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

Reputation Based Security Model for Android Applications

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

Abstract—**The market for smart phones has been booming in the past few years. There are now over 400,000 applications on the Android market. Over 10 billion Android applications have been downloaded from the Android market. Due to the Android popularity, there are now a large number of malicious vendors targeting the platform. Many honest end users are being successfully hacked on a regular basis. In this work, a cloud based reputation security model has been proposed as a solution which greatly mitigates the malicious attacks targeting the Android market. Our security solution takes advantage of the fact that each application in the android platform is assigned a unique user id (UID). Our solution stores the reputation of Android applications in an anti-malware providers' cloud (AM Cloud). The experimental results witness that the proposed model could well identify the reputation index of a given application and hence its potential of being risky or not.**

Keywords-Smartphones; Android OS; Reputation based security; Inter Process Communication

I. INTRODUCTION

Android is a Linux-based open-source operating system, with a layered structure of services including core native libraries and application frameworks. Android isolates different applications and provides safety through OS primitives and environmental features such as type safety. At the application level, each software package is partially sand-boxed by the kernel, making Android a widely deployed system that employs privilege separation as a matter of course. More specifically, by default, each application is not allowed to read, update or delete another application's files. On the other hand, the Android user has the ability to allow or deny one application to communicate with other applications via inter process communication (IPC). If the user grants this permission, malware applications may be able to compromise the Android system. If the application requests this permission, and the user denies this permission, the possibly valid application will not install.

The Android Operating System (OS) provides a rich set of IPC capabilities. Unfortunately there are many security issues in the OS. Since the Android phones are limited in processing power, memory and speed, there are some additional constraints in running strong security defenses such as live antivirus and firewall programs.

Privacy issues are common in all OSes. However with Android's Global Positioning System (GPS) and Locations Based Services (LBS), hackers may be able to track the movements and location of the Android phone owner.

Microsoft Windows applications do not have their own UID. When a Microsoft Windows application runs, it uses the security impersonation feature, which allows the application to run, in the security context of the user who runs the application. When the Android application runs, they run in the security context of their own UID. Therefore, with Android, we can have more control over the security of the application. I.e., the Android applications do not, by default, have the security permissions of the phone user. However, we are missing some tools to fine tune the control the applications' security, based on these unique application UID's.

We have the following different types of application components, in the Android OS: Activities, Services, Content providers and Broadcast receivers.

A service in Android runs in the background. Other components, from various different applications can interact with the service and exchange data. When a service is installed, the developer has the option to make its service available to the applications. To do this, the installation manifest file must include a permission statement and give a name to the shared service (for example SuperService). If a user installs an application, which requires the use of SuperService, that application, must request access to SuperService via its installation manifest file.

There are three problems with the above permissions strategy:

1. The installation options are either take it or leave it. If the user wants to prevent an application, from accessing just one service, the user is must not install the application. It would be better if the user could install the application, but just prevent the application from accessing that one service.
2. The permission to use SuperService, is only checked at installation time. To make any permission changes (revoke or grant), requires the user to completely uninstall and reinstall the application, which uses SuperService.
3. It is the end user who must make the decision, if a newly installed application should be allowed to use the SuperService. Users are not in a good position to know which combinations of applications services are used by malware.

The purpose of this paper is to present a solution to identify the reputation of applications. These reputations can be stored in the AM Cloud which will mitigate all three of these Android permission security limitations. In peer to peer network environments, the trust mechanism, integrality grade of force access control can be improved by calculating the reputation grade which the application have developed [14]. Through the trust mechanism, users can get the historical experiences of target nodes, thus the network resource security and the integrity of download resources (including applications) will be ensured [15].

The remainder of the paper is structured in the following way: Section II describes related work, while Section III describes the solution we propose. Section IV presents the experiments performed, Section V indicates results, and Section VI, finally, concludes the paper and indicates future work.

II. Related Work

Shabtai et al. [1] provided a security assessment of the Android framework after identifying high-risk threats to the framework.

Bläsing et al. [2] propose an Android Application Sandbox (AASandbox) being able to perform dynamic analysis on Android programs. Also, it automatically detects suspicious applications. Dynamic analysis executes the application in a fully isolated environment, which intervenes and logs low-level interactions with the system for further analysis.

Ongtang et al. [3] pointed out that smartphone security is not yet fully developed. They considered security requirements of smartphone applications and augmented the existing Android OS with a framework to meet them. Having presented a modified infrastructure, called Secure Application INTeraction (Saint), governing install-time permission assignment and their run-time use as dictated by application provider policy, the authors provided an in-depth description of the semantics of application policy.

Shin et al. [4] specified the permission mechanism for the Android OS representing it in terms of a state machine. Security needs were identified and the authors showed that the specified system is secure over the specified states and transitions.

Teufl et al. [5] describe a method for extracting knowledge about security permissions for apps and their relations, description terms, and download counts. Based on their earlier work on Activation Patterns the paper highlights its benefits and provides a number of examples.

Orthacker et al. [6] presented a method circumventing the Android permission system by spreading the permissions over two or more apps that communicate with each other via arbitrary communication channels. Three apps were demonstrated as well as a possible detection method.

Burns [7] created a guide intended for developers of Android applications. Burns takes the developer through the security model of Android, including many of the key security mechanisms and how they can be used safely. We take the next logical step and propose a solution to some of the Android security model weaknesses.

Burguera et al. [13] suggested earlier approaches for dynamic analysis of application behavior as a means for detecting malware in the Android platform

An application can share a service and specify which permissions the consumer needs. During install, the end user can allow these consumers to access the service. The application service can further restrict access, via any authentication methods. To the best of our knowledge, there are currently no published results to further restrict permissions. Neither are there published solutions to check whether a given application is malicious or not based on its reputation record on the fly.

III. Proposed Solution

As part of a solution to the above identified pitfalls in the android security model, we propose a reputation based security trust model to evaluate and validate the applications prior to installation. We have also analyzed the consequences of a malicious application that has managed to get installed with the full consent of the end user.

The Internet is full of genuine and malicious applications. An Android mobile owner can download different applications with varying reputation ratings. In this model, it is proposed that after downloading and before installing, the mobile device asks the AM Cloud for the reputation of the downloaded application.

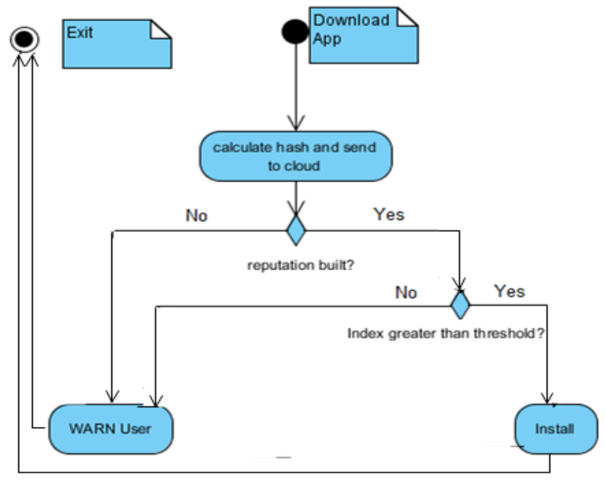

Exit
Download App
calculate hash and send to cloud
No
Yes
reputation built?
No
Yes
Index greater than threshold?
WARN User
Install

Figure 1. Schematic overview of the proposed protocol.

Based on the downloaded applications' behavior and reputation index the downloaded application can be classified in any of the following three ways.

1. The application has built a good reputation and there is likely no harm installing it on the client's device. Good reputation will be set after some threshold of positive feedback from those clients that have downloaded and automatically reported.
2. The application has not yet developed any good or bad reputation in the AM Cloud. In general, if an application has not developed a good reputation, we should be extremely cautious with such an unknown application. In this scenario, the anti-malware provider may wish to recommend that the user does not install the application or that the user installs the application in a sandbox. Let's assume that the user, Fred, installed the application anyway, in a sandbox. Our solution proposes that the anti-

malware provider keep track of which unknown applications were installed on which users Android devices. As time goes on, more information will be received from thousands of other user, who have installed the identical application. As the unknown application builds a reputation (good or bad), the anti-malware provider can inform all users, including Fred, that the unknown application has now been reclassified (as good or perhaps bad).

3. The application has a bad reputation. In this case, the user is warned about the application's bad reputation.

All Android applications and updates must include a digital signature. However vendors are allowed to use the same certificate in different applications. Also the application updates can be signed with the same certificate. Therefore a certificate does not uniquely identify any given application. Our solution requires a unique identifier for each application from a given vendor (even if they use the same certificate). Therefore our solution must create a unique application identifier. Our solution is the following. As depicted in figure 1 of the proposed protocol, we take the cryptographic hash (ex. MD5, SHA, …) of the application (APK file). Our unique application identifier is a combination of the certificate and the hash. So if a given vendor uses the same certificate for ten applications and each application has five versions, we will maintain the reputation of 50 unique applications. So our Android security application sends just the certificate and hash to the AM Cloud. We normally do not need to send the entire application (APK file) to the cloud. The AM Cloud then sends the reputation back to the Android. Then based on the reputation, the Android takes the appropriate action (as discussed above).

The reputation building protocol also considers the vendors' reputation. For example, applications from the vendor Google would have a much higher (good) reputation than applications from an unknown vendor. If a vendor releases even just one malware, that vendor' past and future applications would immediately have a much lower (bad) reputation. If an application does not indicate who the vendor is, that application would begin with a very low reputation.

A given anti-malware provider is able to keep track of the number of their users, who are running any given Android application. If a given vendor's application has under ten users, the reputation would be extremely low (unknown reputation). The reputation would increase (good reputation) as hundreds, thousands and even millions of users run a given application.

The reputation value of the application increases as more positive feedback is collected from Android phones that have installed that application.

IV. Experiments

Concerning just the applications which have not yet developed a strong reputation, we need to analyze those applications. To analyze the behavior of an Android application, it is easier to start with analyzing the set of permissions that the application has set in the Android application package file which includes all of the application's code, resources, assets, and manifest file. This can be done by some cloud based servers. The servers can then estimate the reputation based on the above analysis. This way, future client requests will receive the reputation based on the above analysis. If and when any new malware is detected, the cloud based reputation can be immediately updated (based on the application's certificate and hash value). To do this, we have experimented with a reputation based security model for Android applications. A second experiment was also done to analyze how a malicious application could track a mobile owners' location and report it to a third party.

The results were achieved using two experiments.

A. *Experiment 1*

One solution which has been used by anti-malware vendors is to perform analysis of the application, on the Android platform. However the Android is low on resources, such as performance, battery life and main memory. So it makes more sense to perform the analysis in the AM Cloud. To overcome these issues, another solution which has been used by anti-malware providers is to upload the entire application for analysis (for each user).

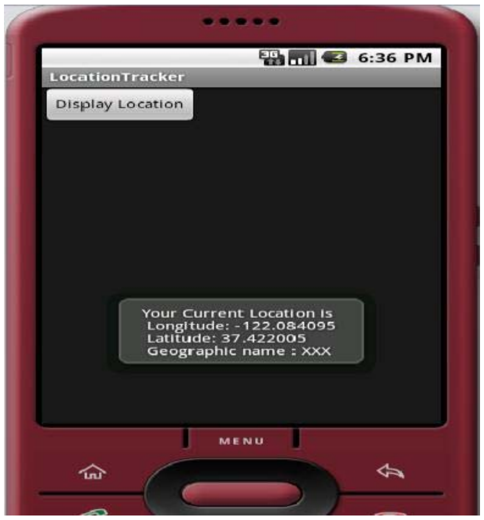
For our solution, we will minimize the uploading of applications to the AM Cloud. I.e., we do not want two users, with the same exact application, to both upload the same application. Our approach to minimize the uploading of applications now follows. Prior to installing the application, the hash value of the application will be computed. The combination of certificate and hash value will be unique (not vulnerable to forced hash collisions). The combination of certificate and unique hash value will now be referred to as the Application Identifier. So instead of uploading the entire application, only the Application Identifier will be uploaded to the AM Cloud. The AM Cloud will then perform a database look up, based on the Application Identifier. The AM Cloud will retrieve the applications reputation from the database. The AM Cloud can then send the applications reputation information to the Android user. The Android phone can then advise the user of any security issues or take automatic actions.

If the reputation based security model or another security safe guarding mechanism is breached, the malware can tweak the android platform permission model to access private values of the mobile holder, like SMS, GPS position, contacts, call history and so forth.

B. *Experiment 2*

In this second experiment, we have developed two applications namely LocationTracker, see figure 1, and ProxyMailer, see figure 4. The LocationTracker application has ACCESS_FINE_LOCATION, ACCESS_MOCK_ LOCATION, and ACCESS_COARSE_LOCATION permissions in the user permission manifest file of the application. The manifest file declares which permissions the application must have in order to access protected parts of the API and interact with other applications [18]. It also declares the permissions that others are required to have in order to interact with the application's components [18]. On the other hand the ProxyMailer is granted only the INTERNET permission.

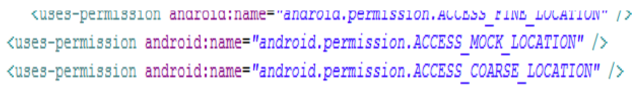
The LocationTracker application implements a location listener class that returns the latitude and longitude of the present location by consulting the LocationManager, which provides access to the system location services. We can use the latitude and longitude to locate the associated geographic place such as the street address, hotel, and zip codes.



Your Current Location is
Longitude: -122.084095
Latitude: 37.422005
Geographic name : XXX

Figure 2. Geographic location from LocationTracker App.

The user permission line of the LocationTracker application was granted the following permissions, see figure 3.



```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 3. User permission line of Location Tracker.

The GPS setting has to be enabled for the access location permissions to have access to requestLocationUpdates, which returns the present geographical location of the device by requesting for location updates from a GPS server.

The ProxyMailer application makes use of the ACTION_SEND intent of the Android platform, which is used to share contents (Email, Twitter, and Facebook). After receiving the current geographic location from the LocationTracker application, it composes a rogue email to a third party and includes the location as its email body part.

The AndroidManifest.xml file of the LocationTracker application includes a declaration of a service that will be used by the ProxyMailer client intent application to have access to the coordinates, see figure 4.

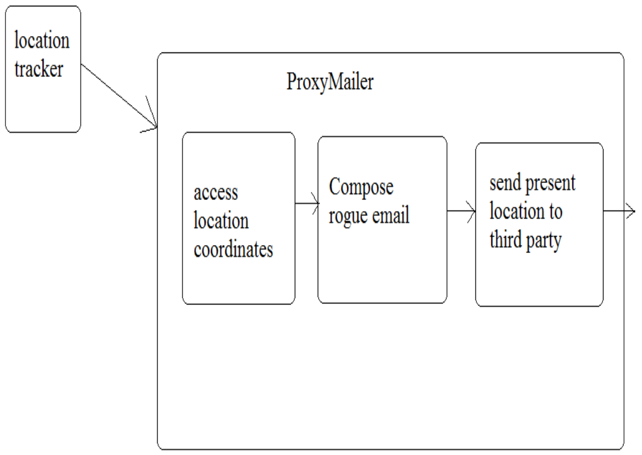

location tracker
ProxyMailer
access location coordinates
Compose rogue email
send present location to third party

Figure 4. ProxyMailer accessing location parameters and sending to third party.

The LocationTracker application has been denied of the INTERNET permission but gives Service line permission to the ProxyMailer application.

The ProxyMailer application accesses the Service of the LocationTracker application by defining Intent-an abstract description of an operation to be performed.

V. Results

The results of the first experiment set up, as depicted in figure 6 show that the device downloading a specific application can check for the reputation value pre-installation. For our example, we used a threshold for applications with reputation index greater than 20 to indicate a good and reputable application. For applications with a reputation index between 10 and 20, our policy allows the application to be installed. However, the application is sandboxed and monitored. For applications with reputation index less than 10, users should be warned to not install the application unless they know it is trustworthy. The user can even be informed as to the specific type of any malware previously associated with the untrustworthy application.

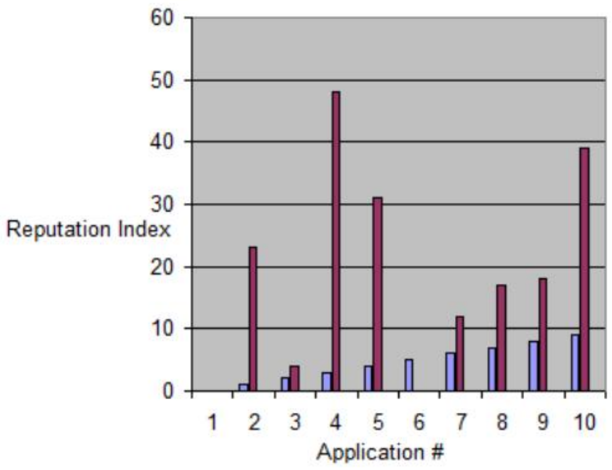
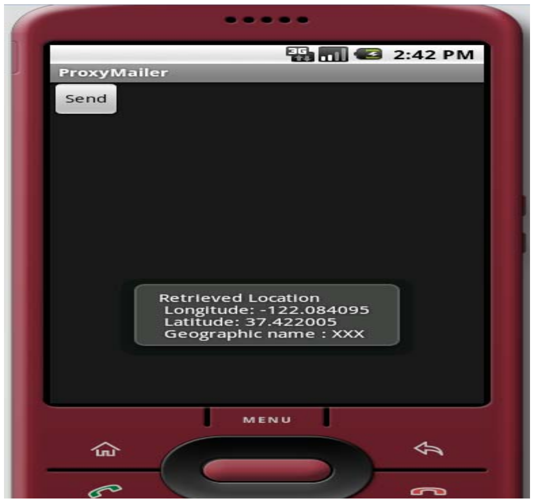


Figure 5. Results of the reputation matrix of each application.

As stated in the second experimental set up, the two applications were developed and installed separately with the consent of the user to the permissions in each application's manifest file. The ProxyMailer application, with its defined intent, has managed to access the present location coordinates from the LocationTracker application, which demonstrates that a malicious application can seamlessly access another application's services without the knowledge of the user. The ProxyMailer can then send the innocent user's current location to a third party email without the mobile holder's consent.

The service intent of the ProxyMailer defined in the application's Activity is used to move the new location data between the processes, as it is the messenger that crosses the actual system security boundaries. Intents are actually at the core of much of Android's IPC. The LocationTracker Activity will keep updating its new area coordinates whereas the ProxyMailer has to be manually run in order for the intent to have access to the client application. The Intent object is passed to the Conext.startService() to initiate the service or deliver new instructions to an the service in the LocationTracker application. startService() called over a context object is an android built-in function that enables one application to start the service of another [16]. In a similar implementation, the Intent could be passed to Context.bindService() to establish a connection between the calling Activity from the ProxyMailer and a targeted service in the LocationTracker application. The android bindService() function called over a context object can initiate the service if it isn't already running [17].



Retrieved Location
Longitude: -122.084095
Latitude: 37.422005
Geographic name : XXX

Figure 6. ProxyMailer after location retreival.

Reverse geo-coding can be used to get the actual geographical name of the new location of the mobile holder except that the experiment was done on the Android SDK emulator.

VI. CONCLUSIONS AND FUTURE WORK

Android smart phones are becoming very popular among the different smart phone platforms and this is expected to increase in popularity [20]. Despite its popularity momentum, its open source software and programmable framework behavior make it vulnerable to virus attacks [19]. In this paper, we present a reputation based security model for android applications. The paper takes into consideration the fact that Smart phones are memory, battery and speed constrained and hence exploiting the cloud to do the reputation index computation of a given application. By referring to the calculated matrix of reputation built by a given application, the model will notify users on the potential risk of the application before installation. Applications can be classified as highly risky, medium risk, less risk and genuine all based on reputation they have built in the cloud.

The experimental results show that some application need to be regarded as highly risky and therefore warn users not to install them until they improve their reputation by passing the threshold set by the reputation based security model proposed in this paper.

ACKNOWLEDGMENT

This work was partially supported by the NIMO project [21] funded by the European InterReg IVA North program.

REFERENCES

[1] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: A Comprehensive Security Assessment. In IEEE Security & Privacy, Volume 8, Issue 2, pp. 35–44, March–April 2010.

[2] T. Bläsing, L. Batyuk, A.-D. Schmidt, S.A. Camtepe, and S. Albayrak. An Android Application Sandbox system for suspicious software detection. In Proceedings of 5th International Conference on Malicious and Unwanted Software (MALWARE 2010), Nancy, France, Oct. 19–20, 2010.

[3] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android. In Proceedings of the Annual Computer Security Applications Conference (ACSAC '09), Austin, TX, USA, December 6–10, 2009.

[4] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka. Towards Formal Analysis of the Permission-Based Security Model for Android. In Proceedings of Fifth International Conference on Wireless and Mobile Communications (ICWMC '09), Cannes/La Bocca, France, August 23–29, 2009.

[5] P. Teufl, C. Orthacker, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder and O. Prevenhueber. Android Market Analysis with Activation Patterns, 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.

[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.

[18] http://developer.android.com/guide/topics/manifest/manifest-intro.html, Accessed on May 13, 2012.

[19] H. Bing. Analysis and Research of Systems Security Based on Android, In Proceedings of 2012 Fifth International Conference on Intelligent Computation Technology and Automation (ICICTA), Zhangjiajie, Hunan, January 12-14, 2012.

[20] B. Berger, M. Bunke, and K. Sohr, An Android Security Case Study with Bauhaus, In the proceedings of 2011 18th Working Conference on Reverse Engineering (WCRE), Limerick, October 17-20.

[21] NIMO: Nordic Interaction and Mobility Research Platform, http://www.nimoproject.org. Accessed on May 8, 2012.