iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 3 **Issue:** III **Month of publication:** March 2015

**DOI:**

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

# Cloud Based Security System for Android Applications

Baburao Gadekar[#1], Vishal Patil [*2], Ajim Shaikh[#3],Manoj Baral[#4] ,Punna Rao[#5]
[#] *Department of Computer Engineering, Univercity of Pune*

***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 Smartphones; Android OS; Reputation based security; Inter Process Communication***

## 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.

Facebook, Twitter, and Google Apps (Calendar, Contacts, and Picasa) are a few examples, of non-malware applications which



©IJRASET 2015: All Rights are Reserved

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

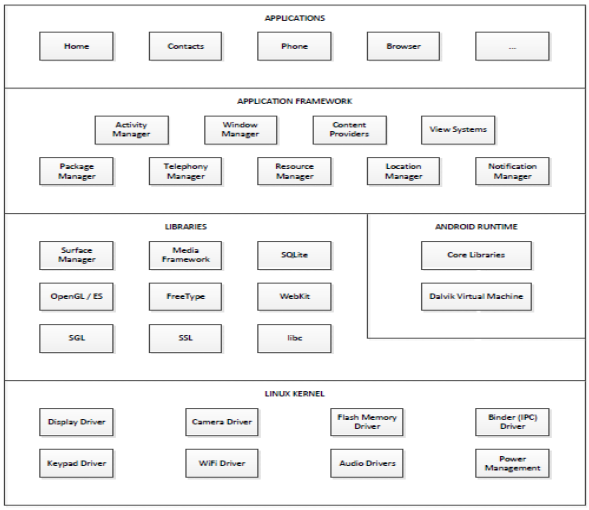



Figure 1. Overview of the Android OS

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



©IJRASET 2015: All Rights are Reserved

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].

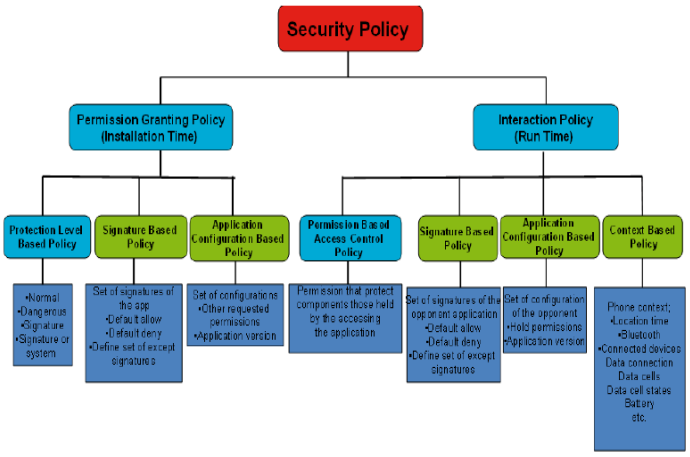



Figure 2. Andriod Security Policy Overview

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 Internetserver [6]. Any application can also download and/or upload any kind of file in th 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 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].

## 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. 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.



©IJRASET 2015: All Rights are Reserved

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 I. 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.

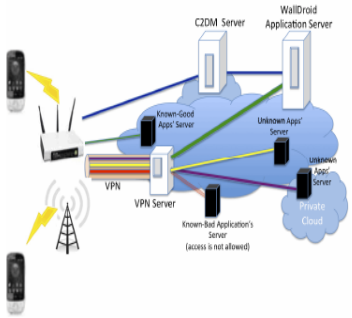



Figure3: Solution Architecture

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 rate 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.



©IJRASET 2015: All Rights are Reserved

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 the user has a free WLAN connection.

## V. EVALUATION

We based on solution on a rooted Andro reason is the following. The market share for prior to 2.2, is extremely small. Our solution Android 2.2. However, future Android OS rele our solution, which requires NetFilter. Like all Linux distributions, the Andro standard with a PPTP VPN client, which is t The WallDroid application can generate a scri to automatically configure and initiate the VPN a standard PPTP server on both +Linux Ubu and Windows 2008 R2 Server. Also, the Android phone comes standar (IPTables). This enables the redirection of cert VPN Server. The Android OS is quite uniq application has its own userid. We have taken feature in the following way. What we have unique, is to use the application's unique u gathered all applications' unique IDs which are Android OS [17][18] and store the IDs in a H called ApplicationIdMap. The map tree holds keyword and other information, e.g. Tag in value. When an application is requested to ac have iterated the ApplicationIdMap with the u configure NetFilter based on the Tag of that a Tag is Unknown. Running a script does the that only that apps' traffic is sent via the VP WallDroid Application Server. By doing this capture and observe the applications' traffic at and able to decide whether an app is malicio we make a decision whether an app is mali inform the WallDroid Application Server using Once we have decided that it is a bad app block the traffic and the WallDroid app is firewall) via the C2DM Server. Implement mechanism for an application has been desc [15]. If we decided that the app is good that update, connect to the Internet directly. On the installed apps' tag is Known-Good it is al Internet directly. To the best of our knowledg industry or academia has so far come up with t WallDroid is also an efficient solution e scenarios. For example, in one of our use ca application, App_X, tagged as "Known Goo that the app does not access any malicious ser transmit any personal data as clear text circumstances. We also had another app tagged App_Y, being a newly installed unknown a concerned about the Android application polic were developing App_Y. As App_X cou App_Y, it can quickly access a Good Unkno files system and start to transmit the personal d to a server. WallDroid allows us to observ transmission and prevent any leaking of d traffic from App_X at the VPN server and also be delayed until android 2.2 OS. The Android phones, on was tested on eases also support id phone comes the one we used. ipt, which is used N client. We used untu Server 11.10 rd with NetFilter tain flows via the que, in that each advantage of that e done, which is userid. We have e installed on our HashMap which is s the uniqueID as information, as a ccess Internet, we unique ID and we application if the configuration, so PN Server to the s we are able to t the VPN Server ous or not. When icious or not we ng a push method. we immediately updated (like a nting the C2DM cribed clearly by app can, after an e other hand, if an llowed to access ge, no one in the this approach. even for extreme ases we have an od" which means rver and does not t under normal d as "Unknown", app. We weren't cy at all while we ld interact with own apps' private data over Internet ve the malicious data by blocking o push the

Figure 4. The User Interface Of WallDroid

information immediately to the Wal being malicious. After that we can well via Google C2DM servers. VI. CONCLUSION AND Existing systems are too genera Application ID-based solution for en very fast mechanism in terms of act data is leaked. As a result of our wo prototype, to demonstrate the feature see fig. 5. Our design and prototype ha advantage of the unique Android O per application), that we can forward VPN, for cloud based analysis. Also send just a small subset of all the s the cloud. Last, our design also allow on an application-by-application bas Moreover, being a cloud as solution WallDroid is really q applications pretending to be Good personal data over the Internet. Our future plans are to create an of our solution and to



©IJRASET 2015: All Rights are Reserved

perform a pil group. We are also planning to buy Android emulators. Then it will be analyse a wide variety of appli malware. ACKNOWLEDG This work has partially been Interaction and Mobility Research [19] funded by the InterReg IVA No
ace of WallDroid lDroid Server about App_X easily update WallDroid as FUTURE WORK . We therefore provide an enhancing security which is a tions being taken before any work, we have implemented a es of our proposed solution,
as shown, that by taking OS feature (unique user-id card the live IP flows via a so, our design allows us to same application's traffic to ows us to monitor statistics, sis. assisted traffic observation quick and robust against App but actually transmits n industrial strength version ot study with a larger study Id up a controlled group of be easy for us to quickly communications, including known malware.

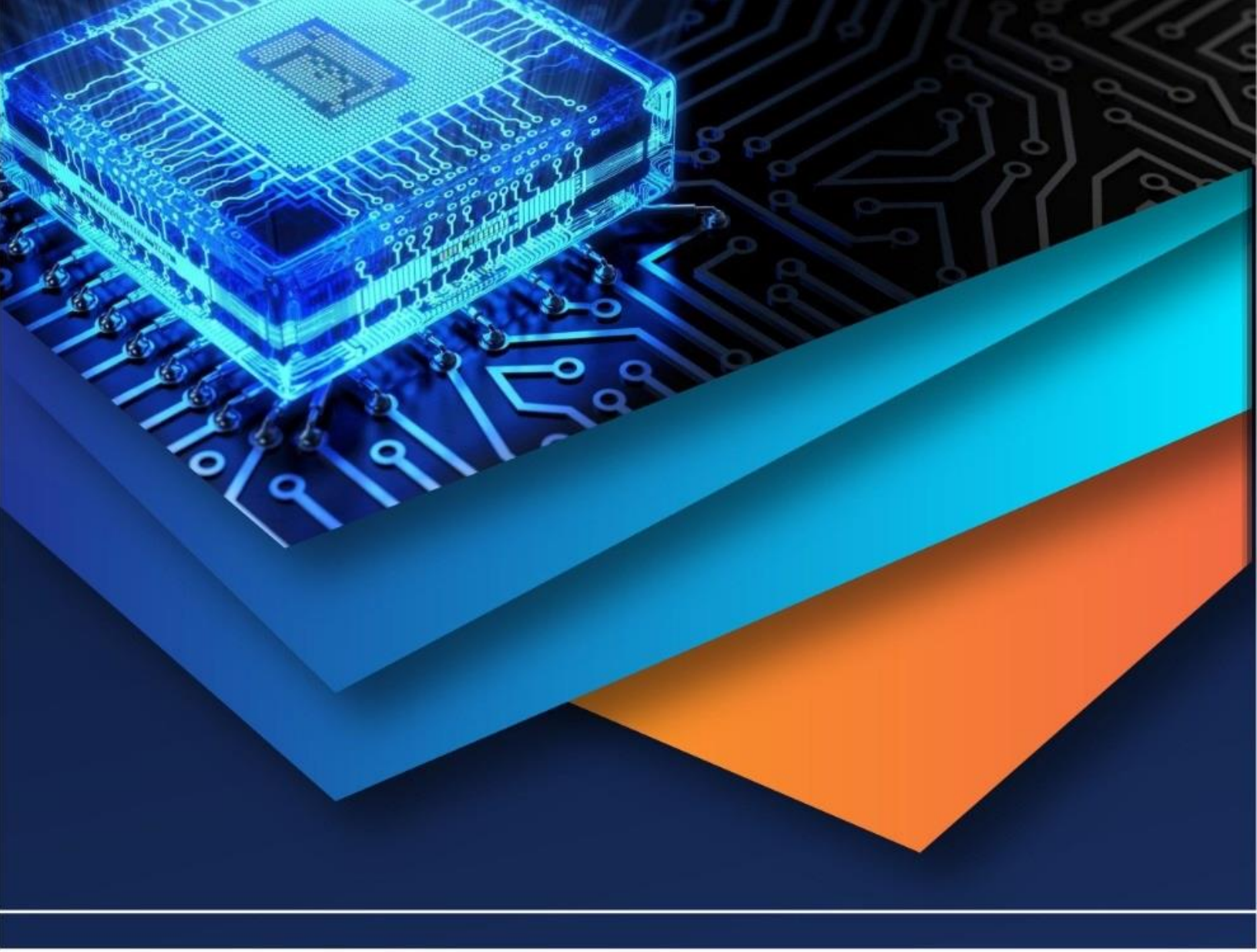
## VI. ACKNOWLEDGEMENT

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] C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber. Android Security
Permissions - Can we trust them? In Proceedings of 3rd International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MOBISEC 2011), Aalborg, Denmark, May 17–19, 2011.
[7] J. Burns. Developing Secure Mobile Applications for Android—An Introduction to Making Secure Android Applications, http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf, Accessed on May 8, 2012.
[8] E. Chin, A. Porter Feltm, K. Greenwood, and D. Wagner. Analyzing the Inter-application Communication in Android, University of California, Berkeley, Berkeley, CA, USA.
[9] T. Vidas, D. Votipka, and N. Christin. All Your Droid Are Belong To Us: A Survey of Current Android Attacks, INI/CyLab, Carnegie Mellon University.
[10] Android Market, http://www.android.com/market, Accessed on May 13, 2012.
[11]Android permissions, http://android.git.kernel.org/?p=platform/ frameworks/base.git;a=blob;f=core/res/AndroidManifest. xml. Accessed on May 13, 2012.
[12] A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-powered mobile devices using SELinux. In IEEE Security & Privacy, Volume 8, Issue 3, pp. 36–44, May–June 2010.
[13] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani Crowdroid. Behavior-Based Malware Detection System for Android. In
Proceedings of the Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'11), Chicago, IL, USA,
October 17, 2011.
[14] L. Yihe. An Information Security Model Based on Reputation and Integrality of P2P Network. In Proceedings of 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, Hubei, China, April 25–26, 2009.
[15] L. Qi. Network Security Analysis Based on Reputation Evaluation. In Proceedings of 2011 International Conference on Information Technology, Computer Engineering and Management Sciences (ICM 2011), Nanjing, China, September 24–25, 2011.
[16] http://developer.android.com/reference/android/content/Context.html #startService(android.content.Intent), Accessed on May 13, 2012.
[17] http://developer.android.com/reference/android/content/Context.html #bindService (android.content.Intent,android.content.ServiceConnection, int), Accessed on May 13, 2012.

©IJRASET 2015: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)