# A Leaky Bucket called Smartphone

Osman Ugus
*Hamburg University of Applied Sciences*
*Hamburg, Germany*
*Email: ugus@haw-hamburg.de*

Dirk Westhoff
*Hamburg University of Applied Sciences*
*Hamburg, Germany*
*Email: westhoff@haw-hamburg.de*

Hariharan Rajasekaran*
*AGT Group (R&D) GmbH*
*Darmstadt, Germany*
*Email: hrajasekaran@agtgermany.com*

***Abstract*—This work is a survey presenting attacks on smartphones and recommending the best possible locations for deploying defensive mechanisms to mitigate those attacks. The attack vectors are categorized into three classes according to their characteristics as attacks via application layer, communication layer and operating system layer. We describe various attacks belonging to each of these classes and suggest locations where defensive mechanisms could be deployed to mitigate them. This paper does not intend to present a complete list of attacks. It rather tries to evaluate the best possible place for implementing the potential countermeasures to either prevent or to detect attacks with similar characteristics. The focus of this work is therefore to present a set of representative attacks on smartphone specific services and to identify meaningful locations where particular countermeasures available from the literature could be installed.**

## I. INTRODUCTION

The growing popularity of new smartphone platforms and their applications have created an excellent revenue stream for the players involved: device vendors, carriers, content providers, and application (App) developers. In addition to the commonly known security risks arising due to shorter development cycles, insecure browsers, and patchy protocol implementations, we now have thousands of third-party App developers with the ability to easily target millions of smartphone users via the application stores, by implementing insecure or malicious Apps for a device that carries sensitive personal information of its user. Failure to address security issues in this area will eventually lead to smartphone users paying the price in terms of money, identity theft, and loss of privacy.

At the moment it is easy for a malicious App developer to steal private data on the smartphone by hiding a malicious functionality behind a legitimate looking App. The usual practice of scanning for malicious Apps will not work here due to the sheer number of Apps that have yet been created for smartphones. By the time an App is discovered to be malicious the damage done will be enormous. We argue from a market-driven perspective that the time may be ripe now to integrate enhanced security solutions and offer them as a service to smartphone users. However, we are fully aware that the challenge is to judge *what level of imperfection is appropriate* [1] rather than aiming for a perfect solution.

*This work is partly done at NEC Laboratories Europe.

Security and privacy risks for smartphone users had been pushed to the background in that past when the primary objective was to aggressively promote smartphone adoption. This objective is fulfilled to a large extent today with hundreds of millions of smartphones in users' hands. However, the industry's players will have to address the security and privacy issues. As the recent CarrierIQ controversy [2] shows, smartphone users are becoming more and more sensitive to security and privacy issues affecting their smartphones. We argue that from a market strategy perspective it would now be the ideal time to further raise the end users' awareness and with this awareness in mind, prepare the ground for a new business model: Security Apps on a smartphone which connect the mobile with the cloud allowing computation intensive security for mobile devices. This article differentiates security services with respect to their potential locations. We classify security services which are required to run *locally* on the smartphone and those which shall be preferably placed in the *small cloud* or in the *big cloud*.

## II. ATTACK VECTORS IN SMARTPHONES

As depicted in Figure 1, we consider vulnerabilities at the application layer, show security weaknesses via some of the wireless interfaces a smartphone is equipped with, and finally pinpoint to attacks at the operating system as well as at the instruction level of smartphones. The list of attacks we present is surely not complete but aims to provide an overview of the various possible attack vectors with similar characteristics.

### A. Exploits via the Application Layer

*Attacks via Application Stores:* Online application stores (appstores) allow users to purchase and download third-party Apps for their smartphones. While some appstore owners enforce certain standards for the third-party Apps (e.g. Apple), others are open to anybody without any control (e.g. Android Market). As a result, the trustworthiness of the Apps vary depending on the appstore policies. Poorly implemented Apps provide attack surfaces which may compromise security of smartphones. It is very easy for an attacker to write a legitimate App and then update its functionality to steal user data based on the permissions granted to the

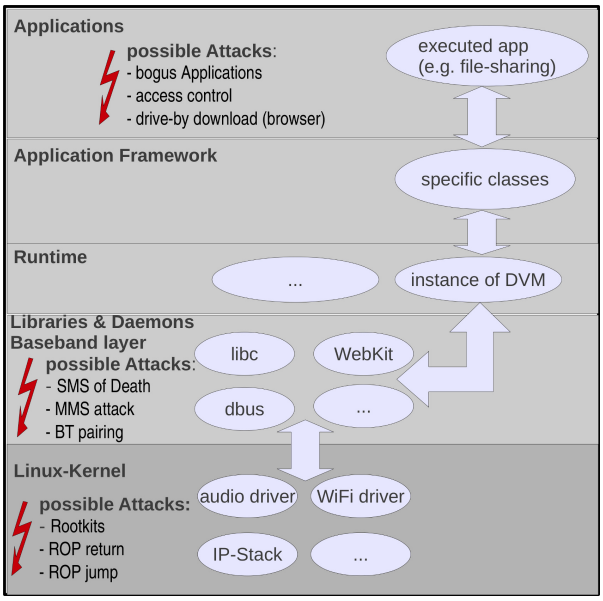


Figure 1. Overview of possible attacks on (Android) smartphones.

original App [3], [4]. Recently, it has been reported that over 50 popular free Apps are re-released on the Android market after injecting them with malware [5]. It is very difficult to stop such behavior unless someone actually studies the behavior of the App in a test environment. Even Apple's control over its appstore focuses largely on the features and business model of the App in question. Evaluation of Apps' security is not intended.

*Attacks via the Browser:* Smartphone browsers today offer the same functionality as a full-fledged browser and as a result inherit the same problems that affect the ones on the PC. Bugs in the browser code can be used to compromise the security of the smartphone. For instance, bogus JavaScript code makes use of the Android WebKit engine's parsing errors with respect to NaN-expressions such that privileged function calls can be executed by the attacker by opening a remote shell on a smartphone with a specific IP-address [6][1]. The jailbreak[2] attack on the iPhone makes use of a vulnerability of the PDF viewer exploited via the browser [7]. The attack is executed simply by visiting the web page JailbreakMe.com with iPhone's Safari browser where the malicious jailbreak code is injected into the system. A vulnerability in the Internet Explorer allowing to perform a DoS attack on the windows mobile is another example of attacks via the mobile browser [8].

*Annulling Access Control:* Smartphone platforms provide enhanced security by applying virtualization and strict access control. For instance, Android offers a virtualization environment where every App runs in its own private Dalvik virtual machine such that data created by an application is private to that App. However, the security provided by such an architectural design is easily compromised if the user "roots" or "jailbreaks" the device. In addition, sensors available on the smartphone can be used to circumvent the built-in access control features. [9] shows how a voice App can use the seemingly innocent permission to access the smartphone's microphone for stealing credit card numbers. This attack takes advantage of the notification mechanisms of the Android system to know when a user dials a credit card hotline and captures the number when the user speaks via the microphone. In some instances, the access control architecture itself is misused by naive App developers, who request more permissions than necessary for their Apps thereby increasing the chance of data leakage.

### B. Exploits via the Communication Protocols

*1) Attacks via GSM/UMTS:* We do not cope with specific security weaknesses of GSM/UMTS itself and refer to [10] for an overview. Instead, we consider a set of important attacks performed via GSM/UMTS which are representative with respect to specific services.

*SMS of Death Attack:* SMS (Short Message Service) is a communication protocol used to exchange short text messages between mobile devices like smartphones. The possibility of broadcasting text messages to a large number of mobile devices makes SMS a particularly attractive target for attackers. Attackers can easily infect a large number of smartphones with a malicious text message by using SMS-spamming techniques. In [11], the authors show how to shut down even low-end phones, which are harder to attack than smartphones, by sending small binaries to be executed via the SMS protocol. The support for the transmission of small programs is intended to be used by network operators to perform e.g., some configuration changes on the end devices remotely.

*MMS Attack to Exhaust Mobile Phone's Battery:* MMS (Multimedia Messaging Service) protocol is used to exchange multimedia contents such as picture, audio, and video files. One of the main constraints of smartphones is the battery power. Hence, smartphones change into the standby mode when no active task is performed to preserve battery life. In [12], the authors demonstrated a battery exhaustion attack by continuously forcing a mobile device into the ready state to perform a location update. The basic idea of this attack is to keep the smartphone persistently in the ready state in which the battery consumption is high. The attack is performed in two steps and not limited to a certain mobile device or hardware. Firstly, the attacker obtains the IP address of the mobile devices with active Internet connection by exploiting the vulnerability in the MMS notification messages. Subsequently, the battery of

[1]The attacker can not control who will be the victim of the attack. Any smartphone user visiting an URL with a fake HTML-page containing bogus JavaScript code will become the victim.

[2]Jailbreak allows to install Apps and themes on iPhone which are not approved by Apple.

the mobile device e.g. the smartphone is drained by sending UDP packets periodically. The battery of the victim's device was drained 20 times faster than in regular use.

*Baseband Attacks:* This class of attacks make use of the vulnerabilities found in the firmware of the baseband processor of smartphones which is used to communicate with base stations. Weinmann presented a successful attack that exploits GSM baseband stack vulnerability [13], [14]. Malicious code is inserted into the smartphone by exploiting a heap overflow vulnerability which is in turn executed on the baseband processor. Delivery of malicious code to the baseband processor requires the smartphone to be connected with a base station controlled by the attacker. At first glance, a bogus base station controlled by the attacker seems to be an unrealistic assumption. However, [14] shows that setting up a rogue base station using open source projects such as OpenBTS[3] is possible at the cost of a work station hardware. Once the attacker gets a smartphone connected to his rogue base station, he can perform the attack by inserting malicious code into the communication. Execution of malicious code on the baseband layer allows the attacker to compromise the security of data and audio communication as well as to place rootkits in case of shared memory baseband processors [14].

*Rogue WLAN Access Point:* A rogue WLAN access point is an unauthorized access point (AP) [15]. It may be a compromised AP or even provided by the attacker. By sending a high signal strength it may cover a broad region. The victim falsely assumes the bogus AP as a legitimate gateway to the Internet and connects with it. Since all traffic may be directed via the attacker's AP, it is easy to redirect the traffic or extend some malicious code to the website requested by the victim.

*2) Attacks via Bluetooth:* .

*Exploiting the Implementation:* Famous exploits are *blueJacking*, *blueSnarfing*, and *blueBugging* [16]. BlueJacking is principally an harmless attack. The attacker sends anonymously a vCard through the object exchange protocol. The name field of the vCard message is misused as a message field. Since some Bluetooth implementations permits the device to receive a vCard without a prior authentication (pairing), the attacker's message saved in the name field is displayed without the victims approval. With Bluesnarfing, the attacker may gain access to the victim's phone book or calendar or any other file whose name is guessed correctly by using the OBEX GET request. Bluebugging even allows the attacker to control the victim's phone [16]. The attacker uses a backdoor on certain devices in which an AT-parser is listening on a RFCOMM channel. Depending on the set of AT-commands available on the victim's smartphone, the attacker can access phone book data, send and receive SMSs, and listen to or make phone calls from the victim's device. Basically only a software update is necessary to protect the device from these attacks. But either because of long update cycles or because of missing indulgence from the user, devices typically keep running with the buggy firmware.

*Exploiting the Specification:* In earlier versions (2.0+EDR and earlier) of Bluetooth, the PIN may have only 4 digits or it is fixed in the device, thus limiting the security to a low degree. Since Bluetooth version 2.1+EDR the Secure Simple Pairing (SSP) has been introduced [17]. The SSP aims to prevent from passive eavesdropping as well as man-in-the-middle (MITM) attacks. As shown in [17], the latter can be circumvented. The attacker exploits the agreement of the association model prior to the pairing process, forcing the devices to use the Bluetooth specification which does not provide the MITM protection. Since the range of Bluetooth is highly limited, the attacker needs to be fairly close to his victim to perform this attack. [18] introduces a method to increase the range up to one mile by using a home-made antenna. Most attacks need the Bluetooth MAC address of the victim device which is visible only when the device is in the discoverable mode. However, Bluetooth discovery tools like *RedFang*[4] can be used to guess the MAC address of any device in range. Thus, simply being invisible is no guarantee for security.

*3) Location Privacy and Tracking:* Geoposition and mobility pattern of users are revealed intentionally or unintentionally. For instance, geo-tracking Apps for smartphones belong to the first category whereas Botnet tracking belongs to the second one. The Google's App 'Latitude' offers a smartphone user's geo-position information to other smartphone users. However,the smartphone user can specify which contacts are allowed to watch his actual geo-position. If a Botnet of smartphones can be established (e.g. by rootkits or bogus App downloads or by starting JavaScript within the WebKit browser) and a critical mass of WLAN enabled mobile devices within the same metropolitan region are affected with a 'tracking SW', it becomes possible to track the movement pattern of the users of other smartphones. This attack is possible without using infrastructures either by the intersection of GSM/UMTS cell coverage or GPS triangulation. Consequently, this attack can be launched from attackers who do not have access to a significant amount of money nor have access to telecommunication networks nor the victim's GPS module [19].

### C. Exploits at OS and Instruction Level

*Rootkit Attacks:* Rootkits are malware that hide the presence of a malicious software on an infected system [20]. Concealing the infection enables long-term malicious activities on a compromised system. Typically this requires the modification of the OS. Thus, the installation of rootkits requires privileged access. For a long time, rootkits have been considered as a challenging security problem

[3]http://openbts.sourceforge.net.

[4]http://www.securiteam.com/tools/5JP0I1FAAE.html.

for desktop class computers. However, smartphones possess similar hardware and functional capability as today's desktop computer. In fact, in addition to the Internet connection, they are equipped with interfaces like GPS that are not available on a desktop computer. Possible vulnerabilities in these interfaces are likely to provide additional backdoors for attackers to penetrate and install rootkits on smartphones. [20] has demonstrated the feasibility of rootkit attacks on a Neo Freerunner smartphone running the Openmoke Linux. Rootkits for spying conversations via GSM, for compromising the location privacy via GPS, and for performing a DoS attack via battery exhaustion were successfully implemented with a few hundred lines of code. Recently, a kernel-level rootkit for Android has been developed [21]. The rootkit runs as a loadable kernel module and allows an attacker to read SMS messages on a smartphone, makes phone calls, and pin-points a smartphone's location via GPS. The attacker obtains a privileged access on the smartphone via a shell. The attack is activated by calling the smartphone from a trigger number while the smartphone has an active Internet connection.

*Return-Oriented Programming:* Return-oriented Programming (ROP), a technique to construct malware without uploading bogus code into the target platform, was presented by Shacham [22]. Recently, ROP has been shown to be feasible for smartphones in [23]. With ROP, a new sequence of bogus instructions is generated from the code image already available on the smartphone. ROP originally is purely based on using return instructions to generate malicious behavior desired by the attacker. Recently, it has been argued that jump instructions can also be used to launch ROP attacks. For the latter, it has been shown that with indirect jumps it is possible to construct a Turing-complete gadget set with Android (ARM) platform libraries [24].

## III. COUNTERMEASURES

The level of security desired by a smartphone user depends on the group he belongs to and the type of information stored on the device, ranging from purely private data to sensitive business secrets. The multiple dimensions of the presented attack vectors indicate that most probably it is not possible to fix all leaks. This is mainly true due to the involvement of multiple players and the fact that the majority of the smartphone users does not want to pay the price for an enhanced security level by rigorously restricting the use of services, protocols, and wireless interfaces. We want to highlight meaningful locations where a particular countermeasure could be installed to presented attacks. For their placement, we differentiate between an installation on the smartphone (SP) itself, in the big Internet cloud (BC), or in the small cloud (SC) meaning other devices located within the (multi-hop) WLAN or Bluetooth transmission range or within a corporate environment. As shown in Table I, we differentiate between two types of security services: detective (d) ones and preventive (p) ones. A detective security service ideally allows to detect an attack or an infection on a smartphone and to subsequently remove it. However, it provides no mean to repair the security vulnerability which makes the attack repeatedly possible. Moreover, with respect to the attack vectors we tried to identify if a specific (s) victim can be addressed, or if victims are addressed at random (r). One could argue that if an attack vector addresses victims at random this attack is surely annoying for the victim. However, attack vectors aiming at a specific victim should be considered even more seriously since most likely the attacker is looking for dedicated information.

### A. On the Smartphone

*Mobile Security Tools (d):* There are several mobile security tools for smartphones available on the market. They offer services such as remote wipe, remote lock, phone finding function, remote backup etc. These tools provide users with a way to protect their data in case of lost devices and hence should be available on every smartphone. A remote lock of the smartphone can be defeated as shown in [25]. Depending on the provider and OS, defeating such protection mechanisms may be more or less difficult. Hence, even these tools help only to some extent. Moreover, whether these applications can offer antivirus detection facility similar to ones offered in the PC world is questionable. This is because of the virtualization architecture used in today's smartphones (e.g. in Android) which prevents such inspection functions.

One way to check if an App is malicious on Android is to disassemble the dex file of the application and check for malicious code. However, running such a service on the phone is impractical at present. Such a feature can be provided as part of a cloud based security solution where a security provider scans the Apps offline in the cloud and presents the results to the user.

Another way to prevent malicious Apps is to have a strict screening process for Apps (See Section III-B). However, not all smartphone platforms implement such a solution and some, for instance Android, by design, allow users to sideload Apps on the phone.

Finally, an alternative way to prevent malicious Apps from execution is a white-list approach combined with a trust anchor e.g. a mobile trusted module (MTM) as recently proposed in [26]. Such a harsh security policy perfectly fits to companies or other organizations where the user of a smartphone is frequently not its owner.

*App Right Management (p):* Offering a fine granular App right management may protect against attacks exploiting permissions granted to the App. Having the flexibility of selectively denying the permissions required by the App may prevent against e.g. seamlessly stealing user data stored on the smartphone. For instance, typical gaming Apps require the permission to access the networking interface of the

| Attack 'via' | Attack vector | SP | BC | SC |
|---|---|---|---|---|
| Application | via App (r) | mobile security tools (d) | App inspection (p) | co-operative antivirus detec. (d) |
| | via Browser (r) | mobile security tools (d) | App inspection (p) | - |
| | annulling ACL (r) | MVP (p), App right management (p) | App inspection (p) | - |
| protocol/service | SMS of Death (s,r) | - | SMS-filtering (p) | - |
| | MMS attack (s,r) | - | MMS-filtering (p) | - |
| | rogue WLAN (r) | WLAN policies (p) | - | AP reputation lists (p) |
| | BT attacks (s,r) | secure passw. manag. (p) | - | co-operative insecure BT detec. (d) |
| | Baseband attacks (s,r) | Baseband firewall (d) | - | - |
| OS resp. IL | Rootkits (s,r) | mobile security tools (d) | rootkit detection (d) | co-operative rootkit detec. (d) |
| | ROP return (s,r) | ROP defender (d) | - | - |

Table I
PROPOSED LOCATIONS OF PREVENTIVE (P) RESP. DETECTIVE (D) SECURITY SERVICES FOR SMARTPHONES: ON THE SMARTPHONE (SP), IN THE BIG CLOUD (BC), IN THE SMALL CLOUD (SC).

smartphone to report the score of users to a central database in the Internet. Obviously, a maliciously written App may exploit this interface to deliver confidential user data to the attacker. The possibility of removing the permissions which are not compulsorily needed to run the App would prevent this class of attacks. Hence, App right management ideally has to be considered in the design of a mobile OS as in e.g. Blackberry. Unfortunately, this is not the case for e.g., Android. An approach for implementing such a mechanism is feeding the interface, required by the permission desired to be denied, with dummy data to avoid the App crash[5]. Tools allowing fine-grained control over the permissions should be installed on the smartphone to protect against permission misuses if it is not offered by the OS.

*Virtualization and System Level Safeguards (p):* The usage of business devices for personal use is widespread today. To mitigate possible security compromises, several virtualization solutions giving the possibility of deploying multiple isolated accounts on a same device have been presented. The Mobile Virtualization Platform (MVP)[6] allows Apps to run in separated virtual environments per user role e.g., private or business. This helps in protecting e.g., business related confidential data even if an App installed in the private account is compromised. Blackberry balance[7] allows to use the same smartphone for business and personal purposes without compromising security by offering solutions which separate enterprise and private data. It also provides features for administrators which enable them to remotely delete only the enterprise data, restrict data available to third party Apps installed on the phone and provide role based access control features for connecting to the corporate network. Furthermore, [9] describes a method to install system level services which restrict third-party App's privileges to intercept data when the user performs some sensitive task. This can be further extended to give the user a "safe mode" switch that disables all system level notifications and access to sensory hardware on the phone to third-party Apps. [27] describes a more comprehensive policy framework that takes run time aspects into consideration for enforcing security.

*Baseband Firewall (d):* Baseband attacks make use of the design flaws in the baseband stack which relies on the security assumptions and the code created in the 1990s. The assumption made in 1990s that cell network elements are trusted is not fully true today. As we outlined in Section II-B, one may set up his own base station using only open source software and cheap hardware. This was obviously not the case in 1990s. Considering this fact, baseband attacks are not a theoretical curiosity anymore, but an important threat to smartphone security which we will likely face in the near future. Baseband attacks execute malicious code directly on the baseband processor. Hence, they are difficult to detect. If the attack is performed properly with the right parameters as expected by the protocol exploited (e.g. GSM), it even leaves no trace on the system [14]. The detection mechanisms should therefore be based not only on forensic techniques recognizing an attack from traces left on the system. As currently discussed in the research community, heuristic techniques based on side channel information like power analysis may be applied to detect baseband attacks. The basic idea is to recognize a malicious behavior via e.g. an increase in the power usage compared to the standard operating conditions. Moreover, control flow integrity (CFI) enforcement during the runtime on the baseband processor can be a possible direction to investigate. The challenges specific to the architecture of the baseband processor in implementing CFI enforcement is the major issue that needs to be solved.

*ROP Detection (d):* ROP attacks based on return instructions can be detected on Windows and Linux systems via ROPDefender recently introduced in [28]. However, in its current version ROPDefender has a rather poor performance since the shadow stack as the core of the detection mechanism at runtime is running in user space. G-Free, a compiler-based approach detecting all possible forms of ROP attacks, has been proposed in [29]. G-Free defeats ROP attacks by prepending a protection code to the free-branch instructions to prevent the construction of gadgets. However, a limited number of gadgets can still be constructed by

[5]For example, WhisperCore's selective permissions tool for Android uses this principle http://whispersys.com/permissions.html.

[6]For example, VMware's MVP: http://www.vmware.com/products/mobile/ or L4Android project: http://l4android.org/.

[7]http://us.blackberry.com/apps-software/business/server/full/balance.jsp.

skipping the alignment sleds within the protection code [29]. Smartphones compared to conventional computer systems are limited in terms of hardware capabilities. Therefore, tools like ROPDefender, G-Free, and other possible defenses[8] still need to be optimized for resource constrained platforms before being deployed on smartphones for detecting ROP attacks.

### B. In the Big Internet Cloud

There are good reasons for shifting some security services from the device to the big cloud considering security as a cloud service: i) parallel processing allows running a powerful (D)IDS: E.g. one could use multiple engines on the same dataset in parallel like ClamAV, Symantec, McAfee, BitDefender, F-Secure, ii) only a small proxy on the smartphone is needed which ensures connecting the smartphone to the cloud DIDS security service, and, iii) in case of an update of the DIDS no update on the smartphone is required. This allows timely reaction to actual exploits in the cloud which would not be the case if an update of the smartphone would be needed. Generally, the big cloud approach is most interesting when a large amount of data (malware) needs to be analyzed and compared with signed data. Moreover, this approach is a promising scenario for a third-party security provider. The following preventive and detective security services could be delegated to the big Internet cloud[9].

*App Inspection and CFI Enforcement (p):* A security architecture for off-loading the heavy computation needed to detect malware has been proposed in [30]. An exact replica of the software running on the smartphone is hosted in a virtual environment in the big cloud. Malware detection is performed in the virtual replica of the smartphone using the records received from the smartphone. As security checks are not performed locally on the smartphone, this technique allows to apply multiple attack detection mechanisms independent from the constraints of smartphones. An another envisioned preventive smartphone security service in the big cloud is the security inspection of App code, e.g. Android Dalvik code. Android Apps are written in bytecode for the Dalvik VM. Potential and available tools for the inspection of Apps written in Dalvik code could be `ddx`, `dex2jar`, `jad`, and the java decompiler. So, as a consequence, one could promote dedicated appstores offering classified Apps: those which have passed the App inspection service (by a certified third party or cellphone vendors) and those which have not passed and eventually are bogus. This could provide preventive security against malware attacks via the appstore and attacks in which the App developer requests the user to provide excessive permissions. Moreover, to mitigate runtime attacks on Apps exploiting e.g. buffer overflow vulnerabilities, the App code could be extended with control flow integrity (CFI) enforcement by instrumentation compatible with the smartphone architecture. Such a framework for smartphones has been recently proposed in [31].

*Rootkit Detection (d):* Depending on the detectability by user-space security tools, rootkits may be classified in two groups [20]. Rootkits detectable by user-space security tools are typically implemented as kernel modules. Infected kernel modules are loaded every time the operating system is started. Hence, mobile security tools can be exploited to detect this class of rootkits leaving a disk footprint on the smartphone by using e.g. signatures. Rootkits non-detectable by user-space security tools are more complex and implemented by modifying system utilities e.g., the functions used for checking the integrity of the system. User-space detection tools exploiting infected functions therefore cannot be trusted for the detection of rootkits. Even more sophisticated rootkits leave no disk footprint by installing themselves in the kernel memory and are not detectable by user-space detection tools. Detection of this class of rootkits requires usually a trusted platform module (TPM) or a separate virtual machine running on the same platform. However, the resource limitation of smartphones are still the main challenges to tackle in implementing such detection mechanisms. We envision that TPM based rootkit detection mechanisms can be developed in a combination with an integrity verification protocol residing in the big cloud.

### C. In the Small Cloud

*Co-operative Antivirus/Rootkit Detection (d):* The idea of running several antivirus engines in parallel has been exploited by the VirusTotal project[10]. Similarly, a co-operative antivirus detection in the small cloud can be achieved by starting various mobile security tools (see Section III-A) installed on different smartphones on the same set of data. This can provide a higher success ratio for detecting malware since different antivirus software may have implemented different detection algorithms. In case rootkits are not too sophisticated (i.e., rootkits detectable by security tools based on e.g., signatures) this may also be an option.

*Reputation System for WLAN Access Points (p):* In a region with multiple hot-spots there could be a black-listing respectively white-listing of available access points. Smartphone users or other WLAN users rate the trustworthiness of WLAN access points as preventive security service. Clearly, requesting the reputation service can only be done via Bluetooth (neighbors) or GSM/UMTS.

*Bluetooth security (d):* The recommended way to protect devices against Bluetooth attacks is to simply turn off the Bluetooth radio whenever not needed. The passkeys used to establish a secure communication between devices being paired should be at least 8-digits long and should be unique for each pairing [16]. Using a secure password management

[8]We refer to [29] for an overview of the defenses against ROP attacks.

[9]Countermeasures such as SMS and MMS filtering are not discussed further.

[10]http://www.virustotal.com.

tool on the smartphone for generating good passkeys and for managing them is therefore recommended. Bluetooth security requires not only the use of newest Bluetooth versions, but also users with high security awareness. However, this is not always possible, as many smartphones in use today still have the insecure version of Bluetooth devices, and average users are in general not aware of security threats. Other techniques running in the small cloud may be useful in tackling this problem. For example, smartphones in the small cloud can be used to warn the user of a smartphone using a Bluetooth device that is configured insecurely. This can be achieved e.g., by designing a tool which periodically scans the neighborhood for certain weaknesses unique to an insecure Bluetooth device. The acceptance of such a tool from all smartphone users is of course questionable. Moreover, the practical implementation of the mechanism to be used to warn insecure devices is also an open question.

## IV. Conclusions and Open Issues

The work at hand presents an overview of attack vectors against today's smartphones. It was not our intention to present a complete list of attacks. We rather wanted to present attack vectors from different angles. However, we believe that this work could stimulate the discussion *where* to locate potential countermeasures to either prevent or to detect attacks of the aforementioned categories. More elaborate analysis and discussion required on the proposed solutions is out of the scope of this work and left as a future work.

## Acknowledgment

The work was supported by the German BMB+F SKIMS project. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the SKIMS project, or the BMB+F.

## References

[1] J. F. Brenner, "Why isn't cyberspace more secure?" *Communications of the ACM*, vol. 53, pp. 33–35, Nov. 2010.

[2] "Carrier IQ faces federal probe into allegations software tracks cellphone data," Washington Post, Article, Dec. 2011.

[3] "Android wallpaper app beams users data to Chinese servers," geek.com, Blog News, Jul. 2010.

[4] "Researchers show infecting smartphones with malware is relatively easy," h-online.com, Blog News, Mar. 2010.

[5] "Stolen apps that root Android, steal data and open backdoors available for download from Google Market," androidpolice.com, Blog News, Mar. 2011.

[6] "Backdoor exploit for Android phones," h-online.com, Blog News, Nov. 2010.

[7] "iPhone jailbroken by Safari vulnerability again - Update," h-online.com, Blog News, Aug. 2010.

[8] "Denial Of Service in Internet Explorer for MS Windows Mobile 5.0," National Vulnerability Database (NVD), Vulnerability CVE-2007-0878, Feb. 2007.

[9] R. Schlegel *et al.*, "Soundminer: A Stealthy and Context-Aware Sound Trojan for Smartphones," in *the 18th NDSS'11*, San Diego, California, USA, Feb. 2011.

[10] M. Becher *et al.*, "Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices," in *the IEEE SP'11*, Oakland, California, USA, May 2011.

[11] ""SMS of Death" Could Crash Many Mobile Phones," technologyreview.com, Blog News, Jan. 2011.

[12] R. Racic *et al.*, "Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phones' Battery," in *the SecureComm'06*, Baltimore, MD, USA, August 2006.

[13] R.-P. Weinmann, "Coming soon: A new way to hack into smartphones," infoworld.com, Blog News, Jan. 2011.

[14] R. P. Weinmann, "All Your Baseband Are Belong To Us (Presentation)," Jan. 2011, http://youtu.be/CPPQ8vA6cRc.

[15] L. Ma *et al.*, "A Hybrid Rogue Access Point Protection Framework for Commodity Wi-Fi Networks," in *the 27th IEEE INFOCOM'08*, Phoenix, AZ, USA, Apr. 2008.

[16] K. Scarfone and J. Padgette, "Guide to Bluetooth Security," NIST Special Publication 800-121, NIST, Sept. 2008.

[17] K. Hypponen and K. Haataja, ""Nino" Man-in-the-Middle Attack on Bluetooth Secure Simple Pairing," in *the 3rd IEEE/IFIP ICI 2007*, Tashkent, Uzbekistan, Sept. 2007.

[18] "How To: Building a BlueSniper Rifle - Part 1," tomsguide.com, How to Article, Mar. 2005.

[19] N. Husted and S. Myers, "Mobile Location Tracking in Metro Areas: Malnets and Others," in *the 17th ACM CCS'10*, Chicago, Illinois, USA, Oct. 2010.

[20] J. Bickford *et al.*, "Rootkits on Smart Phones: Attacks, Implications and Opportunities," in *the 11th ACM HotMobile'10*, Annapolis, Maryland, USA, Feb. 2010.

[21] "Android rootkit demonstrated," h-online.com, Blog News, Aug. 2010.

[22] S. Hovav, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," in *the 14th ACM CCS '07*, Alexandria, Virginia, USA, Oct. 2007.

[23] T. Kornau, "Return Oriented Programming for the ARM Architecture," Diploma Thesis, RUB, Dec. 2009.

[24] S. Checkoway *et al.*, "Return-Oriented Programming without Returns," in *the 17th ACM CCS '10*, Chicago, Illinois, USA, Oct. 2010.

[25] "Ubuntu Lucid Lynx 10.04 can read your iPhone's secrets," zdnet.com, Blog News, May 2010.

[26] O. Ugus and D. Westhoff, "An MTM based Watchdog for Malware Famishment in Smartphones," in *the 11th I2CS*, Berlin, Germany, June 2011.

[27] M. Ongtang *et al.*, "Semantically Rich Application-Centric Security in Android," in *the ACM ACSAC'09*, Orlando, Florida, USA, Dec. 2009.

[28] L. Davi *et al.*, "ROPdefender: A Detection Tool to Defend Against Return-Oriented Programming Attacks," Technical Report HGI-TR-2010-001, Ruhr-Universität Bochum, Dec. 2010.

[29] K. Onarlioglu *et al.*, "G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries," in *the 26th ACM ACSAC'12*, Austin, Texas, USA, Dec. 2010.

[30] G. Portokalidis *et al.*, "Paranoid Android: versatile protection for smartphones," in *the 26th ACM ACSAC'10*, New York, NY, USA, Dec. 2010.

[31] L. Davi *et al.*, "MoCFI: A Framework to Mitigate Control-Flow Attacks on Smartphones," in *the 19th ANDSS'12*, Feb. 2012.