15].

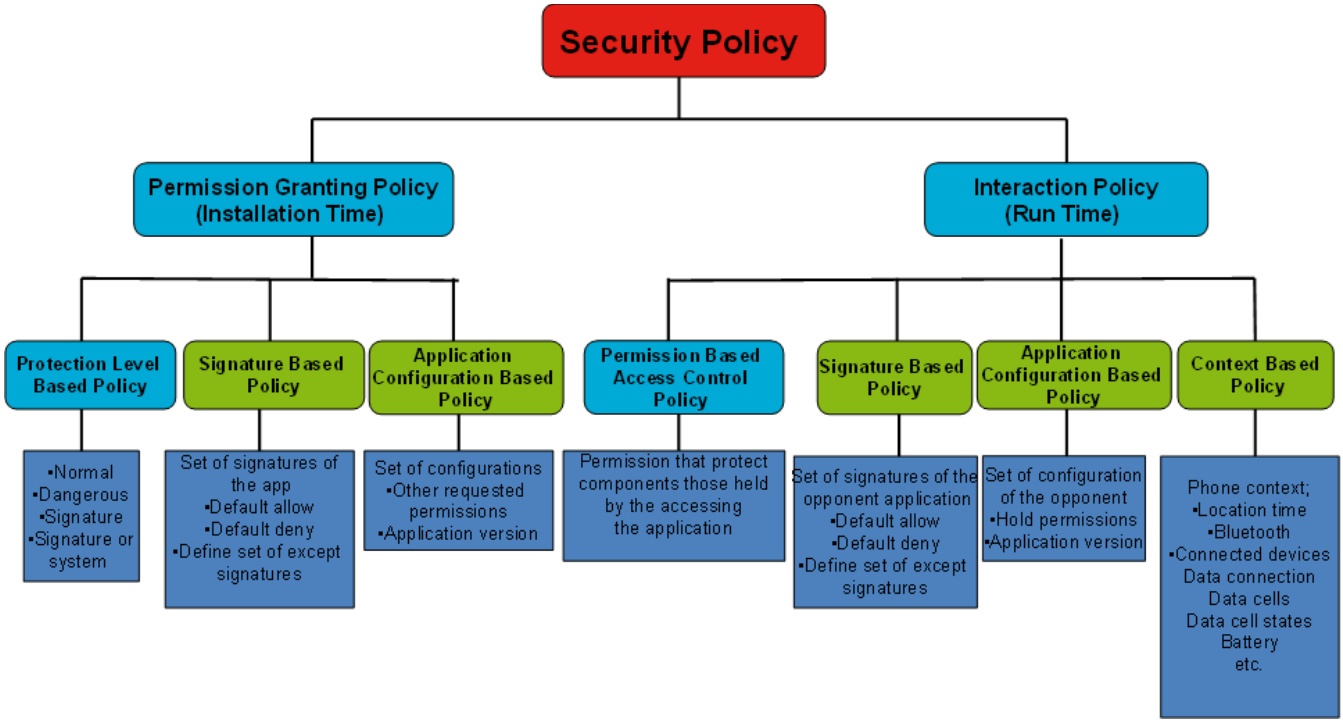

Figure 2. Andriod Security Policy Overview

The developer requests various permissions, by including tags in the application's Manifest.xml file. This file contains all critical information such as unique ID, protected parts, and access permissions. For example, if an application has the READ_PHONE_STATE and INTERNET permissions, that app can be used to get phone numbers, IMEI, user location etc. from the phone and can transmit the information to any Internet server [6]. Any application can also download and/or upload any kind of file in the background with appropriate permissions.

To protect an application from other applications, the permission label policy model is also defined in the applications manifest file.

The Android Security Policy is divided into groups; "Permission Granting Policy" and "Interaction Policy". Protection Level-based Policy, Signature–based Policy and Application Configuration-based Policy are found during installation in the "Permission Granting Policy". Interaction Policy covers four different policies as well, which includes the following: 1) Permission based Access Control Policy, 2) Signature based Policy, 3) Application Configuration based Policy, and 4) Context-based Policy, see fig. 2.

Interaction Policies are defined at runtime, for example Signature-based Policy can be used to restrict the component applications. The implementation is based on the applications' signatures, which includes default-allow and default-deny modes [12].

B. Android Cloud to Device Messaging Framework

The Android Cloud to Device Messaging Framework (C2DM) is a lightweight communication mechanism [16], which is used between application servers and their mobile Android applications, via the Google Cloud. The requirements follow: The Android OS must be at least version 2.2. The Android application must be installed via the Android market and the Android device must be logged into at least one Google account.

Pervasive and mobile computing consists of a few major challenges. One of the major challenges is battery life. C2DM is an elegant solution to reduce battery usage. Even if there are several different applications which wish to maintain communication with their own application servers, C2DM allows all applications to initially communicate over the same single shared TCP connection.

The application servers can initiate communication (push method instead of pull) with their own mobile applications. Then data can be exchanged, updated and/or queued for installation. The C2DM handles queuing of messages and delivery to its client Android mobile application.

Therefore, C2DM is the most efficient way for the application server and its client Android mobile application to communicate. This is also the most efficient way to notify the user of new application updates. It is still the user who must approve or grant the application update.

Figure 3 illustrates the C2DM service mechanism. The structure consists of three components, which are Google C2DM servers, Application Server and Android applications. The Application Server sends messages to the Google C2DM servers with an HTTP POST, whenever the server wants to communicate with its Android application. It is only the

TABLE I. EXAMPLE OF THE WALLDROID APPLICATION SERVER TABLE CONTENT

Certificate ID	Hash Result	Vendor	Application	Number of Installs	Reputation	Status
67e2…	3d17…	Skype	Skype	1,667,224	Known-Good	Direct Internet Ok
78a2…	67ac…	Unknown	Game_X	672	Unknown	See Below
78a2…	8a59…	Unknown	Game_Y	52	Unknown	See Below
8c22..	b62a…	SocialNet	The SocialNet	8,767	Known-Bad	Block Internet
729b..	c87d…	Games4You	Angry Birds	578,913	Known-Good	Direct Internet Ok

C2DM that actually communicates directly with the target Android application.

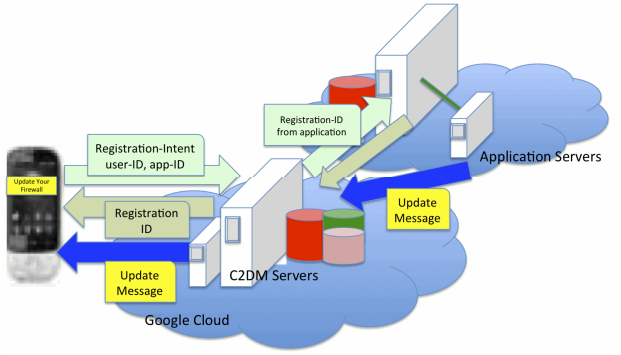

Figure 3. The Android Cloud To Device Messaging Framework (C2DM) Overview

The application must register with the C2DM servers before this C2DM service becomes active. Registration is performed by sending an Intent-register (com.google. android.c2dm.intent.REGISTER) with two parameters:

- Sender ID which is the role-based account authorized to identify an Android application before sending message. Developers set up the sender ID e.g. myApp@gmail.com
- Application ID which is used for assuring that the messages are targeted to the correct application. Application ID is defined by the package name, which is found in the manifest.xml file.

After a successful registration, the C2DM server sends a registration ID to the Android application by broadcasting a REGISTRATION Intent message. The application stores this ID for later use. Finally, the application sends the registration ID to its own application server.

The third-party application server is authorized to send messages to its Android application after it receives the registration ID. The application server issues an HTTP POST request to https://android.apis.google.com/c2dm/send in order to send messages to its client application. The server sends the following parameters: registration_id, collapse_key, data.<key>, delay_while_idle, and Authorization: GoogleLogin auth=[AUTH_TOKEN]

When a user has installed an application that supports C2DM, they will be informed by the Android Market that the application supports C2DM. The user will then be requested to grant the application, including the C2DM rights.

IV. SOLUTION ARCHITECTURE

This section presents the architecture of our application-based security model, which is called WallDroid. The aim of WallDroid is to detect malicious activity at a very early stage and then to quickly prevent any future malicious activity. The WallDroid architecture consists of three main components:

1) a VPN Server,

2) the WallDroid Application Server, and

3) a WallDroid app (running on the device).

The WallDroid app can be considered as an Android Firewall Application but with some extra functionality.

Before presenting more details concerning our solution, we will prevent various anti-malware strategies, which we believe are inferior, to our solution. Note that the following are general strategies, which are used on various clients (e.g. Microsoft, Linux, and Mac OS).

Some anti-malware solutions require the user to decide what to do, for applications which are not clearly safe and not clearly malware. However, the user is often not in the best position to make a decision. We therefore propose that the user choose a security policy. There could be a large number of different security policies, which the user could subscribe to. However, we will greatly simplify the security policy discussion and just mention a few examples. The user, for example, could choose one of the following security policies:

1) High Security

2) Medium Security

3) Low Security

One anti-malware strategy is to grant permission to all Unknown applications. Another anti-malware strategy is to deny permission to all Unknown applications. The problem with these strategies is that these are far too general.

Our solution's first component is, as mentioned, an ordinary VPN server. The second component is, also as mentioned

above, the WallDroid Application Server maintaining a table of applications, including their status and other statistics. A simplified version of the WallDroid Application Server table is available in table 1. Since vendors can use the same certificate for multiple applications and updates, we must first find a way to create our own unique application ID. Our strategy is to run the application or update through a hash function (ex: MD5 or SHA). Then our unique identifier is a combination of the certificate and hash value. The second column (Hash Result) contains the results of running the application install file through a cryptographic hash function (ex: MD5 or SHA). We are not specifying which hash function should be used. Therefore we are using very simply hash results, in order to simplify the table.

In the above, we have three classifications of Android applications.

1) "The Good" - We have applications which are known to be good. For these applications, we grant permission for these good applications to have direct Internet access.

2) "The Bad" - We have applications which are known to be malicious (bad). For these applications, we deny permission for these bad applications to have direct Internet access. We also attempt to have these uninstalled.

3) "The Unknown" - The very interesting case is for applications which are not known to be good and not known to be bad. These unknown applications are the focus of our solution.

TABLE II. EXAMPLE OF WALLDROID HANDLING OF APPLICATIONS

Application	*Tag*	*Chain*
Skype	Known-Good	Internet_Access_Direct
Game_X	Unknown	Internet_Access_Indirect
Game_Y	Unknown	Internet_Access_Indirect
TheSocialNet	Known-Bad	No_Internet_Access
Angry Birds	Known-Good	Internet_Access_Direct

The third component of our proposed solution is the WallDroid application. Part of the WallDroid application is a cloud based database service. It is this database service which contains the list of all applications and their reputations (good, bad and unknown) which WallDroid has ever encountered, on any user's Android. When WallDroid is installed, it sends the list of installed application hash values to the cloud (based on a subset of the applications' extracted files). It is then the cloud that returns the reputation of each installed application. If WallDroid detects any application-ID, which is not in the cloud's database service, it tags that application as Unknown. According to the reputation tab, WallDroid treats each application based on the given label as illustrated in the following table. Table 2 shows an example.

WallDroid allows the Known-Good App to access Internet and connect its server directly without any limitation. It blocks the Known-Bad Apps' Internet-traffic, by restricting permissions of that malicious app. When WallDroid determines an Unknown App, it automatically turns on the Android VPN service and establishes an Internet connection via a VPN server.

An according connection is established via VPN server, WallDroid System are also able to observe the Unknown app's data traffic to determine whether it is malicious app or the app is sending any personal data as a clear text. Once if WallDroid System figures out that the Unknown app is malicious or it does not care about network security, VPN server blocks the data traffic and informs the WallDroid Application server.

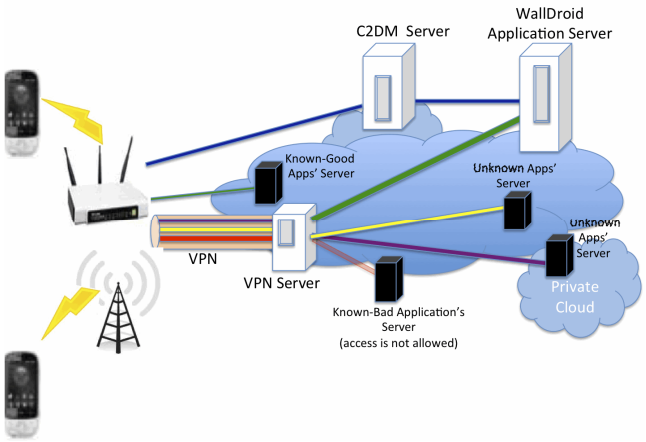

Figure 4. Solution Architecture

Application Server sends an instant message to client immediately via Google C2DM servers to notice that he/she has installed a malicious application and needs to update WallDroid as shown on figure 4.

Note that we can actually have a little more finely grained security classification. For example, let's consider the "Good". We could rate the "Good" applications, with a number between 1 and 1,000. Perhaps we would rate a "Good" application, which is known to come from Microsoft as "Good-4". Other "Good" applications, from a relatively new vendor, could be rated as "Good-728". Let's take for example, the "Good-728" reputation rating. For those applications, we could send more IP traffic statistics to our WallDroid Server for analysis. If the application is rated over 800, for example, we could also send live traffic flows, via the VPN Server for more careful analysis. If for example, there are 100,000 applications from Vendor X, with the rating of Good-729, we would only send a few of the live traffic feeds for analysis. We would not send every single application's traffic to our VPN Server.

Moreover, the reason for using a Hash Result, as the index to the table, is the following. Most of the time, perhaps more often than 99.999% of the time, the WallDroid server has seen the downloaded application before. If an Android device downloads a large application, our Android WallDroid app can upload just the Hash of the application file. The server can use the Hash Result as an index to see if the application has been uploaded to the WallDroid Application server before. Only if the application has never been seen before, will we upload the application to the server. This way, instead of uploading the application every time, we will only upload perhaps 0.0001%

percent of the time. Further, this upload can be delayed until the user has a free WLAN connection.

V. EVALUATION

We based on solution on a rooted Android 2.2 OS. The reason is the following. The market share for Android phones, prior to 2.2, is extremely small. Our solution was tested on Android 2.2. However, future Android OS releases also support our solution, which requires NetFilter.

Like all Linux distributions, the Android phone comes standard with a PPTP VPN client, which is the one we used. The WallDroid application can generate a script, which is used to automatically configure and initiate the VPN client. We used a standard PPTP server on both +Linux Ubuntu Server 11.10 and Windows 2008 R2 Server.

Also, the Android phone comes standard with NetFilter (IPTables). This enables the redirection of certain flows via the VPN Server. The Android OS is quite unique, in that each application has its own userid. We have taken advantage of that feature in the following way. What we have done, which is unique, is to use the application's unique userid. We have gathered all applications' unique IDs which are installed on our Android OS [17][18] and store the IDs in a HashMap which is called ApplicationIdMap. The map tree holds the uniqueID as keyword and other information, e.g. Tag information, as a value. When an application is requested to access Internet, we have iterated the ApplicationIdMap with the unique ID and we configure NetFilter based on the Tag of that application if the Tag is Unknown. Running a script does the configuration, so that only that apps' traffic is sent via the VPN Server to the WallDroid Application Server. By doing this we are able to capture and observe the applications' traffic at the VPN Server and able to decide whether an app is malicious or not. When we make a decision whether an app is malicious or not we inform the WallDroid Application Server using a push method. Once we have decided that it is a bad app we immediately block the traffic and the WallDroid app is updated (like a firewall) via the C2DM Server. Implementing the C2DM mechanism for an application has been described clearly by [15]. If we decided that the app is good that app can, after an update, connect to the Internet directly. On the other hand, if an installed apps' tag is Known-Good it is allowed to access Internet directly. To the best of our knowledge, no one in the industry or academia has so far come up with this approach.

WallDroid is also an efficient solution even for extreme scenarios. For example, in one of our use cases we have an application, App_X, tagged as "Known Good" which means that the app does not access any malicious server and does not transmit any personal data as clear text under normal circumstances. We also had another app tagged as "Unknown", App_Y, being a newly installed unknown app. We weren't concerned about the Android application policy at all while we were developing App_Y. As App_X could interact with App_Y, it can quickly access a Good Unknown apps' private files system and start to transmit the personal data over Internet to a server. WallDroid allows us to observe the malicious transmission and prevent any leaking of data by blocking traffic from App_X at the VPN server and also push the

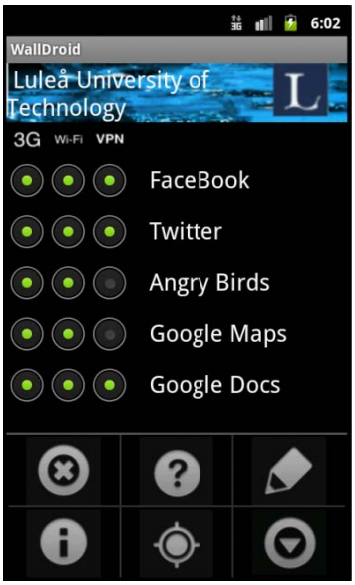

Figure 5. The User Interface of WallDroid

information immediately to the WallDroid Server about App_X being malicious. After that we can easily update WallDroid as well via Google C2DM servers.

VI. CONCLUSION AND FUTURE WORK

Existing systems are too general. We therefore provide an ApplicationID-based solution for enhancing security which is a very fast mechanism in terms of actions being taken before any data is leaked. As a result of our work, we have implemented a prototype, to demonstrate the features of our proposed solution, see fig. 5.

Our design and prototype has shown, that by taking advantage of the unique Android OS feature (unique user-id per application), that we can forward the live IP flows via a VPN, for cloud based analysis. Also, our design allows us to send just a small subset of all the same application's traffic to the cloud. Last, our design also allows us to monitor statistics, on an application-by-application basis.

Moreover, being a cloud assisted traffic observation solution WallDroid is really quick and robust against applications pretending to be Good App but actually transmits personal data over the Internet.

Our future plans are to create an industrial strength version of our solution and to perform a pilot study with a larger study group. We are also planning to build up a controlled group of Android emulators. Then it will be easy for us to quickly analyze a wide variety of applications, including known malware.

ACKNOWLEDGMENT

This work has partially been supported by the Nordic Interaction and Mobility Research Platform (NIMO) project [19] funded by the InterReg IVA North program.

REFERENCES

[1] E. Chu. 10 Billion Android Market downloads and counting, Official Google Blog. Available: http://googlemobile.blogspot.com/2011/12/10-billion-android-market-downloads-and.html. Accessed on May 13, 2012.

[2] F-Secure. New Century in Mobile Malware. Available: http://www.fsecure.com/weblog/archives/00000864.html. Accessed on May 13, 2012.

[3] R. Siciliano. Android Apps Infected with a Virus. Available: http://www.blogtalkradio.com/robert-siciliano/blog/2011/04/02/android-apps-infected-with-a-virus. Accessed on May 13, 2012.

[4] N. Olivarez-Giles. Google removes 21 apps infected with malware from its Android Market, report says. http://latimesblogs.latimes.com/technology/2011/03/google-removes-apps-android-marketplace-malware.html. Accessed on May 13, 2012.

[5] R. McGarvey. Look Out: Your Android Is Leaking. Available: http://www.esecurityplanet.com/trends/article.php/3937516/Look-Out-Your-Android-Is-Leaking.htm. Accessed on May 13, 2012.

[6] W. Tang, G. Jin, J. He, and X. Jiang. Extending Android Security Enforcement with A Security Distance Model. *Proceedings of International Conference on Internet Technology and Applications (iTAP 2011)*, Wuhan, China, August 2011

[7] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android, *Proceedings of Annual Computer Security Applications Conference (ACSAC '09)*, Honolulu, HI, USA, December 2009

[8] I. Rassameeroj and Y. Tanahashi. Various Approaches in Analyzing Android Applications with its Permission-Based Security Models. *Proceedings of 2011 IEEE International Conference on Electro/ Information Technology (EIT)*, Mankato, MN, USA, May 2011

[9] D. Barrera and P. Van Oorschot. Secure Software Installation on Smartphones, *IEEE Security & Privacy* 9(3), pp. 42-48, May-June 2011

[10] P. McDaniel and W. Enck. Not So Great Expectation, Why Application Market Haven't Failed the Security. *IEEE Security & Privacy* 8(5), pp. 76-78, October 2010

[11] G. Keizer. Google throws 'kill switch' on Android phone. Computer Worlds 7, March 2011. Avaliable http://www.computerworld.com/s/article/9213641/Google_throws_kill_switch_on_Android_phones

[12] K. H. Khan and M. N. Tahir. Android Security, A survey. So far so good. Available: http://imsciences.edu.pk/serg/2010/07/android-security-a-survey-so-far-so-good. Accessed on May 13, 2012.

[13] Google. What is Android? http://developer.android.com/guide/basics/what-is-android.html, Accessed on May 13, 2012.

[14] W. Enck, M. Ongtang, and P. McDaniel. Understanding Android Security. *IEEE Security and & Privacy* 7(1), pp. 50-57, January-February 2009

[15] E. Chin, A. Porter Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. *Proceedings of the 9th international conference on Mobile systems, applications, and services (MobiSys '11)*, Washington D.C., USA, June-July 2011

[16] Google. Android Cloud to Device Messaging Framework. Available: http://code.google.com/android/c2dm, Accessed on May 13, 2012.

[17] http://code.google.com/p/droidwall, Accessed on May 13, 2012.

[18] http://code.google.com/p/droidwall/source/browse/trunk/src/com/google code/droidwall/Api.java?r=148, Accessed on May 13, 2012.

[19] NIMO: Nordic Interaction and Mobility Research Platform. http://www.nimoproject.org, Accessed on May 13, 2012.