# STUDY OF PRIVILEGE ESCALATION ATTACK ON ANDROID AND ITS COUNTERMEASURES

REJO MATHEW

Department of Information Technology, NMIMS University, MPSTME,
Mumbai, Maharashtra 400056, India
rejo.mathew@nmims.edu

**Abstract :**

Android is most commonly used platform for smartphones today which boasts of an advanced security model having MAC and sandboxing. These features allow developers and users to restrict the execution of an application to the privileges assigned. The exploitation of vulnerabilities of the program is confined within the privilege boundaries of an applications sandbox. Privilege escalation attacks have grown manifold as the use of android systems have increased. Different kinds of mechanisms have provided some sort of respite to the developers but the security feature handling by the developers has not helped much. In this paper we discuss the basics of the privilege escalation attack and the various techniques used to counter and prevent this problem.

**Keywords:** privilege escalation attacks ; sandboxing; android security.

## 1. Introduction

The popularity of smartphones and the vast number of the corresponding applications makes these platforms attractive to attackers. Currently, various forms of malware exist for smartphone platforms; including android. Most smart phones rely entirely on application sandboxing and privileged access for security. Applications are isolated and granted privileged permissions only. The application performs actions which are explicitly allowed in the application's sandbox. Android checks corresponding permission assignments at runtime. Hence, an application is not allowed to access privileged resources without having the right permissions.

In this paper we show that Android's sandbox model is conceptually flawed and actually allows privilege escalation attacks. This is not an implementation bug, but rather a fundamental flaw. In Section 2 we discuss the different Android security mechanisms and briefly explain how the privilege escalation attack can be carried out bypassing the sandboxing feature. In Section 3, we show the privilege escalation attack. In Section 4, we discuss the related work for the prevention of this kind of attacks and the various models. In Section 5, we analyze the various countermeasures and desirability of the solutions. In Section 6, we conclude based on observations

## 2. Android Security Mechanisms

Here we discuss the Android security mechanisms in brief.

*Discretionary Access Control (DAC):* The DAC mechanism is based on files (objects) and process (subjects) which access rules. The rules are set and specified to have better access control mechanism.

*Sandboxing:* Android is a privilege separated operating system. Sandboxing isolates applications from each other and from system resources. System files are owned by either the "system" or "root" user, while other applications have own unique identifiers.

*Permission Mechanism:* Applications may declare custom types of permission labels to restrict access to own interfaces. Required permissions are explicitly specified in a Manifest file and are approved at installation time based on checks against the signatures of the applications declaring these permissions and user confirmation. At runtime, the reference monitor checks whether the application of this component possesses requisite permissions.

*Component Encapsulation:* Application components can be specified as public or private.

*Application Signing:* Android uses trust based permission mechanism which is verified by third party. But it need not be signed by a certificate authority. It is just a self signed certificate. The certificate is included in its APK file such that the signature is can be validated at install time.

## 3. Privilege Escalation Attack on Android

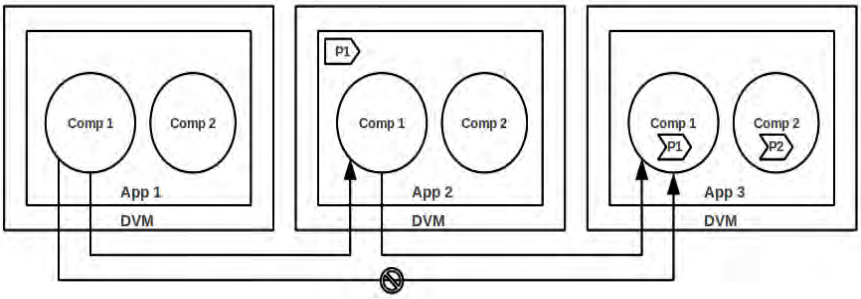


Fig 1: Privilege Escalation Attack on Android

Fig 1 illustrates an example of privilege escalation attack on Android. In the figure, there are three applications running in their own DVMs. Application 1 has no permissions. The components in application 2 is not guarded by any permissions, they are accessible by components of any other application. As a result, both components of application 1 can access components 1 in application 2. Application 2 has permission *P1*, Therefore, both components of application 2 can access component 1 of application 3 which is protected by permission *P1*. From the fig we observe that component 1 of application 1 is accessing component 1 of application 2. But it does not have permission *P1*, so it is not allowed to access component 1 of application 3. On the other hand, application 2 has permission *P1*. Hence, component 1 of application 2 is allowed to access component 1 of application 3. Therefore, although component 1 of application 1 is not allowed to access component 1 of application 3, it can access it via component 1 of application 2. Therefore, the privilege of application 2 is escalated to application 1 in this case. In order to prevent this attack, component 1 of application 2 should enforce that components accessing it must possess permission *P2*. This can be done at code level or by guarding component 1 by permission *P2*. However, this relies on application developers to perform the enforcement at the right places. This is an error prone approach as application developers may not be security experts. [2]

## 4. Related Work

The privilege escalation attack on Android was first proposed by Davi et al. [1] in which they demonstrated an example of the attack. They showed that a genuine application exploited at runtime or a malicious application can escalate granted permissions. However, they did not suggest any defense for the attack in the paper. The most relevant works are security extensions to Android security architecture, namely Saint [12] and Kirin [6, 7], as they could provide some measures against privilege escalation attack. Saint is a policy extension which allows application developers to define comprehensive access control rules for their components. Saint provides a mechanism to ensure that the caller has at least the same permissions as a callee, as a necessary condition to prevent privilege escalation attacks. However, Saint assumes that access to components is implicitly allowed. It provides certain protection against privilege escalation attacks as the application can control which applications can access it. However, this put the burden of enforcing security to application developers which is error prone as most of them are not security experts. Here we see a similarity with the approach undertaken in C/C++ languages to delegate bounds checking to developers. Despite many years of research, attacks that exploit out-of-bounds errors in C and C++ programs are still prevalent: New software bugs continuously appear allowing adversaries to perform runtime exploits. Thus, we believe, similarly it is an error-prone approach to rely on developers to define correct Saint policies or to define them at all.

Kirin is an application certification service to mitigate malware at install time. Kirin is a tool that analyzes Manifest files in the APK of the applications to ensure that granted permissions comply with a system-wide policy. It analyses permissions that require dangerous combinations of permissions [7] or it can analyze a superposition of permissions granted to all applications installed on a platform [6]. However, their approach cannot identify applications vulnerable to privilege escalation attack. The latter approach allows detection of applications vulnerable to privilege escalations attacks as it provides a picture of potential data flows across applications. Nevertheless, as it analyzes potential data flows (as opposite to real data flows) and cannot judge about local security enforcements made by applications (by means of reference monitor

hooks), it suffers from false positives. Thus, it is useful for manual analysis, but cannot provide reliable decisions for automatic security enforcements.
Enck et al. [8] describe Android security mechanisms in details. Burns [3, 4] provides guidance on developing secure applications on the Android platform. Schmidt et al. [14] survey tools which can increase device security and also shows example of Trojan malware for Android [13]. In [11] Nauman et al. proposed permission framework allowing users to approve a subset of permissions the application requires at installation time, and also impose constraints for each permission. Chaudhuri [5] presents a core formal language based on type analysis of Java constructs to describe Android applications abstractly and to reason about their security properties. Shin et al. [18] formalize Android permission framework by representing it as a state-based model which can be proven to be secure with given security requirements by a theorem prover. Barrera et al. [2] propose a methodology to analyze permission usage by various applications and provides results of such an analysis for a selection of 1,100 Android applications. Mulliner[10] presents a technique for vulnerability analysis (programming bugs) of SMS implementations on different mobile platforms including Android. . Shabtai et al. [16, 17] provide a comprehensive security assessment of Android security mechanisms and identify high-risk threats, but do not consider a threat of a privilege escalation attack we describe in this paper. A recent kernel-based privilege escalation attack [9] shows how to gain root privileges by exploiting a memory related vulnerability residing in the Linux kernel. In contrast, our attack does not require vulnerability in the Linux kernel, but instead relies on a compromised (vulnerable or malicious) user space application. Moreover, Shabtai et al. [15] show how to adopt the Linux Security Module (LSM) framework for the Android platform, which mitigates kernel-based privilege escalation attacks such as [9]. Jakobsson et al. [19] proposed a software based attestation approach to detect any malware that executes or is activated by interrupts. Based on memory-printing of client devices, it makes it impossible for malware to hide in RAM without being detected. TaintDroid [20], based on taint analysis, tracks the flow of privacy-sensitive data. When the data are transmitted over the network, users are notified to identify
misbehaving applications. QUIRE [21] is a security solution that can defend against privilege escalation attacks via confused deputy attacks. To address this problem, when there is an Inter Process Communication (IPC) request between Android applications, QUIRE [21] allows the applications to operate with a reduced privilege of its caller by tracking the call chain of IPCs. Chan [36] et al. proposed a vulnerability checking system to detect benign applications which fail to enforce the additional checks on permissions granted.

## 5. Privilege Attack Measures and Considerations

Table 1: Privilege escalation Attack Analysis

| Name of Measure | Type | Technique Used | Effective in | Not Effective in |
|---|---|---|---|---|
| A Vulnerability checking system [2] | Checking permissions | AndroidManifest.xml file used to define permissions to the application | Classifying which applications are vulnerable to attacks | i. Cannot Detect all kinds of privilege escalation attacks<br>ii. Code level checking is missing |
| Kirin [6][7] | Checks security critical vulnerable links | | Focuses on directly reachable interfaces | Transitive permission attacks still possible |
| Saint [12] | Application isolation and protection | Fine grained Access Control Model | Prevention of browser attack [9] | i. Additional security features to the application<br>ii. Developers defining permissions are more error-prone |
| Porscha [23] | Application isolation and protection | Policy-oriented secure content handling | Improvement on model proposed by Saint | i. Data without proper policy tagging can pass through<br>ii. Attacks based on control flows |
| TaintDroid [20] | Detection and checking of privileges | Dynamic Taint Analysis | Addresses data flows | i. Covert channel exploiting sensitive information [24]<br>ii. Attacks based on |

| | | | | |
|---|---|---|---|---|
| | | | | control flows<br>iii. Performance penalty is very high |
| Apex [11] | Application isolation and protection | Deny/accept permission at install time | User friendly and makes Android very flexible | i. Relies on user knowledge<br>ii. Transitive permission attacks still possible |
| CRePE [25] | Deny/accept permissions granting | Context-Related Policy Enforcement for Android | Can use it as company policy and prevent attacks | i. only few functionalities are blocked<br>ii. Transitive permission attacks still possible |
| QUIRE [21] | Prevention of attack especially confused deputy attack | Non System centric system policy | It addresses attacks that exploit vulnerable interfaces of trusted applications | i. Failure to detect and prevent colluding unknown attacks<br>ii. Covert channel exploiting possible |
| IPC Inspection [26] | Prevention of attack especially confused deputy attack | | No policy framework so fast and better results. Can be used to detect and prevent at both install time and runtime | i. Failure to detect and prevent colluding unknown attacks ii. Covert channel exploiting possible<br>iii. Only Control channels covered. Data channels can be exploited<br>iii. Neglects permissions classified as normal<br>iv. Less general than other prevention mechanisms<br>v. Not compatible with legacy Android systems |
| ComDroid [27] Stowaway [28] | Checks security critical vulnerable links | Static Analysis Tool | It warns the developer from broadcasting privacy sensitive data | i. Failure to detect and prevent colluding unknown attacks |
| XmanDroid [22] | Detection and prevention | System centric system policy | i. Prevents attacks on runtime<br>ii. Detects transitive permission usage over any number of hops<br>iii. Handles exceptional cases (e.g.,pending intents and dynamic broadcast receivers). | i. Failure to detect and prevent unknown attacks<br>ii. False detection rates are higher<br>iii. Gets more complex when rate increases |

## 6. Conclusion

Non-privileged applications can escalate permissions by invoking poorly designed higher-privileged applications that do not sufficiently protect their interfaces. Although recently proposed extensions to Android security mechanisms [6,12] aim to address the problem of poorly designed applications, they suffer from practical shortcomings. Saint [12] provides a means to protect interfaces of applications, but relies on application developers to define Saint policies correctly, while Kirin [6] can detect data flows allowing privilege escalation attacks, but results in false positives.

From the analysis we can imply that Android's sandbox model fails to confine boundaries against runtime attacks as the permission system does not check transitive privilege usage. Most of the methods fail to address colluding attacks even though few of them are close enough [22].Looking forward to techniques that can handle all kinds of privilege escalation attacks providing enhanced security keeping developers free from thinking about Android security problems.

## References


[1] L. Davi: A. Dmitrienko: A.-R. Sadeghi: M. Winandy (2010); Privilege escalation attacks on Android, ISC.

[2] Patrick P.F.Chan: Lucas C.K.Hui:S.M. Yui (2011); A Privilege Escalation Vulnerability Checking System for Android Applications, IEEE

[3] J. Burns (2008): Developing secure mobile applications for Android.

[4] J. Burns. Black Hat (2009); Mobile application security on Android.

[5] A. Chaudhuri (2009); Language-based security on Android, ACM SIGPLAN, pages 1–7.

[6] W. Enck:M. Ongtang:P.McDaniel (2008); Mitigating Android software misuse before it happens. Technical Report ,Pennsylvania State University.

[7] W. Enck: M. Ongtang: P.McDaniel (2009); On lightweight mobile phone application certification, ACM CCS '09, pages 235–245.

[8] W. Enck: M. Ongtang: P. McDaniel (2009); Understanding Android security, IEEE Security and Privacy, 7(1):50–57

[9] A. Lineberry:D.L.Richardson:T.Wyatt ( 2010); These aren't t h e permissions you're looking for, BlackHat

[10] C. Mulliner (2009); Fuzzing the phone in your phones, Black Hat USA

[11] M. Nauman: S. Khan: X. Zhang (2010); Apex: Extending Android permission model and enforcement with user-defined runtime constraints,ASIACCS '10, pages 328–332. ACM

[12] M. Ongtang: S.McLaughlin: W. Enck: P. McDaniel ( 2009); Semantically rich application-centric security in Android. In ACSAC '09, pages 340–349. IEEE Computer Society.

[13] A-D.Schmidt: H.-G. Schmidt: L.Batyuk: J. H. Clausen: S.A.Camtepe:S.Albayrak:C. Yildizli ( 2009); Smartphone malware evolution revisited: Android next target?, Malware 2009, pages 1–7.

[14] A.-D. Schmidt: H.G. Schmidt: J. Clausen: K.A.Yuksel:O.Kiraz:A.Camtepe:S.Albayrak (2008); Enhancing security of linux-based Android devices, Lehmann.

[15] A. Shabtai:Y.Fledel:Y.Elovici (2010); Securing Android powered mobile devices using SELinux, IEEE Security and Privacy, 8:36–44.

[16] A. Shabtai:Y. Fledel:U.Kanonov: Y. Elovici:S. Dolev (2009);Google Android: A state-of-the-art review of security mechanisms, CoRR, abs/0912.5101.

[17] A.Shabtai:Y.Fledel:U.Kanonov:Y.Elovici:S.Dolev:C.Glezer (2009); Google Android: A comprehensive security assessment. IEEE Security and Privacy, 8(2):35–44x

[18] W. Shin:S. Kiyomoto: K. Fukushima:T. Tanaka( 2010); A formal model to analyze the permission authorization and enforcement in the Android framework invited paper. In SecureCom 2010

[19] M. Jakobsson: K.-A. Johansson (2010); Retroactive detection of malware with applications to mobile platforms," HotSec'10, pp. 1–13.

[20] W.Enck: P.Gilbert:Sheth.A.N(2010);Taintdroid: An information-flow tracking system for real-time privacy monitoring on smartphones, 9th USENIX Symposium on Operating Systems Design and Implementation

[21] M. Dietz: S.Shekar:Wallach(2011); Quire: lightweight provenance for smartphone operating systems, USENIX Security Symposium

[22] Sven Bugiel:Lucas Davi:Alexandra Dmitrienko:Thomas Fischer:Ahmed-Reza Sadeghi (2011); XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks, Technische University.

[23] M.Ongtang: K.Butler: P.McDaniel (2010); Porscha: Policy oriented secure content handling in Android. In ACSAC'10:

[24] R. Schlegel:K. Zhang: X. Zhou: M. Intwala: A. Kapadia:X. Wang (2011); Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones, NDSS, pages 17-33.

[25] M. Conti: Nguyen:B. Crispo (2010); CRePE: Context-related policy enforcement for Android, ISC 2010

[26] A. P. Felt:H. Wang: A. Moshchuk: S. Hanna:E. Chin (2011); Permission re-delegation: Attacks and defenses. USENIX Security Symposium

[27] E. Chin: A. P. Felt:K. Greenwood:D.Wagner (2011); Analyzing inter-application communication in Android. MobiSys.

[28] A. P. Felt:E. Chin:S. Hanna: D. Song:D. Wagner (2011); Android permissions demystified. TR, Berkeley.