Andoid Custom Permissions Demystified : From Privilege Escalation to Design Shorcomings

Authors : Rui Li, Wenrui Diao, Zhou Li, Jianqui Du, Shanqing Guo.

Published in IEEE S&P 2021

Siva Sai Nagender Vasireddy

September 7, 2021

Android Permission Mechanism

- Permission is the fundamental security mechanism for protecting user data and privacy on Android.
  - Any app must request specific permissions to access the corresponding sensitive user data and system resources.

- Types of Permissions
  - System Permissions
  - Custom Permissions

System and Custom Permissions

- System Permissions
  - Defined by system apps
  - Protect system resources
  - Focus of most of the previous research on security issues
- Custom Permissions
  - Defined by third-party apps
  - Protect apps' own resources
  - Focus of the paper being presented
  - 52,601 (about 25.2%) apps declare 82,052 custom permissions

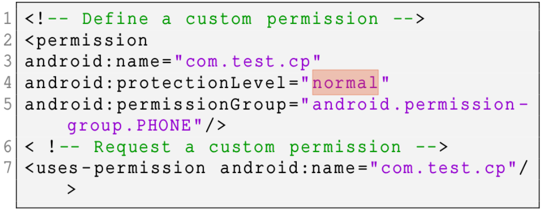
Android permission mechanism

- Apps declare required permissions in their manifest files.
- Permissions are mainly divided into three protection levels.
  - Normal
  - Signature
  - Dangerous

Permission	Granted during	Requires	Revocable?
Normal	Installation	non-sensitive resource	No
Signature	Installation	Signature Certificate	No
Dangerous	Runtime	User Permission	Yes

- All dangerous permissions belong to permission groups. Access to one dangerous permission gives access to all other dangerous permissions in that group.
  - Custom permissions can be added to an existing system group or to a custom group.
  - A permission at any protection level can be assigned to a group.

Example of a custom permission



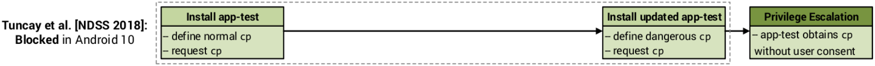
```
<!-- Define a custom permission -->
<permission
android:name="com.test.cp"
android:protectionLevel="normal"
android:permissionGroup="android.permission-group.PHONE"/>
< !-- Request a custom permission -->
<uses-permission android:name="com.test.cp"/>
```

Listing 1: Define and request a custom permission.

- Android mechanisms to ensure custom permissions will not affect the scope of system permissions
    - Cannot define a permission with the same name as an existing permission.
    - Permission owner is the app that defines the permission first.
    - System apps are installed before any third-party apps and first define a set of permissions to protect specific platform resources.

Permission upgrade attack

Issue : When an app obtains dangerous custom permission without user consent through privilege escalation.



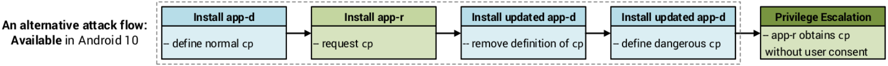
Tuncay et al. [NDSS 2018]:
Blocked in Android 10

Install app-test
– define normal cp
– request cp

Install updated app-test
– define dangerous cp
– request cp

Privilege Escalation
– app-test obtains cp without user consent

Google has fixed the above attack in Android 10 by **preventing** the permission level changing operation from **normal or signature to dangerous**.



An alternative attack flow:
Available in Android 10

Install app-d
– define normal cp

Install app-r
– request cp

Install updated app-d
-- remove definition of cp

Install updated app-d
– define dangerous cp

Privilege Escalation
– app-r obtains cp without user consent

How do we identify such design shortcomings lying in the permission framework?

Automatic Analysis

- There exist two ways to conduct automatic analysis for custom operations
  - Static Analysis
    - Analyzing source code of Android OS to find design flaws
    - Difficult approach as the internal implementation of the permission mechanism is quite complicated.
  - Dynamic Analysis
    - Executing numerous test cases to trigger unexpected behaviours.
    - The permission model can be treated as a black box.

- **CuPerFuzzer** : An automated fuzzing tool designed to trigger privilege escalation issues by executing massive test cases.

CuPerFuzzer

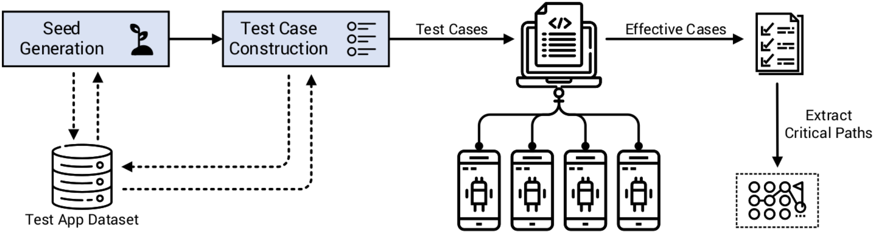
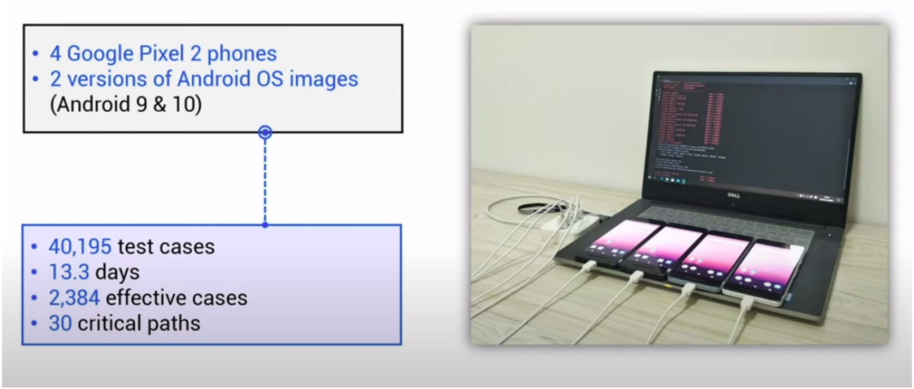


Fig. 2: Overview of CUPERFUZZER.

Test cases consist a sequence of app installation, app uninstallation and OS update operations.

Implementation and Experiment Results



- 4 Google Pixel 2 phones
- 2 versions of Android OS images (Android 9 & 10)

- 40,195 test cases
- 13.3 days
- 2,384 effective cases
- 30 critical paths

Four fatal design shortcomings have been identified after analyzing the 30 critical paths and the corresponding source code of Android OS.

DS #1 : Dangling custom permission

- If the removed custom permission is an install-time permission, the corresponding permission granting status of apps will be kept, causing dangling permission.

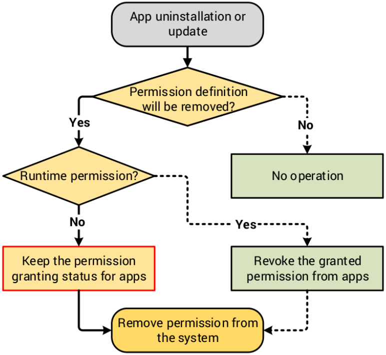
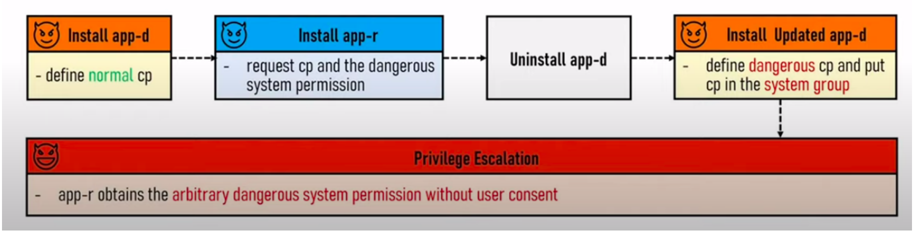

App uninstallation or update
Permission definition will be removed?
Yes
No
Runtime permission?
No operation
No
Yes
Keep the permission granting status for apps
Revoke the granted permission from apps
Remove permission from the system

Fig. 4: Dangling custom permission.

DS #1 : Example



Install app-d

- define normal cp

Install app-r

- request cp and the dangerous system permission

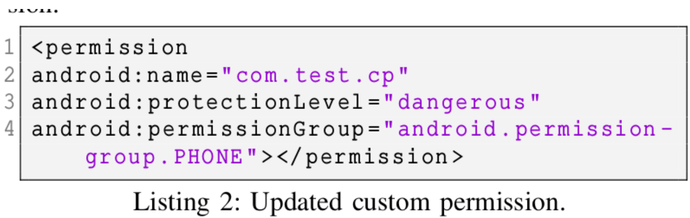
Uninstall app-d

Install Updated app-d

- define dangerous cp and put cp in the system group

Privilege Escalation

- app-r obtains the arbitrary dangerous system permission without user consent



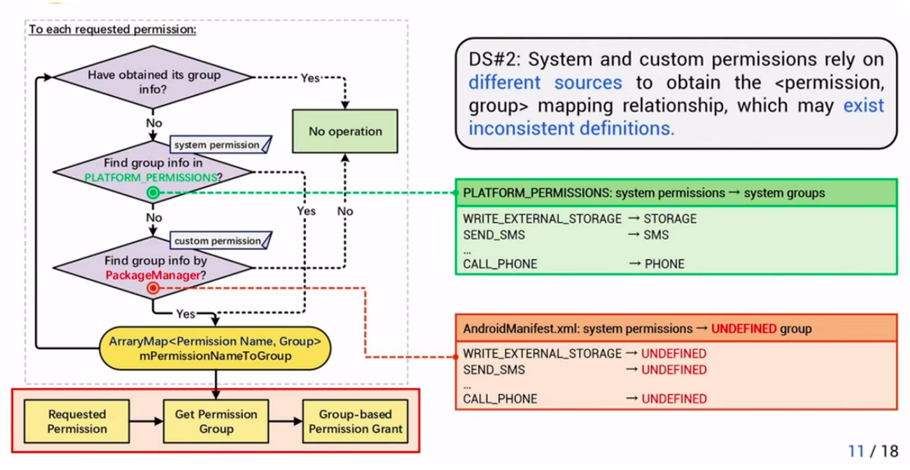
sion.

```
<permission
android:name="com.test.cp"
android:protectionLevel="dangerous"
android:permissionGroup="android.permission-
    group.PHONE"></permission>
```

Listing 2: Updated custom permission.

Demo.

DS #2 : Inconsistent permission group mapping


To each requested permission:
Have obtained its group info?
Yes
No operation
No
system permission
Find group info in PLATFORM_PERMISSIONS?
Yes
No
No
custom permission
Find group info by PackageManager?
Yes
ArraryMap<Permission Name, Group> mPermissionNameToGroup
Requested Permission
Get Permission Group
Group-based Permission Grant

DS#2: System and custom permissions rely on different sources to obtain the <permission, group> mapping relationship, which may exist inconsistent definitions.

PLATFORM_PERMISSIONS: system permissions → system groups

WRITE_EXTERNAL_STORAGE	→ STORAGE
SEND_SMS	→ SMS
...	
CALL_PHONE	→ PHONE

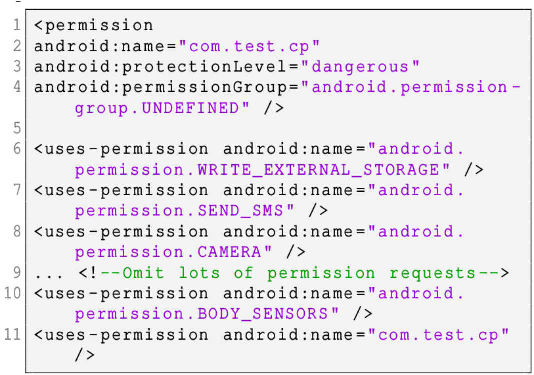
AndroidManifest.xml: system permissions → UNDEFINED group

WRITE_EXTERNAL_STORAGE	→ UNDEFINED
SEND_SMS	→ UNDEFINED
...	
CALL_PHONE	→ UNDEFINED

11 / 18

DS # 2: Example

- Install an app app-ds2 which requests WRITE_EXTERNAL_STORAGE permission. Grant the permission.
- Update app-ds2 to create a dangerous custom permission com.test.cp. Request the custom permission along with all dangerous system permissions as shown below

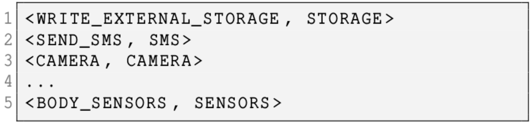


```
<permission
android:name="com.test.cp"
android:protectionLevel="dangerous"
android:permissionGroup="android.permission-
    group.UNDEFINED" />

<uses-permission android:name="android.
    permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.
    permission.SEND_SMS" />
<uses-permission android:name="android.
    permission.CAMERA" />
... <!--Omit lots of permission requests-->
<uses-permission android:name="android.
    permission.BODY_SENSORS" />
<uses-permission android:name="com.test.cp"
    />
```

Listing 3: Updated version of app-ds2.

DS #2: Example Contd...

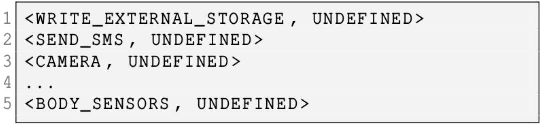
- To system permissions (Line 6-10), the <permission, group>mapping looks like:



```
<WRITE_EXTERNAL_STORAGE, STORAGE>
<SEND_SMS, SMS>
<CAMERA, CAMERA>
...
<BODY_SENSORS, SENSORS>
```

Listing 4: Mapping mPermissionNameToGroup.

- When reaching the custom permission (Line 11), since it belongs to the UNDEFINED group, and this group contains all dangerous system permissions. The mapping is refreshed as:



```
<WRITE_EXTERNAL_STORAGE, UNDEFINED>
<SEND_SMS, UNDEFINED>
<CAMERA, UNDEFINED>
...
<BODY_SENSORS, UNDEFINED>
```

Listing 5: Updated mapping mPermissionNameToGroup.

DS #2: Example Contd...

- Therefore, under this situation, if one dangerous permission (WRITE_EXTERNAL_STORAGE) has been granted, the other dangerous permissions will be granted without user permitting because they belong to the same permission group, that is, android.permission-group.UNDEFINED.
- Demo.

DS #3 : Custom permission elevating

- When Android OS overrides a custom permission (changing the owner), the granting status of this permission is not revoked, further resulting in permission elevating.

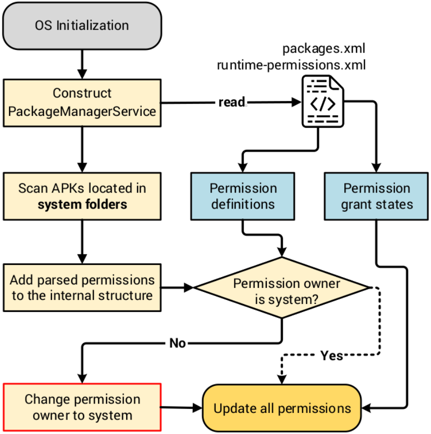
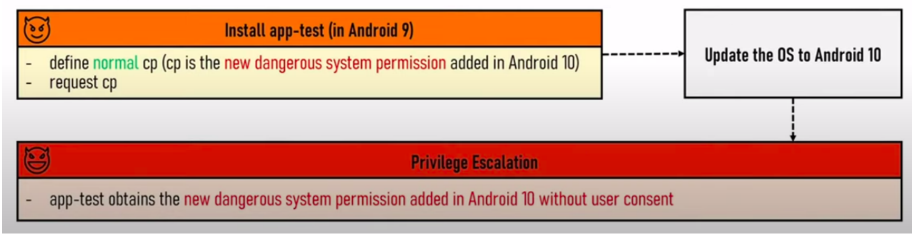

OS Initialization
packages.xml
runtime-permissions.xml
Construct PackageManagerService
read
Scan APKs located in **system folders**
Permission definitions
Permission grant states
Add parsed permissions to the internal structure
Permission owner is system?
No
Yes
Change permission owner to system
Update all permissions

Fig. 6: Custom permission elevating.

DS #3 : Example



Install app-test (in Android 9)

- define normal cp (cp is the new dangerous system permission added in Android 10)
- request cp

Update the OS to Android 10

Privilege Escalation

- app-test obtains the new dangerous system permission added in Android 10 without user consent

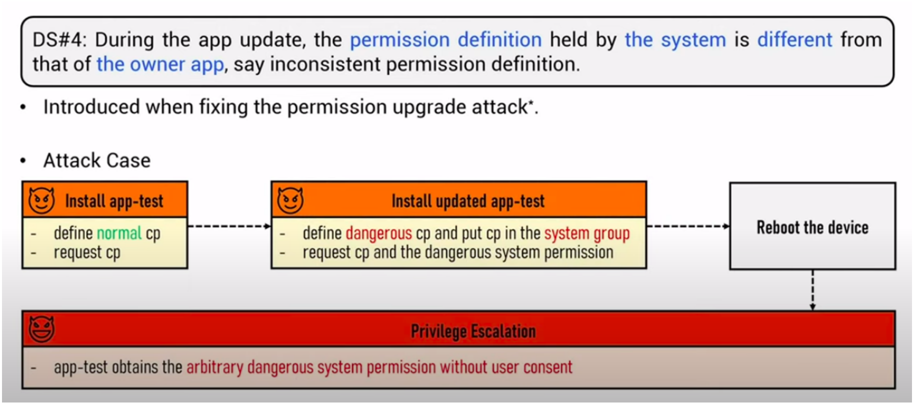
```
<permission
android:name="android.permission.
    ACTIVITY_RECOGNITION"
android:protectionLevel= "normal"/>

<uses-permission android:name="android.
    permission.ACTIVITY_RECOGNITION" />
```

Listing 6: Define and request ACTIVITY_RECOGNITION.

DS #3 : Example Contd...

- ACTIVITY_RECOGNITION permission is a new dangerous system permission introduced in Android 10.
- On devices running Android 9, ACTIVITY_RECOGNITION is only treated as a normal custom permission.
- After updating the system to Android 10 and finishing OS initialization, app-ds3 has been granted with the ACTIVITY_RECOGNITION permission (dangerous system permission) automatically, say privilege escalation.
- Demo.

DS # 4 : Inconsistent permission definition



DS#4: During the app update, the permission definition held by the system is different from that of the owner app, say inconsistent permission definition.

- Introduced when fixing the permission upgrade attack*.

- Attack Case

Install app-test
- define normal cp
- request cp

Install updated app-test
- define dangerous cp and put cp in the system group
- request cp and the dangerous system permission

Reboot the device

Privilege Escalation
- app-test obtains the arbitrary dangerous system permission without user consent

DS # 4 : Inconsistent permission definition

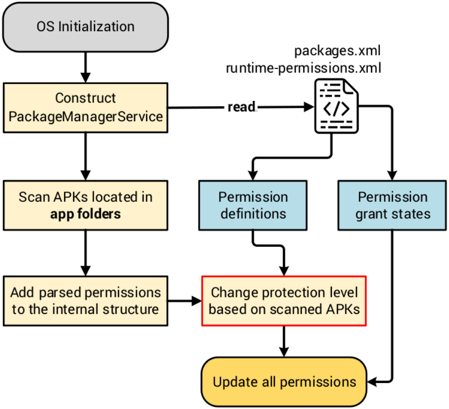
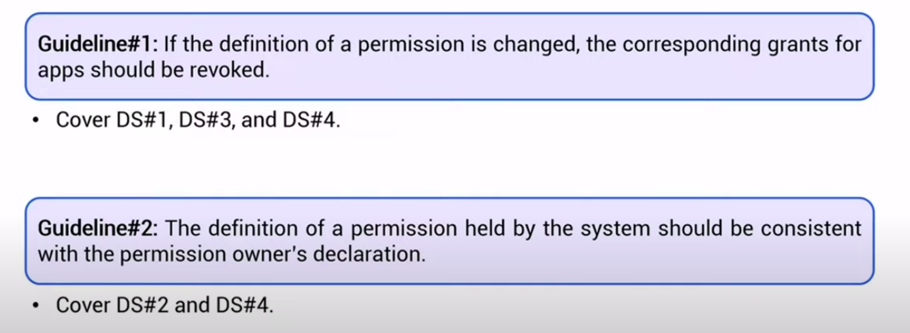

OS Initialization
Construct PackageManagerService
read
packages.xml
runtime-permissions.xml
Scan APKs located in **app folders**
Permission definitions
Permission grant states
Add parsed permissions to the internal structure
Change protection level based on scanned APKs
Update all permissions

Fig. 7: Inconsistent permission definition.

Demo.

Proposed design guidelines



Guideline#1: If the definition of a permission is changed, the corresponding grants for apps should be revoked.

- Cover DS#1, DS#3, and DS#4.

Guideline#2: The definition of a permission held by the system should be consistent with the permission owner's declaration.

- Cover DS#2 and DS#4.

Conclusion

- The presented work introduced a tool to detect some vulnerabilities caused in Android OS due to custom permissions.
- The proposed testing method is based on generating random test cases and the scope of the test cases considered is limited. For example, the maximum number of actions in a sequence is limited to 5.
- The authors have exposed some critical vulnerabilities in the permission mechanism. But there is scope to detect more vulnerabilities by considering the source code and generating more interesting test cases.