On Detection of Privilege Escalation Attacks in Android Smartphones.

By

Bruno Ssekiwere

SEP15/COMP/0643U

Option-Mobile Computing

Supervisor

John Ngubiri (PhD)

....

UTAMU

A Proposal Submitted to the Gradute School for a Dissertation in Partial Fulfilment of the Requirement for the Award of MSc Computing of Uganda Technology and Management University.

February, 2017

1 INTRODUCTION

1.1 Background

Mobile phones have become a key component of our daily lives as one of the dominant means of communication. They have increasingly become popular and adept at handling a variety of roles ranging from web-browsing and emailing, to multimedia and entertainment applications (games, videos, audios), navigation, trading stocks, and electronic purchases.

Top mobile operating systems in 2016 were Android (65.87 percent), iOS (29.52 percent), Windows Phone (1.76 percent), Java ME (1.44 percent), Symbian (0.85 percent), Samsung (0.01 percent) and BlackBerry OS (0.54 percent) according Net Market Shares: Mobile/Tablet Operating System Market Share figures. Other mobile operating systems account only for 0.02 percent market share. However, the popularity of android smartphones and the vast number of the corresponding applications makes these platforms also more attractive targets to attackers.

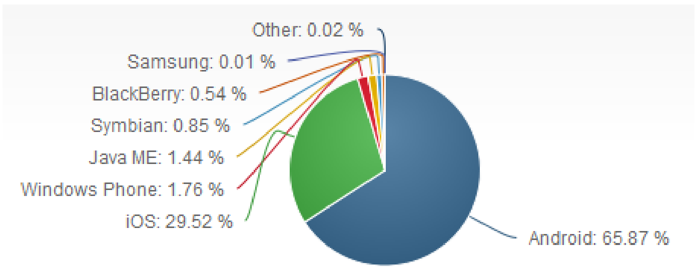

Fig. 1. SmartPhones Operating Systems Market Share

Currently, various forms of malware exist for smartphone platforms, including Android. Advanced attack techniques, such as code injection, return-oriented programming (ROP) and ROP without returns affect applications and system components at runtime. Against malware and runtime attacks, well-established security features of today's smartphones are application sandboxing (isolates applications from each other and from system resources) and privileged access to advanced functionality (Davi, 2010) to control access to system resources and mediate application communication (Bugiel, 2012).

Resources of sandboxed applications are isolated from each other, and each application can be assigned a bounded set of permissions allowing an application to use protected functionality (Dmitrienko, 2015). Sandboxing is enforced at the kernel level of Android, while permission framework is implemented as a reference monitor at the middleware layer to control access to system resources and mediate inter-application communication. As a result, malicious or compromised applications are only able to perform actions which are explicitly allowed by permissions of their sandboxes (Davi, 2010) and since most smart phones rely entirely on application sandboxing and privileged access for security, applications are isolated and granted privileged permissions only (Mathew, 2012).

Android is a privilege separated system, with each application having its own distinct system identity. This security model is not able to detect transitive usage of permissions that can be leveraged to launch privilege escalation attacks (Rangwala, 2014). It is very much possible for a malicious application to gain privileges leaked from benign applications making its capabilities more than it is permitted to have (Rangwala, 2014).

The current mobile applications business and usage model allows developers to upload arbitrary applications to the Android applications markets and involves the end-user in granting permissions to applications at install time (Bugiel, 2012). This, however, opens attack surfaces for malicious applications to be installed on users' devices. It is on this background, that since the introduction of Android, a variety of attacks have been reported on Android showing the deficiencies of its security framework and of particular interest and importance in this context are the application level privilege escalation attacks as a result of inter application communication.

Since Android malware is not only often limited to an application's sandbox, but to the collaboration of many applications. Android allows a malicious application to collaborate with other applications in order to access critical resources without requesting for corresponding permissions explicitly (Bugiel, 2011). It is based on this fact that an unprivileged application can take advantage of a privileged applications to perform a privilege escalation attack.

It is on the basis of the existence of application level Privilege Escalation Attacks in Android, that Androids sandbox model is conceptually flawed and hence, not an issue with implementation, but rather a total fundamental flaw.

1.2 Problem Statement

Application level privilege escalation attacks are very much existent in android smartphones due to Android's security model/scheme which puts the burden of enforcing security to application developers who upload applications to a marketplace floaded with uncertified apps. This is error prone as most application developers are not security experts hence not able to prevent transitive usage of permissions that can be leveraged to launch privilege escalation attacks due to inter application communication .

1.3 Justification

- Smartphones with Android OS have become an integral part of our daily life as one of the dominant means of communication. They have increasingly become predominant and proficient at handling a variety of tasks ranging from web-browsing and emailing, to multimedia and entertainment applications (games, videos, audios), navigation, trading stocks, and electronic purchases.
- The current mobile applications business and usage model allows developers to upload arbitrary applications to the Android applications markets and involves the end-user in granting permissions to applications at install time. This, however, opens attack surfaces for malicious applications to be installed on users' devices.
- Android is a privilege separated system, with each application having its own distinct system identity. This security model is not able to detect transitive usage of permissions that can be leveraged to launch privilege escalation attacks.

1.4 Objectives

1.4.1 Main Objective

The main objective of this research is to efficiently automate the profiling and detection of application level privilege escalation attacks on Android.

1.4.2 Specific objectives

- To establish a dataset of malicious applications and a dataset of benign applications, in relation to privilege escalation attacks.
- To review machine learning concepts and its application to the detection of privilege escalation attacks.
- To propose a machine learning algorithm that will better profile and detect privilege escalation attacks on Android.

1.5 Scope

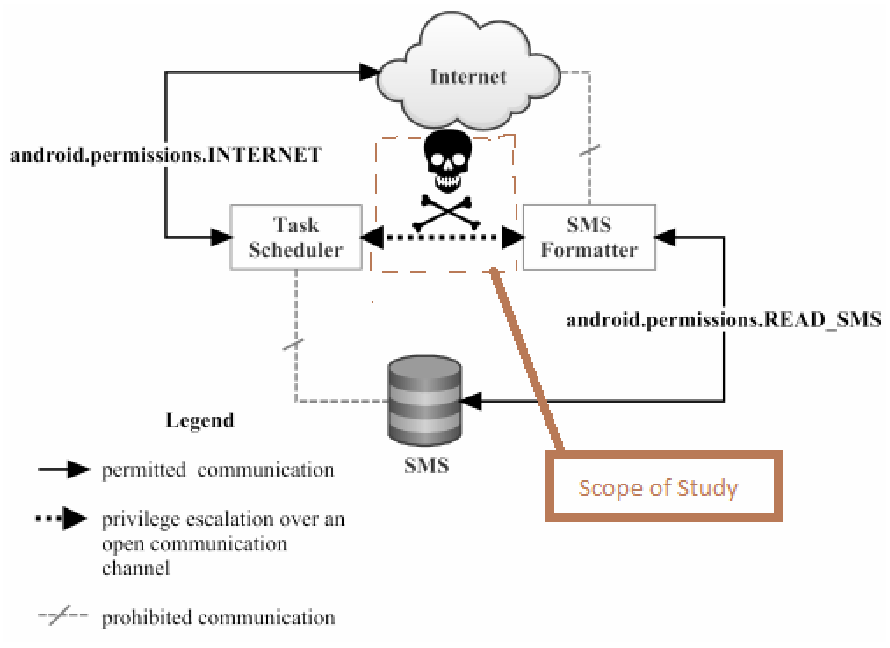
Internet
android.permissions.INTERNET
Task Scheduler
SMS Formatter
android.permissions.READ_SMS
SMS
Scope of Study
Legend
permitted communication
privilege escalation over an open communication channel
prohibited communication

Fig. 2. Scope of the Study: Privilege Escalation Attack due to IAC

2 LITERATURE REVIEW

2.1 Android

2.1.1 Android Operating System Framework

Android includes a Linux kernel, middleware framework, and core applications. The Linux kernel provides low-level services, such as networking, storage, memory, and processing. A middleware layer consists of native Android libraries (written in C/C++), an optimized version of a Java Virtual Machine called Dalvik Virtual Machine (DVM), and core libraries written in Java (Davi, 2010).

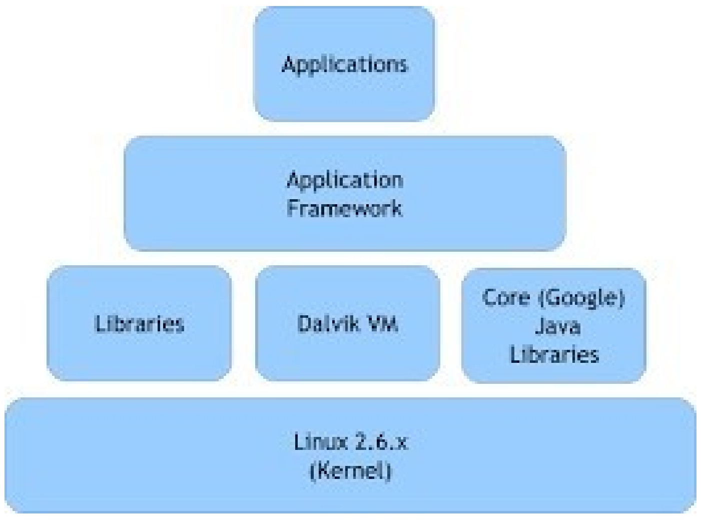

Fig. 2. Android Operating System Framework

The application framework consists of system applications written in C/C++ or Java which provide system services, e.g., Activity Manager manages the life cycle of applications, Application Installer installs new applications, while Package Manager maintains information about all applications loaded in the system (Bugiel, 2011).
Applications consist of four basic types of components:

1. **Activities** are responsible for the user interface, typically each screen shown to a user is represented by a single Activity component.

2. **Services** implement functionality of background processes which are invisible to the user.

3. **Content Providers** are special purpose components which are used for sharing data among applications.
4. **Broadcast Receivers** serve for receiving event notifications from the system and from other applications.

Android employs a quite efficient and convenient IPC mechanism, Binder, which is extensively used for interaction between applications as well as for application-OS interfaces (Zhang, 2013). Binder is implemented as a kernel driver and user-level applications could just interact with it through standard system calls, like open(), ioctl(). Binder is the key infrastructure of Android system and aggressively used to connect various parts of the system together (Zhang, 2013).

To facilitate access of resources from isolated applications, data sharing among applications and the system, Android designs a permission-based security mechanism (Nigam, 2015). Each application needs permissions to access system resources. These permissions are granted from users at install time. At runtime, each application is checked by Android before accessing sensitive resources. Any access to resources without granted permissions is denied (Zhang, 2013).

Wei et al (2012), focuses on understanding how Android permissions and their use evolve in the Android ecosystem, a comprehensive usability study of Android permissions was conducted through surveys in order to investigate Android permissions effectiveness at warning users, which showed that current Android permission warnings do not help most users make correct security decisions. Enck et al. (2014), presented a framework that read the declared permissions of an application at install time and compared it against a set of security rules to detect potentially malicious applications.

Most recently, research results show that in application advertisement, libraries can also actively leak private information. To mitigate that, Pearce et al. (2012) propose AdDroid which separates the advertisement functionality from host apps by introducing a new set of advertising APIs and permissions. AdSplit moves the advertisement code into another process. Moreover, mobile malware may also aggressively collect personal information and upload to remote servers (Zhou, 2013).

In addition, another line of research aims to deal with the classic confused-deputy problem or permission leaks on Android. Examples include ComDroid and Woodpecker, which employ static analysis to identify such problems in either third-party apps or preloaded apps. QUIRE and Felt et al. (2011) propose solutions to mitigate them by checking IPC call chains to ensure unauthorized apps cannot invoke privileged operations.

Kirin is an enhanced installer for Android that extracts the permissions required by the application from the manifest file for each application. These permissions are validated against the organizational policies to verify their compliance to the different stakeholder requirements (Nauman, 2010).

To improve the smartphone security and privacy, a number of platform-level extensions have been proposed by Zhou et al. (2012) specifically, Apex, Mock-Droid, TISSA and AppFence which extend the current Android framework to provide fine grained controls of system resources accessed by untrusted third-party apps. Saint protects the exposed interfaces of an app to others by allowing the app developers to define related security policies for runtime enforcement (Ongtang, 2009).

L4Android and Cells run multiply Operating Systems on a single smartphone for improved isolation and security. None of them characterizes (or studies the evolution of) existing Android malware (Zhou, 2012). Much of the recent work in cell phone security has centered around validating permission assignment at application installation. For example, the Kirin enforces install policies that validate that the permissions requested by applications are consistent with system policy (Ongtang, 2009).

Kirin does not consider run-time policies, and is limited to simple permission assignment. Conversely, the Application Security Framework (Beguelin, 2006) offered by the Open Mobile Terminal Platform (OMTP) recommends a certificate-based mechanism to determine the applications access rights based on its origin. Symbian as discussed by Leone et al. (2013) offers a stricter regimen in the Symbian-signed program. In this program Symbian essentially vouches for applications, and prevents unsigned applications from accessing protected interfaces.

The MIDP 2.0 (Ongtang, 2009) security model regulates sensitive permissions such as network access or file system access based on protection domain defined by Mobile Information Device Profile (MIDP) implementer (e.g., manufacturers and network providers) (Beguelin, 2006). Systems for run-time policy are less developed. The Linux Security Module (LSM) framework has been frequently used to protect Linux phones. For example, the trusted mobile phone reference architecture realized the Trusted Mobile Phone specification using an isolation technique developed for mobile phone platform.

The application of SELinux security policies to Openmoko to ensure the integrity of the phone and trusted applications. In a related work, a mandatory access control (MAC) system for smartphones which uses input from multiple stakeholders to dynamically create the policies run-time permission assignment was developed (Ongtang, 2009).

The Windows Mobile .NET compact framework (Beguelin, 2006; Desmety, 2008) uses security-by-contract that binds each application to a behavioural profile enforced at run-time. This technique was further explored as a means for safely executing potentially malicious code. Techniques such as system call interposition have also been explored for Windows Mobile. None of these systems allow applications to place context-sensitive policies on both the interfaces they use and those that use their interfaces.

2.1.2 Android Security Mechanism

The Android software stack for mobile devices defines and enforces its own security model for apps through its application-layer permissions model. However, at its foundation, Android relies upon the Linux kernel to protect the system from malicious or flawed apps and to isolate apps from one another (Smalley, 2013).

Android security mechanism highly relies on sandboxing, permission framework and application signing.

- **Sandboxing:** is a mechanism to isolate applications from each other and from system resources. Application isolation is done by means of assigning a unique user identifier (UID) to each application, while the underlying Linux kernel enforces discretionary access control to resources (files and devices) by user ownership (Bugiel, 2011). System resources are owned by either system or root. Applications can only access own files or files of others that are explicitly marked as world-wide readable.

- **Application Signing:**, android enforces application signing, however, not centrally since developers themselves have to sign the application code with the self-certified key. Thus, application signing does not provide protection against malware, but helps to establish trust relationships among applications originating from the same developer. Applications signed with the same key may request to share the same UID since they will be placed into the same sandbox.

- **Android Permission Framework:** is provided by the 3rd middleware layer (Backes, 2014). It includes a reference monitor which enforces mandatory access control (MAC) on ICC calls. Security sensitive application programming interfaces (APIs), are protected by permissions. These are security labels which can either be required to enforce access control, or granted to allow access.

  Granted permissions are assigned to application sandboxes and inherited by all application components. Unlike this, required permissions are assigned to application components. Both, required and granted permissions are explicitly specified in an applications manifest file which is included into application installation package. Granted permissions are approved at installation time based on user confirmation. Once granted, permissions cannot be modified (Bugiel, 2011).

2.2 Privilege Escalation Attacks on Android

An application with less permissions (a non-privileged caller) is not restricted to access components of a more privileged application (a privileged callee). In other words, Android's security architecture does not ensure that a caller is assigned at least the same permissions as a callee (Davi, 2010).

2.2.1 Attack Scenario

A user downloads a non-malicious, but vulnerable application from the Internet, for example a chess game that has a memory bug, e.g., suffers from a buffer overflow vulnerability. During the installation, the user grants to the game the permission to access the Internet, e.g., for sharing high-scores with friends. The adversary's goal is to send text messages via SMS to a specified number each time the user saves the game state. To achieve his goals, the adversary exploits the vulnerability of the application and performs a privilege escalation attack in order to gain a permission to sent messages. In such an attack scenario the user most likely will not suspect the game in performing malicious actions since the application was not granted permissions to send text messages.

2.2.2 Proof of Concept

We describe a proof-of-concept example that has been introduced in (Enck, 2008). It illustrates our privilege escalation attack, although it was considered as an example of a poorly-designed application. The attack relies on a vulnerability of a core Android application and allows to make unauthorized phone calls (Davi, 2010). The discovered (and later fixed) vulnerability of the Phone application was that: It had an unprotected component which provided an interface to other applications to make phone calls.

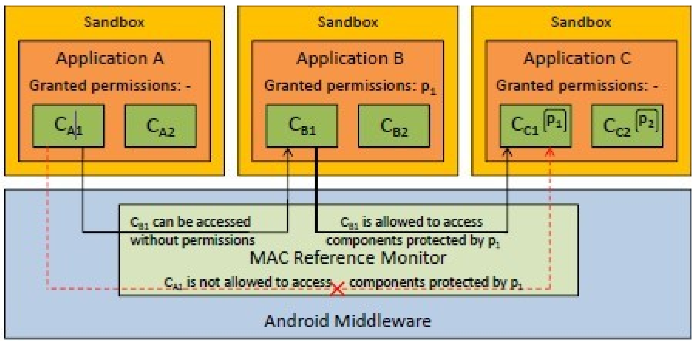

Fig. 2. Component Based Permission Escalation Attack

One could see the Phone application as the application B, while the system interface protected by a system permission PHONE CALLS can be represented as the application C. The role of the application A can be taken by any non-privileged application, e.g., by the Activity Manager (Davi, 2010). The Activity Manager could access the unprotected component of the Phone application when it was invoked from the shell console. As a result, the phone dialled the specified number. In this example, the unprivileged application Activity Manager (am) was able to perform an unauthorized phone call.

2.3 Privilege Escalation Attack Detection Approaches

Current study about the detection of privilege escalation attacks on Android has given birth to several approaches, among which include the following;

- **Static Program Analysis:** Several methods have been proposed for statically inspecting applications and disassembling their code. Kirin (Enck, 2012) checks the permission of applications for indications of privilege escalation attacks. Stowaway (Felt, 2011) similarly analyses API calls to detect over-privileged applications.

- **Dynamic Analysis (Detection at runtime):** DroidScope and TaintDroid (Enck, 2010) enable monitoring of applications dynamically during their execution. The former enables introspection at different layers of the Android platform and the latter focuses on taint analysis. Dynamic analysis is mainly applied for offline detection of privilege escalation attacks, such as scanning and analysing large collections of Android applications.

  Detection through system calls is another type of dynamic analysis. Although some study uses system calls to detect malicious re-packaged applications, they suffer from few drawbacks. They first use the number of system calls, but the metric is too coarse, resulting in low detection accuracy. Secondly, they also need the original benign application so that they can distinguish the number of system calls between the original benign application and the malicious one.

  Lin et al. (2013) used the thread-grained system call sequences, because these sequences can be regarded as the actual behaviour of the application. These approaches need generally more refinements in the choice of features to take into account the sequence of calls. Many works evaluate the detection of privilege escalation attacks with permissions using machine learning on Android (Sanz, 2013a), Aung et al. (2013) realised that a permission-based mechanism can be used as a quick filter to identify malicious applications, and that it must be associated with a second element (such as dynamic analysis) to conduct a complete analysis for a malicious application. However, Tchankounte and Dayang (2014), analysed dangerous combinations of permissions to deny the installation of the application.

- **Security-code repackaging:** This involves rewriting a not trusted application in such a way that the code monitoring the application is directly embedded into its code or its package instead of relying on a system centric solution. Appguard (Backes, 2012) for instance is a reference monitor system, which supports permission revocation. However, this approach can be subject to a high overhead.

2.4 Android Security Extensions

Android has security extensions which have both weaknesses and strength:

1. **Kirin:** Kirin is an extension to **Androids application installer** (Enck, 2008; Enck, 2009). Kirin checks the permissions requested by applications at install-time. It denies the installation of an application if the permissions requested by the application encompass a set of permissions that violates a given system centric policy. The main security goal of Kirin is to mitigate malware contained within a single application.

   The Kirin framework described in (Bugiel, 2012; Enck, 2008) also allows identification of security-critical communication links by analyzing which interfaces the new application is authorized to contact. However, due to its static nature, Kirin has to consider all potential communication links over the unprotected interfaces. This will stop any application from being installed, since applications can potentially establish arbitrary communication links over the unprotected interfaces.

2. **Saint:** Saint (Bugiel, 2012; Ongtang, 2009) introduces a fine-grained access control model that allows application developers to attach security policies to their applications, in particular, to the applications interfaces. It enforces security decisions based on signatures, configurations and contexts (e.g., phone state or location), while security decisions themselves are enforced both at install-time and at runtime.

In order to prevent confused deputy attacks, developers have to assign appropriate security policies on each interface, i.e., they have to specify which permissions/configuration/signature the caller is required to have in order to access an interface. However, if the incentives behind these policies protecting the callee conflict with the properties of the calling application, ICC is denied leading to the problem of caller malfunction or crash. Saint does not address malicious developers, who will not deploy Saint Policies for the obvious reason that they might want to mount a collusion attack.

3. **QUIRE:** QUIRE (Bugiel, 2012; Dietz, 2011) is a recent Android security extension which provides a lightweight provenance system to prevent confused deputy attacks via Binder IPC. In order to determine the originator of a security-critical operation, QUIRE tracks and records the IPC call chain, and denies the request if the originating application has not been assigned the corresponding permission. Due to its application-centric nature, QUIRE cannot prevent colluding applications, because they may drop the IPC call chain and act on their own behalf, and hence, circumvent QUIREs defense mechanism. Furthermore, the unexpected denial of access by the receiver of the call chain might lead to application dysfunction/ crash on the caller's side.

4. **IPC Inspection:** IPC Inspection (Felt, 2011) is a very recent work that is similar to QUIRE and tackles confused deputy attacks via Binder IPC. IPC Inspection reduces the permissions of an application when it receives a message from a less privileged one. IPC Inspection does not require a policy framework, and hence, can prevent unknown attacks without the deployment of appropriate policies.

   However, IPC Inspection does not provide a solution against maliciously colluding applications. Although the receivers' permissions are reduced to the senders' permissions, the individual application instances at the receivers' side still reside in one sandbox and thus are not properly isolated from each other and can communicate freely. Moreover, IPC Inspection will induce a significant performance overhead, because it requires the maintenance of multiple application instances with different sets of privileges (Bugiel, 2012).

5. **TaintDroid:** TaintDroid (Bugiel, 2012; Enck, 2010) is a framework which detects unauthorized leakage of sensitive data. TaintDroid exploits dynamic taint analysis in order to label privately declared data with a taint mark, audit on-track tainted data as it propagates through the system, and warn the user if tainted data aims to leave the system at a taint sink (e.g., network interface). TaintDroid is able to detect data leakage attacks potentially initiated through a privilege escalation attack. However, TaintDroid mainly addresses data flows, whereas privilege escalation attacks also involve control flows.

6. **AppFence:** AppFence (GIZMODO, 2010) builds upon and extends the current TaintDroid framework by allowing users to transparently enable privacy control mechanisms. In particular, AppFence tackles the usability problem when permissions are revoked from applications, e.g., it provides faked or blank data when applications access content providers they should not be allowed to access (Bugiel, 2012). Nevertheless, like TaintDroid, AppFence provides no means to detect privilege escalation attacks beyond data leakage attacks.

7. **SELinux:** SELinux on Android (Bugiel, 2012; Shabtai, 2010) presents a prototype implementation of SELinux on an Android device. Although this work argues for an SELinux-based solution to Androids security vulnerabilities at the kernel level, it does not attempt to provide a solution for the same. Furthermore, the prototype presented in lacks a coordination mechanism between the Linux kernel and the Android middleware.

8. **TrustDroid:** TrustDroid (Bugiel, 2011) is a recent security extension to Android that aims to provide domain isolation, typically between a business domain and a private domain. Thus, TrustDroid builds on a pre-determined basis for classifying applications at install time. On the other hand, our work offers amore generic solution against privilege escalation attacks (Bugiel, 2012). Since there is no predetermined basis to classify applications as in the case of TrustDroid, access control in our solution is more dynamic (requiring decisions to be made on the basis of application behaviour e.g., files read/written), and is carried out at run-time.

   Android mediates access to protected resources using a permission system. However, its effectiveness hinges on app developers correctly implementing it (Luy, 2012).

Apps may be exploitable when servicing external intents.

9. **ComDroid:** ComDroid (Bugiel, 2012; Luy, 2012) was built to identify publicly exported components and warn developers about the potential threats. For this purpose, it is sufficient for ComDroid to only check app metadata and specific API usages, rather than performing an in-depth program analysis as CHEX does. As a result, warned public components are not necessarily exploitable or harmful (i.e. the openness can be by design or the component is not security critical). On the other hand, Android permission system is subject to several instances of the classic confused deputy attack (Marforioy, 2012).

2.5 Conclusion

Google's Android is a fast growing mobile operating system that implements some novel security mechanisms such as privilege separation, however android's security mechanism has flaws resulting in vulnerabilities in applications that can be used to escalate privileges. Android's inter-application communication (IAC) enables complex tasks to be done by cooperation of different applications with different abilities, and is a major feature that differentiates Android from its competitors. However, IAC also utilizes malicious applications to collude in an attack against the privilege system.

There are a number of security mechanisms that have been designed but these solutions do not individually provide a complete solution to detect privilege escalation attacks especially the application level privilege escalation attacks due to inter application communication. With the variety of applications currently on the application markets, vulnerabilities have also increased. Thus, there is a need for a security scheme that provides a complete solution against privilege escalation attacks.

3 METHODOLOGY

3.1 Characterisation and Classification

A dataset of applications with their respective permissions will be obtained from the internet and will undergo a characterisation and classification process. This aims to characterise and to classify applications given a dataset of applications into techniques sub-grouped in approaches based on individual permissions, the combination of permissions, and machine learning techniques. This will help us to identify individually the permissions that are widely requested in both malicious and benign applications.

3.2 Data Extraction and Processing

Using the Santoku Platform, we will employ methods that will extracts different feature sets:

1. the frequency of occurrence of the printable strings
2. the different permissions of the application itself and
3. the permissions of the application extracted from the Android market.

The aim is to classify Android applications into several categories such as entertainment, society, tools, productivity, multimedia and video, communication, puzzle and brain games such that we can develop or propose a scheme and or algorithm that circumvents the permission system by spreading permissions over two or more applications that communicate with each other via arbitrary communication channels.

The Santoku platform is a free environment of integrated development based on the distribution Lubuntu. Santoku offers the usual functionalities of Linux systems; it has been particularly designed to offer tools for static and dynamic analysis of applications on mobile and desktop systems

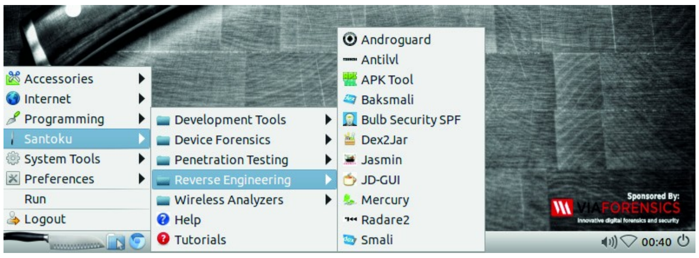

Figure 3: Overview of the Santoku environment

3.3 Analysis

We will use WEKA, an open source software issued under the General Public License (GPL). It is a scientifically well-proven system developed by the University of Waikato in New Zealand, which implements data mining gorithms using JAVA. WEKA is a state-of-the-art collection of machine learning algorithms and their application to real-world data mining problems. It has a collection of machine learning algorithms for data mining tasks. WEKA implements algorithms for data processing, classification, regression, clustering, and association rules, which are directly applied to a dataset. The data file normally used by WEKA is in the ARFF file format, which consists of special tags indicating (foremost: attribute names, attribute types, attribute values, and the data in the data file, but it is also possible to import a .csv file.

Waikato Environment for Knowledge Analysis
Version 3.6.10
(c) 1999 - 2013
The University of Waikato
Hamilton, New Zealand

Figure 4: Overview of WEKA

3.4 Abstract Modeling

To be able to formally state the properties desired of an androids permissions architecture, we shall develop an algorithm (abstract model) of Android-style permissions systems The model will be more general than Android's implementation as its purpose will be to encompass a wider design space of permission systems, including previously suggested android security extensions. We will instantiate our algorithm to describe the key behaviors of Android's permission system. We will then model our classifier algorithm and perform it on our dataset

References

[1] Rangwala, M., Ping, Z., Xukai Z., and Feng L.(2014).A taxonomy of privilege escalation attacks in Android applicationsLtd.International Journal of Security and Networks, Volume 9, Issue 1.

[2] Moore, H. D. (2007). Cracking the iPhone. http://blog.metasploit.com/2007/10/cracking-iphonepart-1.html.

[3] Vennon, T.(2010). Android malware. A study of known and potential malware threats. Technical report, SMobile Global Threat Center.

[4] Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.-R., Shacham, H., and Winandy, M. (2010). Return-oriented programming without returns. In ACM CCS 2010.

[5] Shacham, H. (2007). The geometry of innocent esh on the bone: Return-into-libc without function calls (on the x86). In ACM CCS âĂŹ07, pages 552561.

[6] Dmitrienko, A.(2015). Security and Privacy Aspects of Mobile Platforms and Applications. Cyber-Physical System Security, Darmstadt.

[7] Mathew, R.(2012). Study Of Privilege Escalation Attack On Android And Its Countermeasures. International Journal of Engineering Science and Technology (IJEST).

[8] Nigam, P. P.(2015). VetDroid: Analysis Using Permission for Vetting Undesirable Behaviours in Android Applications. International Journal of Innovative and Emerging Research in Engineering, Volume 2, Issue 3.

[9] Zhang, Y., et al.(2013). Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis. CCS13, November 48, 2013, Berlin, Germany.

[10] Smalley, S., and Craig, R.(2013).Security Enhanced (SE) Android: Bringing Flexible MAC to Android. Trusted Systems Research. National Security Agency.

[11] Backes, M., Bugiel, S., Gerling S., and StypRekowsky, P. V. (2014). Android security framework, extensible multi-layered access control on Android: Proceedings of the 30th Annual Computer Security Applications Conference, Pages 46-55.

[12] Davi L., Dmitrienko, A., Sadeghi, A.-R., and Winandy, M.(2010). Privilege Escalation Attacks on Android. Information Security - 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28

[13] Enck W., Ongtang M., and McDaniel P.(2008). Mitigating Android software mis-use before it happens. Technical Report NAS-TR-0094-2008, Pennsylvania State University.

[14] Ongtang M., Enck, W., and McDaniel P.(2009). On lightweight mobile phone application certification. In 16th ACM conference on Computer and communications security (CCS).

[15] Dietz M., Shekhar S., Pisetsky Y., Shu A., and Wallach, D. S.(2011). QUIRE: Lightweight provenance for smartphone operating systems. In 20th USENIX Security Symposium.

[16] Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2010) . TaintDroid: An informationflow tracking system for realtime privacy monitoring on smartphones. In 9th USENIX

[17] GIZMODO. (2010). http://gizmodo.com/5568458/ toshibaac100-netbook-runs-androidand-has-massive-seven-daysof-standbybattery-life.

[18] Shabtai, A., Fledel, Y., and Elovici, Y.(2010).Securing Androidpowered mobile devices using SELinux. IEEE Security and Privacy, 8(3).

[19] Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.-R., and Shastry, B.(2011). Scalable and lightweight domain isolation on Android. In Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM).

[20] Luy, L., Liz, Z., Wuz, Z., Leey, W., and Jiangz, G.(2012). CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities, Proceedings of the 2012 ACM conference on Computer and communications security 229-2(Sanders, 2010).

[21] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A., and Shastry, B. (2012). Towards Taming Privilege Escalation Attacks on Android,NDSS Symposium.

[22] Leone, M., and Turin.(2013). System for Implementing security on telecommunications terminals. Telecom Italia S.p.A., Milan (1T)

[23] Pearce, P., Felt, A. P., Nunez, G., and Wagner, D.(2012). AdDroid: Privilege Separation for Applications and Advertisers in Android

[24] Marforioy C., Ritzdorfy H., Francillonz A., and Capkun, S. (2012). Analysis of the Communication between Colluding Applications on Modern Smartphones. Proceedings of the 28th Annual Computer Security Applications Conference 51-60.

[25] Wei, X., Gomez, L., Neamtiu, I., and Faloutsos, M.(2012). Permission Evolution in the Android Ecosystem.

[26] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B., Cox, L., Jung, J., McDaniel,P., and Sheth, A.l.(2014). TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, ACM Transactions on Computer Systems (TOCS), 32(2).

[27] Zhou, Y., and Jiang, X. (2013). Detecting Passive Content Leaks and Pollution in Android Applications. Proceedings of the 20th Network and Distributed System Security Symposium (NDSS 2013), San Diego, CA.

[28] Felt, A.P., Wang, H.J., Moshchuk, A., Hanna, S., and Chin, E.(2011). Permission Re-Delegation: Attacks and Defenses. USENIX Security Symposium.

[29] Nauman, M., Khan, S., and Zhang, X. (2010). Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints, Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security Pages 328-332 , 2010

[30] Zhou, Y., and Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. Proceedings of the 2012 IEEE Symposium on Security and Privacy Pages 95-109.

[31] Beguelin, S. Z., Betarte, G., and Luna, C. (2006). A Formal Specification of the MIDP 2.0 Security Model. FCEIA, Universidad Nacional de Rosario, Argentina.

[32] Desmety, L., Jooseny, W., Massacciz, F., Philippaertsy, P., Piessensy, F., Siahaanz, I., and Vanoverbergheу, D. (2008). Security by Contract on the .NET Platform.