ENHANCE CALLING DEFINITION SECURITY FOR ANDROID CUSTOM PERMISSION

Lanlan Pan[1], Ruonan Qiu, Zhenming Chen, Gen Li, Dian Wen, and Minghui Yang

Guangdong OPPO Mobile Telecommunications Corp. Ltd., Guangdong 518000, China

ABSTRACT

Custom permission is an important security feature of Android system. Permission resource app defines the custom permission. Resource provider app can share the app resources with the resource consumer apps which have gained the custom permission. However, evil app may potentially make permission squatting attacks, get ahead of legitimate permission source app to define the custom permission. If permission squatting attack is successful, then evil app can gain the access to the resource shared by resource provider app, and finally lead to security vulnerabilities and user data leakage. In this paper, we propose a scheme to provide permission source validation for the resource provider apps, which can enhance the calling context security for android custom permission, resistant to permission squatting attack, and suitable for app's self-protection.

KEYWORDS

Android, App, Custom, Permission, Squatting, Security

1. INTRODUCTION

Custom permission is an important security feature of Android system to limit app's access to sensitive data [1][2][3]. Any app can use custom permissions to share its resources with other apps, system provides the permission-based access control. Android system fully trusts the permission source app which first defined the custom permission by default.

However, evil app may potentially make permission squatting attacks, get ahead of legitimate permission source app to define the custom permission. Resource provider app just uses the custom permission directly, without any other more permission source validation to identify the permission is defined by evil app. Therefore, evil app will gain the access to the resource shared by the victim resource provider app, and finally lead to security vulnerabilities and user data leakage [4][5][6].

In this paper, we propose a scheme to provide permission source validation for the resource provider apps, which can be resistant to permission squatting attack, and suitable for app's self-protection. The remainder of this paper is organized as follows. In section 2, we describe the permission squatting attack. In section 3, we discuss about the related work. In section 4, describe our permission source validation scheme in detail. Finally, in section 5, we discuss our work and conclude the paper.

2. Permission Squatting Attack

2.1. App Roles in Custom Permission Scenario

As Table 1 shows, we define 4 app roles in the custom permission scenarios:

- **Permission Source App** defines a custom permission.
- **Resource Provider App** declares to provide some resources to the apps which have gained the custom permission defined by permission source app.
- **Resource Consumer App** requests to consume some resources which provided by resource provider app.
- **Evil App** is developed by the attacker, makes permission squatting attacks, gets ahead of legitimate permission source app to define the custom permission.

Table 1. App Roles.

App Role	Description	Example App Name
Permission Source App	The app that defines a permission.	App P
Resource Provider App	The app that declares to provide some resources.	App R
Resource Consumer App	The app that requests to consume some resources.	App C
Evil App	The app that is developed by the attacker.	App E

2.2. Sample of Custom Permission Usage

User has installed app P, app R, and app C.

Permission source app P defines a permission "com.srv.appP.JUSTFORTEST":

```
<permission
android:name="com.srv.appP.JUSTFORTEST"
android:label="JUSTFORTEST"
android:protectionLevel="signature"
/>
```

Resource provider app R declares a provider, only the apps which gained the "com.srv.appP.JUSTFORTEST" permission can access the provider.

```
<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:exported="true"
/>
```

Resource Consumer app C obtains the "com.srv.appP.JUSTFORTEST" permission, to access the "com.srv.appR.sourceprovider" provider.

```
<uses-permission
android:name="com.srv.appP.JUSTFORTEST"
 />
```

2.3. Permission Squatting Attack

User has installed resource provider app R, but hasn't installed permission source app P.

Resource provider app R declares a provider, only the apps which gained the "com.srv.appP.JUSTFORTEST" permission can access the provider.

```
<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:exported="true"
/>
```

Attacker developed an evil app E, and coax user into installing app E.

User hasn't installed permission source app P, therefore, app E can define a permission "com.srv.appP.JUSTFORTEST".

```
<permission
android:name="com.srv.appP.JUSTFORTEST"
android:label="JUSTFORTEST"
android:protectionLevel="signature"
/>
```

Finally, app E can obtain the “com.srv.appP.JUSTFORTEST” permission, to access the "com.srv.appR.sourceprovider" provider, which may lead to user data leakage. Moreover, because of permission definition collision, the user can’t not install permission source app P if app E has existed.

We perform squatting attack evaluation describe in this section on 6 mobile devices from the following manufacturers: Google, Huawei, OPPO, Samsung, Vivo, and Xiaomi. As Table 2 shows, all the 6 mobile devices are affected by squatting attack. Our experiment code can be found in [7].

Table 2. Squatting Attack Evaluation Result.

Manufacturer	Device Model	Operation System	Squatting Attack
Google	Pixel 3	Android 12	✓
Huawei	P40	HarmonyOS 2.0.0	✓
OPPO	Find X5 Pro	Android 13	✓
Samsung	Galaxy S20 5G	Android 12	✓
Vivo	iQOO 7	Android 13	✓
Xiaomi	Mi 10	Android 12	✓

3. RELATED WORK

Existing Android custom permission security research has discussed the permission squatting attack, mostly focus on the permission vulnerabilities detection, and system access control policy improvement on permission.

Reverse Domain Style Permission Name. Talegaon, S., et al. [5] suggest to regulate custom permission name with the reverse domain style. However, the attacker can develop evil app with the same application name of legitimate app.

Naming Convention. Tuncay, G. S., et al. [6] introduce an internal naming convention, which enforces that all custom permission names are internally prefixed with the source id of the app that declares it. To avoid the package name problem, they instead use the app's signature as the source id to prefix permission name. However, it only works for the apps with same signature, but can't deal with the scenario when permission source app, resource provider app, and resource consumer app are signed by different private keys.

Disallow Custom Permission with The Same Name. Bagheri, H., et al. [8] suggests to disallow multiple apps that define a custom permission with the same name from simultaneously existing on the device. However, we can't assume that the legitimate app can be always installed before the evil app, and the attacker may uninstall legitimate app.

Revoke Permission. Li, R., et al. [9] suggests to revoke permission directly. When the system removes a custom permission, its grants for apps should be revoked. When the system takes the ownership of a custom permission, its grants for apps should be revoked. During the permission update, its grants for apps should be revoked. However, attacker can attract users to grant the evil app's custom permission again, with the same permission name, and users hardly to distinguish it.

Dynamic Enforcement. Hill, M., et al. [10] envisioned an approach in which both users and Android enterprise administrators can actively restrict the functionality of potential permission abusing apps by leveraging the dynamic permission updates provided by Android enterprise. However, the management work is heavy, such as identify attack patterns and potential templates for counter-policies.

4. PROVIDE PERMISSION SOURCE VALIDATION FOR THE RESOURCE PROVIDER APPS

The root cause of permission squatting attack is the android custom permission trust scheme. Android system fully trusts the app which first defined the custom permission, if the evil app can make the permission definition squatting, the attack comes. However, there are few discussions about how to enhance resource provider app's self-protection on custom permission, which can help resource provider app to provide its resource to the right consumer app with the right permission defined by the right permission source app.

To address the permission squatting attack mentioned above, we describe a permission source validation scheme for the resource provider app. We add three permission attributes for the permission source validation:

- permission_srcSecondLevelDomainNames
- permission_srcProtectLevel
- permission_srcSignerCerts

4.1. permission_srcSecondLevelDomainNames

permission_srcSecondLevelDomainNames means that, the resource provider app assumes that the permission source application name must belong to some second level domain names.

For example, resource provider app R declares a provider, the permission source application name should belong to "com.srv", or "com.service". Application name "com.srv.appP" matches "com.srv".

```
<string-array name="srcSecondLevelDomainNamesForP">
<item>com.srv</item>
<item>com.service</item>
</string-array>

<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:permission_srcSecondLevelDomainNames="@array/srcSecondLevelDomainNamesForP"
android:exported="true"
/>
```

4.2. permission_srcProtectLevel

permission_srcProtectLevel means that, the resource provider app assumes that the permission source app should belong to one of the protect levels below:

- permission_srcProtectLevel = signature: the permission source app should have the same signer certificate with the resource provider app.
- permission_srcProtectLevel = privileged: the permission source app should be privileged app [11].
- permission_srcProtectLevel = knownSigner: the permission source app's signer certificate digest should be listed in the permission_srcSignerCerts.

For example, resource provider app R declares a provider, the permission source app should have the same signer certificate with the resource provider app.

```
<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:permission_srcProtectLevel="signature"
android:exported="true"
/>
```

For example, resource provider app R declares a provider, the permission source app should be privileged app.

```
<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:permission_srcProtectLevel="privileged"
android:exported="true"
/>
```

4.3. permission_srcSignerCerts

permission_srcSignerCerts contains some signer certificate digests, which are used for the permission_srcProtectLevel = knownSigner scenario. The digest should be computed over the DER encoding of the signer certificate using the SHA-256 digest algorithm.

For example, resource provider app R declares a provider, the permission source app's signer certificate digest should be listed in the permission_srcSignerCerts.

```
<string-array name="srcSignerCertsForP">
<item> 657d6f7c6295d453f027a8cc4ce528f411d95276cca140f540c53f396df1ceff </item>
</string-array>

<provider
android:authorities="com.srv.sourceprovider"
android:name="com.srv.appR.sourceprovider"
android:permission="com.srv.appP.JUSTFORTEST"
android:permission_srcProtectLevel="knownSigner"
android:permission_srcSingerCerts="@array/srcSignerCertsForP"
android:exported="true"
/>
```

4.4. Permission Source Validation

As Fig. 1 shows, system can make permission source validation for the resource provider apps based on above permission attributes.

- Compared to reverse domain style permission name described in [5], permission_srcSecondLevelDomainName is more flexible for application name changing.
- Compared to naming convention described in [6], permission_srcProtectLevel = knownSigner could support the scenario when permission source app, resource provider app, and resource consumer app are signed by different private keys.
- Compared to disallow custom permission with the same name described in [8], permission_srcProtectLevel=signature/knownSigner could prevent the evil app from getting resource provider's data even when the evil app has successful declared the same permission name on Android system.
- Compared to revoke permission described in [9], permission_srcProtectLevel=signature/privileged/knownSigner could support more loose access control policy with less permission revoked work, because the system can confirm that the permission is updated by some legitimate permission source app through permission_srcProtectLevel check.

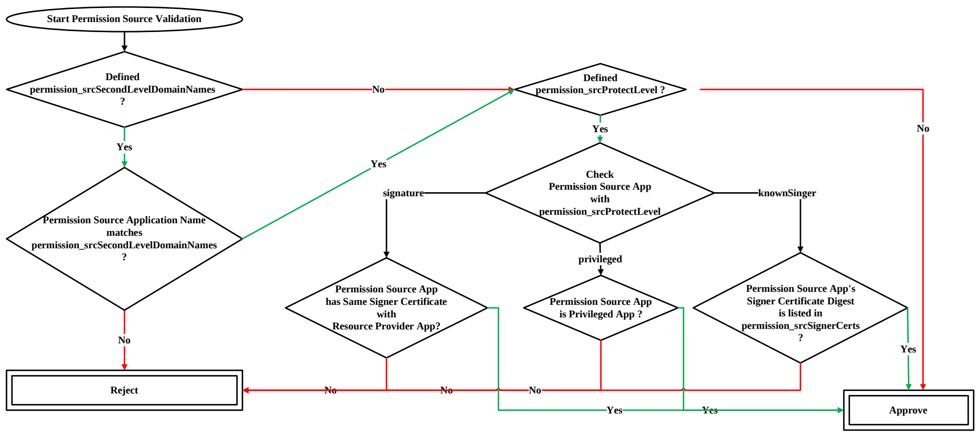
Start Permission Source Validation
Defined permission_srcSecondLevelDomainNames ?
No
Defined permission_srcProtectLevel ?
No
Yes
Yes
Yes
Check Permission Source App with permission_srcProtectLevel
signature
knownSinger
Permission Source Application Name matches permission_srcSecondLevelDomainNames ?
privileged
Permission Source App has Same Signer Certificate with Resource Provider App?
Permission Source App is Privileged App ?
Permission Source App's Signer Certificate Digest is listed in permission_srcSignerCerts ?
No
Yes
Reject
No
No
No
Yes
Yes
Approve

Figure 1. Permission Source Validation

5. Conclusion

This paper describes a scheme to provide permission source validation for the resource provider apps, which can be resistant to permission squatting attack. We define three permission attributes to enhance the calling context security for android custom permission, which are suitable for app's self-protection.

In this paper, we don't discuss about the system how to identify evil app on app store, or the system how to make malware detection when installing app. We don't mention about the attacker how to attract user to install evil app before legitimate permission source app. We focus on the permission source configuration improvement at resource provider app.

We believe that enhance calling definition security can mitigate the evil app's attack on android custom permission. Our future work is to do more experiments on android system, and suggest Google to implement our scheme in the future android version.

References

[1] Define a custom app permission, https://developer.android.com/guide/topics/permissions/defining

[2] Zhou, H., Luo, X., Wang, H., & Cai, H. (2022, November). Uncovering Intent based Leak of Sensitive Data in Android Framework. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (pp. 3239-3252).

[3] Yang, Y., Elsabagh, M., Zuo, C., Johnson, R., Stavrou, A., & Lin, Z. (2022, November). Detecting and Measuring Misconfigured Manifests in Android Apps. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (pp. 3063-3077).

[4] The Custom Permission Problem, https://github.com/commonsguy/cwac-security/blob/master/PERMS.md

[5] Talegaon, S., & Krishnan, R. (2020). A formal specification of access control in android. In Secure Knowledge Management In Artificial Intelligence Era: 8th International Confer-ence, SKM 2019, Goa, India, December 21–22, 2019, Proceedings 8 (pp. 101-125). Springer Singapore.

[6] Tuncay, G. S., Demetriou, S., Ganju, K., & Gunter, C. (2018). Resolving the predicament of android custom permissions.

[7] Squatting Attack，https://github.com/JimmyChenX/SquattingAttackInAndroidCustomPermissions

[8] Bagheri, H., Kang, E., Malek, S., & Jackson, D. (2018). A formal approach for detection of security flaws in the android permission system. Formal Aspects of Computing, 30, 525-544.

[9] Li, R., Diao, W., Li, Z., Du, J., & Guo, S. (2021, May). Android custom permissions de-mystified: From privilege escalation to design shortcomings. In 2021 IEEE Symposium on Security and Privacy (SP) (pp. 70-86). IEEE.

[10] Hill, M., Rubio-Medrano, C. E., Claramunt, L. M., Baek, J., & Ahn, G. J. (2021, September). Poster: DyPolDroid: User-Centered Counter-Policies Against Android Permission-Abuse Attacks. In 2021 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 704-706). IEEE.

[11] Privileged App, https://source.android.com/docs/core/permissions/perms-allowlist

AUTHOR

Lanlan Pan, my current research interests in cryptographic protocols, mobile security, automotive security, and cybersecurity.