Progress

2020019252 김나현

Reading Paper

Android Custom Permissions Demystified:

From Privilege Escalation to Design Shortcomings

previous researches -> focused on the security issues of system permissions

the study of Tuncay et al. is the only work focusing on the security of custom permissions

They manually discovered two privilege escalation attacks
: exploit the permission upgrade and naming convention flaws.

though both attacks have been blocked,
custom permission based attacks can still be achieved with alternative execution paths by passing the fix

In this paper,

systematically evaluate the design and implementation of Android custom permissions

built an automatic fuzzing tool, called CUPERFUZZER, to detect custom permissions related vulnerabilities existing in the Android OS.

CUPERFUZZER…

treats the operations of the Android permission mechanism as a black-box and dynamically generates massive test cases for fuzzing

discovered 2,384 successful exploit cases after executing 40,195 fuzzing tests.

These effective cases were further converted to 30 critical paths, say the least necessary operations triggering a privilege escalation issue.

-> identified 4 severe design shortcomings in the Android permission framework.

Android Permission Mechanism

- 3 protection levels: normal, signature, and dangerous.

install-time permissions

cannot be revoked by users once they are granted

runtime permissions

can be revoked at any time.

All dangerous permissions belong to permission groups.

Ex. both READ_SMS and RECEIVE_SMS belong to the SMS group

dangerous permissions are granted on a group basis.

If an app requests dangerous permissions belonging to the same permission group,
once the user grants one, the others will be granted automatically without user confirmation.

Custom Permissions

system permissions are the permissions defined by system apps located in system folders to protect specific system resources.

Ex) an app must have CALL_PHONE permission to make a phone call.

```
<!-- 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"/
    >
```

For third-party apps, they can define their own permissions as well, called custom permissions, to share their resources and capabilities with other apps.

The app must specify the permission name and protection level

system permissions VS custom permissions

defined by the system (system apps)

defined by third-party apps

system apps are pre-installed and cannot be modified or removed by users.
Accordingly, their defined permissions are stable,
including names, protection levels, grouping, and protected system components.

On the other hand, users can install, uninstall, and update arbitrary third-party apps,
making the usage of custom permissions more flexible.

Usage Status

we conducted a large-scale measurement based on 208,987 APK files
we developed a script to scan the manifest files of apps.

Through parsing custom permission related attributes,
we obtained the first-hand statistics data for further processing.

1) How many apps use custom permissions?
   -> 52,601 apps (around 25.2%) declare a total of 82,052 custom permissions

TABLE I: Protection levels of custom permissions.

Protection Level	Amount	Percentage
`normal`	26,330	32.09%
`dangerous`	1,986	2.42%
`signature`†	53,724	65.48%
`instant`‡	12	0.01%

TABLE II: Permission groups of custom permissions.

Group Type	Amount	Percentage
System Group	4,526	83.64%
Custom Group	885	16.36%

Usage Status

2) What are the purposes of using custom permissions?

-> we crawled the custom permission names and their permission descriptions for analysis. Combined with a number of manual case studies, here we summarize the purposes of using custom permissions.

• Use services provided by third-parties.
Ex. up to 16,259 apps in our dataset declare the JPUSH_MESSAGE permission
to obtain the message push service offered by the JPush platform.

• Restrict the accessing to apps' shared data.
Ex. com.qidian.QDReaderMM" defines the READ_DB4 permission to control the accessing to its database of e-books.

• Control the communication between apps.
Ex. only the apps with the BROADCAST_RECEIVER5 permission
can send a broadcast to the broadcast receiver of com.tencent.portfolio which defines this permission.

MOTIVATION AND THREAT MODEL preliminary exploration

privilege escalation

-> this attack has been fixed on Android 10

Google's fix prevents the permission protection level changing operation
– from normal or signature to dangerous.

However, another app execution path still can achieve the same consequence, which bypasses the fix.

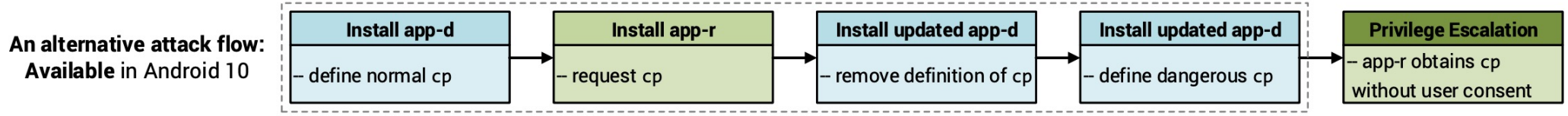


Fig. 1: An alternative attack flow achieving privilege escalation.

Automatic Analysis

- static analysis (e.g., analyzing the source code of Android OS to find design flaws)
- ✓ dynamic analysis (e.g., executing multitudinous test cases to trigger unexpected behaviors)

inspired by the motivation case,
the analysis process could be abstracted as finding specific app execution sequences
that can trigger privilege escalation issues.

The internal operations of the permission mechanism could be treated as a blackbox accordingly.

Design of CUPERFUZZER

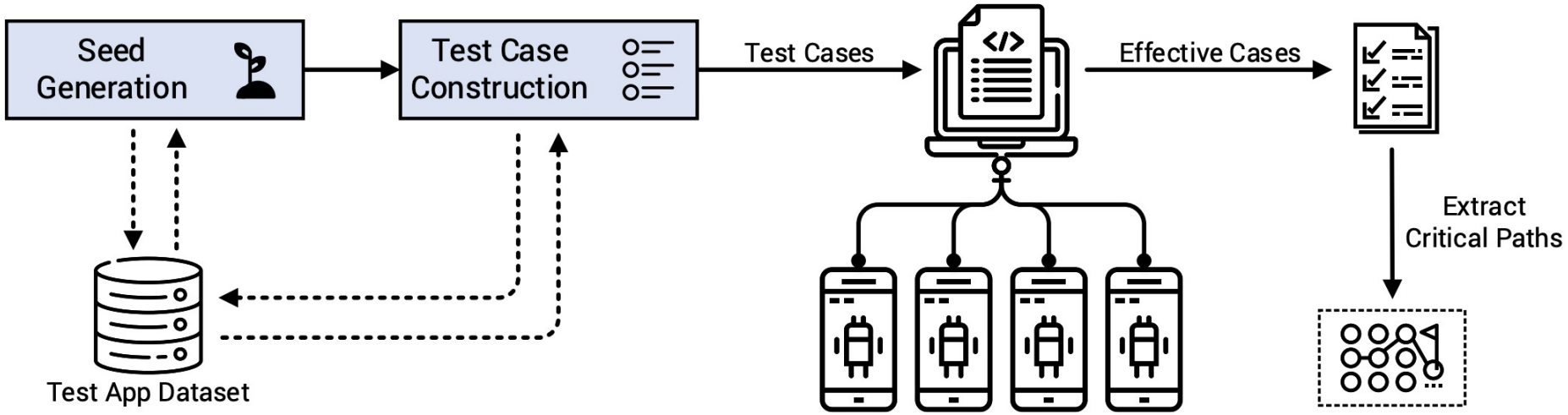


Fig. 2: Overview of CUPERFUZZER.

- Seed Generation.
- Test Case Construction.
- Test Case Execution.
- Effective Case Checking
- Critical Path Extraction.

Seed Generation

- Seed Variables.
  - Permission name: based on a pre-defined list but cannot be the same as a system permission.
  - Protection level: normal, dangerous, or signature.
  - Group: a certain system group or not set.
- Seed Generation Modes

  The key components of the seed app can be split into apps that define the custom permission and apps that request permissions. they are signed by different certificates.
  ->thus, there are two seed generation modes, say single-app mode and dual app mode.
- Seed Generation

  When running tests,
  CUPERFUZZER randomly selects an app from the prepared dataset as the seed
  and quickly activates the fuzzing process.

Test Case Construction

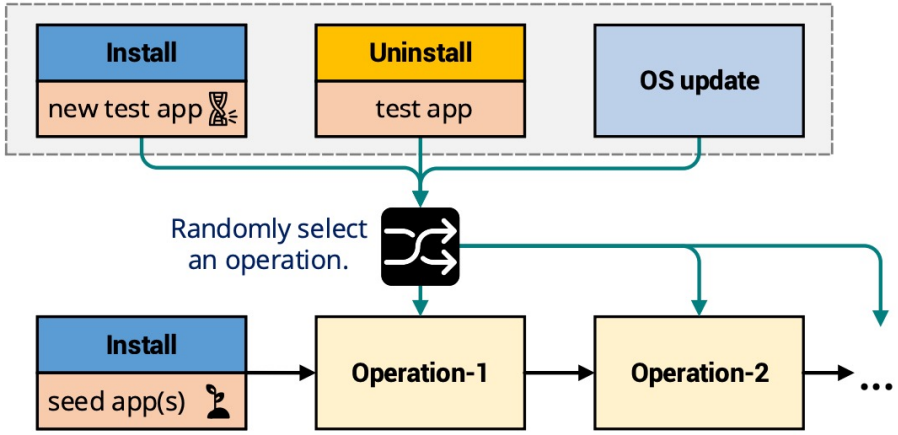


Fig. 3: Construct a test case (an execution sequence).

it is an execution sequence consisting of multiple test apps
and operations that may affect the granting of requested permissions.

Operation Selection.

app installation, app uninstallation, app update, and OS update.

- When installing a new app, new custom permission definitions may be added to the system.
- When uninstalling an app, existing custom permission definitions may be removed.
- When updating an app, existing custom permission definitions may be updated or removed.
- During major OS updates, new system permissions may be added to the system, and existing system permissions may be removed.

Test Case Construction

- The first operation must be seed app installation

because the fuzzing execution environment (physical phone) will be reset before every test.

- Randomy delect an Operation.

Design of CUPERFUZZER

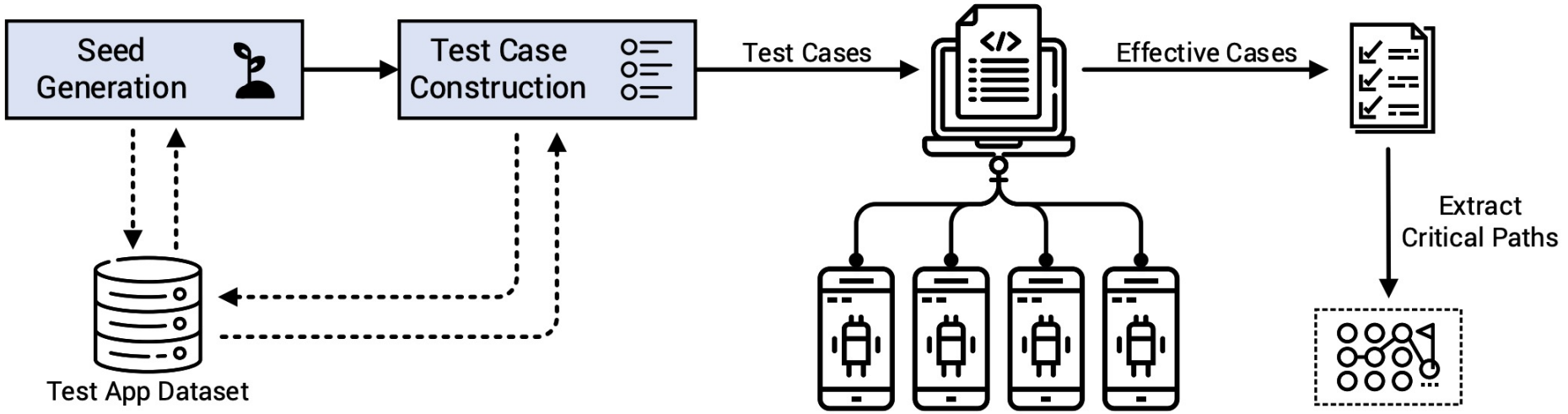


Fig. 2: Overview of CUPERFUZZER.

(1) Test Cases Classification.
(2) Find Critical Path.
(3) Delete Duplicate Cases.

- Seed Generation.
- Test Case Construction.
- Test Case Execution. Parallel Case Execution / Environment Reset.
- Effective Case Checking
- Critical Path Extraction.

Result

CUPERFUZZER further extracted 30 critical paths from these discovered effective cases

TABLE IV: Discovered critical paths in our experiments.

No.	Effective Cases	Seed Mode	Critical Path†	Privilege Escalation (Granted Permissions)	Flaw
1	1,904	single-app dual-app	Installation [ACTIVITY_RECOGNITION, normal, NULL] → OS-update	ACTIVITY_RECOGNITION	**DS#3**
2	3	dual-app	Installation [com.test.cp, normal, NULL] → Installation [NULL, NULL, NULL] → Installation [com.test.cp, dangerous, NULL]	com.test.cp	**DS#1**
3	4	single-app dual-app	Installation [com.test.cp, normal, NULL] → Installation [com.test.cp, dangerous, NULL] → OS-update	com.test.cp	**DS#4**
4	92	dual-app	Installation [com.test.cp, normal, NULL] → Uninstallation → Installation [com.test.cp, dangerous, NULL]	com.test.cp	**DS#1**
5–15‡	44	single-app dual-app	Installation [com.test.cp, normal, {Group}] → Installation [com.test.cp, dangerous, {Group}] → OS-update	com.test.cp system permissions in {Group}	**DS#4**
16	4	single-app dual-app	Installation [com.test.cp, normal, UNDEFINED] → Installation [com.test.cp, dangerous, UNDEFINED] → OS-update	com.test.cp READ_CONTACTS ... (30 dangerous system permissions in total)	**DS#2**
17–27‡	304	dual-app	Installation [com.test.cp, normal, {Group}] → Uninstallation → Installation [com.test.cp, dangerous, {Group}]	com.test.cp system permissions in {Group}	**DS#1**
28	27	dual-app	Installation [com.test.cp, normal, UNDEFINED] → Uninstallation → Installation [com.test.cp, dangerous, UNDEFINED]	com.test.cp READ_CONTACTS ... (30 dangerous system permissions in total)	**DS#2**
29	1	dual-app	Installation [com.test.cp, normal, NULL] → OS-update → Installation [NULL, NULL, NULL] → Installation [com.test.cp, dangerous, NULL]	com.test.cp	**DS#1**
30	1	dual-app	Installation [com.test.cp, normal, NULL] → OS-update → Uninstallation → Installation [com.test.cp, dangerous, NULL]	com.test.cp	**DS#1**

identified four fatal design shortcomings lying in the Android permission framework

DESIGN SHORTCOMINGS AND ATTACKS

DS1: Dangling Custom Permission

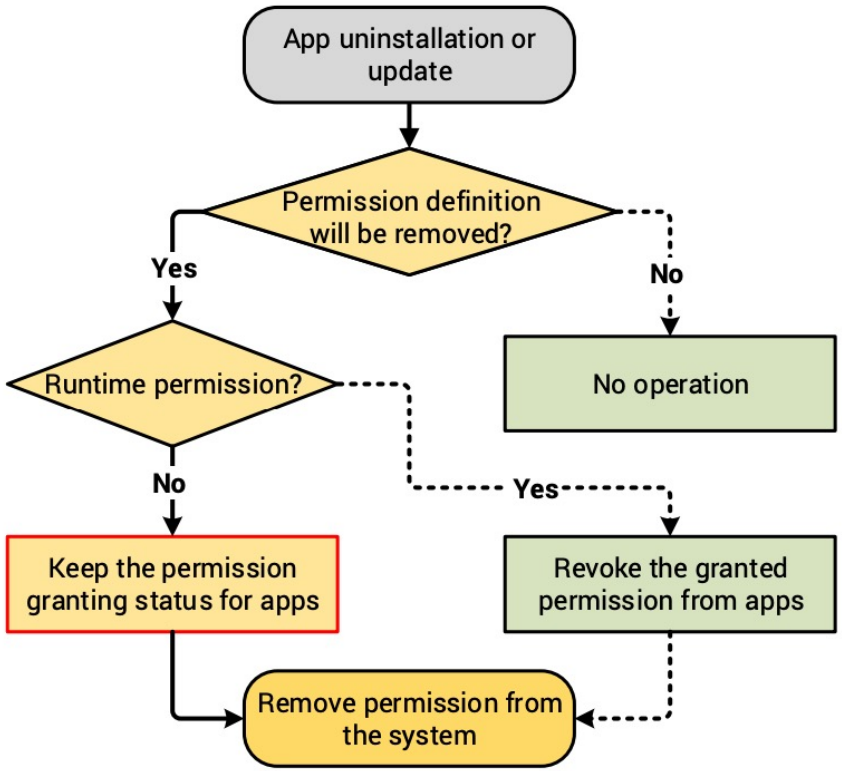


Fig. 4: Dangling custom permission.

DS#1: *If the removed custom permission is an install-time permission, the corresponding permission granting status of apps will be kept, causing dangling permission.*

The user installs app-ds1-d and app-ds1-r

↓

uninstall app-ds1-d and install the updated app-ds1-d

(PMS scans the package
and adds the updated custom permission com.test.cp into the system)

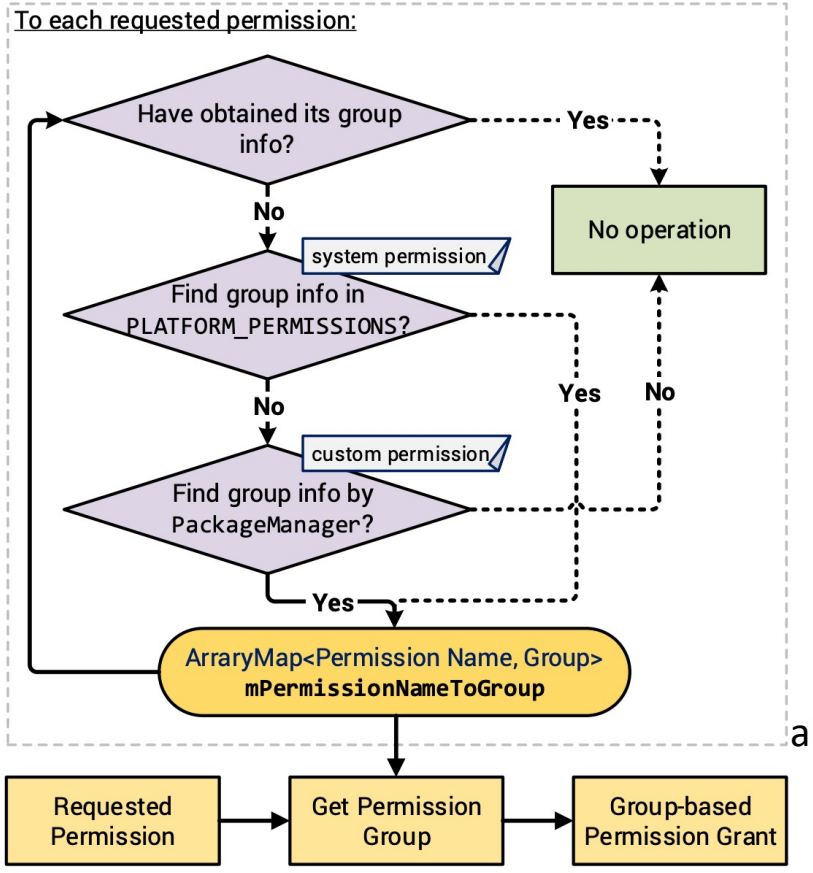
```
1 <permission
2 android:name="com.test.cp"
3 android:protectionLevel="dangerous"
4 android:permissionGroup="android.permission-
    group.PHONE"></permission>
```

↓

PMS adjusts the granting status of the existing apps' requested permissions

*app-ds1-r obtains the CALL_PHONE permission without user consent

DS2: Inconsistent Permission-Group Mapping



In Android, the grant of dangerous permissions is group-based!

-> the correct mapping relationship is quite critical in this process.

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

PLATFORM_PERMISSIONS

a hard-coded <system permission, system group>

PackageManager

relies on AndroidManifest.xml(custom permissions)

all dangerous system permissions are put into a “android.permission- group.UNDEFINED”

DS2: Inconsistent Permission-Group Mapping

```
<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"
    />
```

app-ds2

requests the WRITE_EXTERNAL_STORAGE permission

updated version of app-ds2

requests a dangerous custom permission com.test.cp

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

Listing 4: Mapping mPermissionNameToGroup.

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

Listing 5: Updated mapping mPermissionNameToGroup.

if one dangerous permission has been granted, the other dangerous permissions will be granted without user permitting

DS#3: Custom Permission Elevating

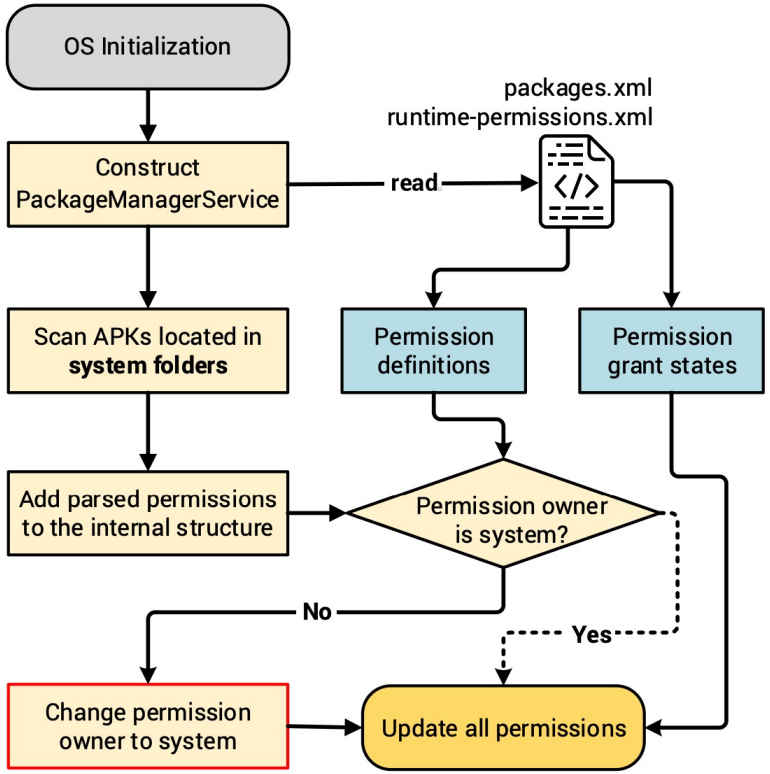
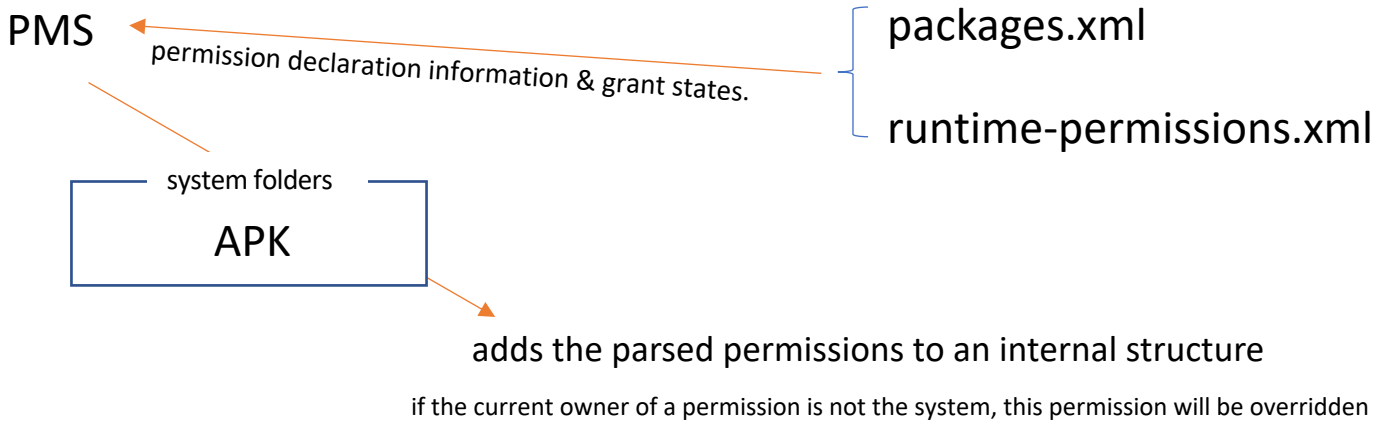


Fig. 6: Custom permission elevating.



DS#3: *When Android OS overrides a custom permission (changing the owner), the granting status of this permission is not revoked, further resulting in permission elevating.*

if an app has been granted with a custom permission with the same name as a system permission, this granted custom permission will be elevated to system permission after permission overriding.

DS#3: Custom Permission Elevating

Android 9 device, the adversary creates an app app-ds3,

```
1 <permission
2 android:name="android.permission.
      ACTIVITY_RECOGNITION"
3 android:protectionLevel= "normal"/>
4
5 <uses-permission android:name="android.
      permission.ACTIVITY_RECOGNITION" />
```

→ a new dangerous system permission introduced in Android 10

Listing 6: Define and request ACTIVITY_RECOGNITION.

OS update

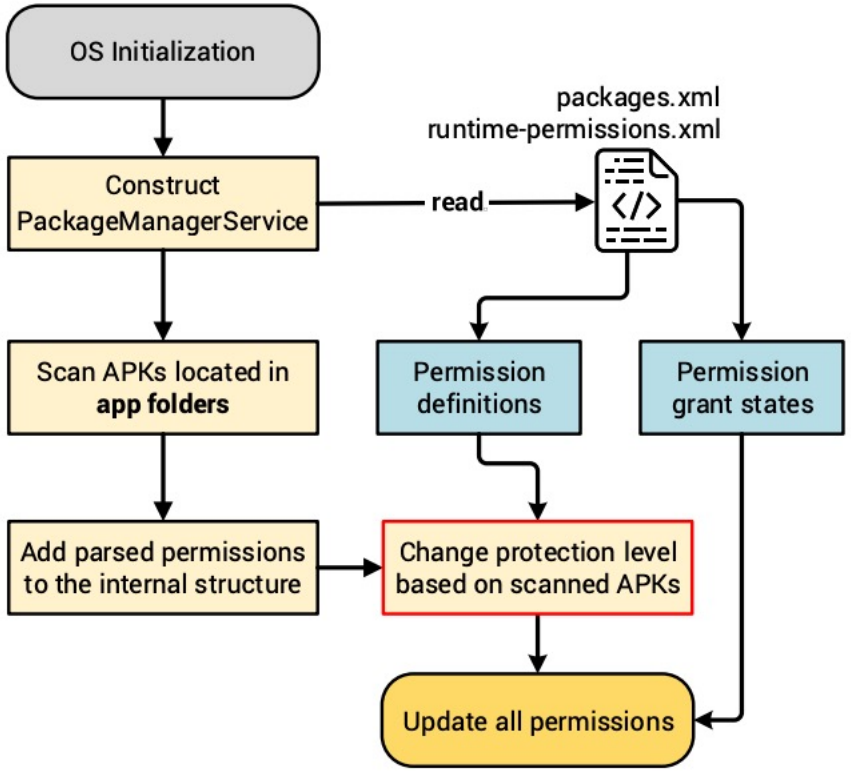
↓

OS initialization

↓

app-ds3 has been granted with the ACTIVITY_RECOGNITION permission (privilege escalation)

DS#4: Inconsistent Permission Definition



An app installation may also update an existing custom permission defined by itself

if the protection level is changed from normal or signature to dangerous, the system will keep its old protection level.
(to block the permission upgrade attack)

the permission definition held by the system != the permission definition provided by the owner app

app-ds4 that defines and requests a normal custom permission com.test.cp.

updated version of app-ds4 which changes the protection level of com.test.cp to dangerous and puts com.test.cp into the PHONE group. And also requests the CALL_PHONE permission

reboots the phone

-> app-ds4 obtains com.test.cp (dangerous custom permission) automatically.
Then it can obtain the CALL_PHONE permission without user consent

Limitation

- **Attacks in Practice** Some attacks need user interactions more than once.

  Ex. if an adversary wants to exploit DS#1,
  she needs to prepare two malicious apps and induce a victim user to re-install an app after uninstalling it

- **Test Case Generation**

  CUPERFUZZER needs to generate massive test cases for fuzzing

  To improve the effectiveness of vulnerability discovery,
  we could deploy some feedback mechanism to generate more interesting test cases

  a feedback mechanism may result in generating too many similar test cases
  which are duplicate from the view of critical paths.
  Thus, it needs to trade off the diversity against the effectiveness of test cases.

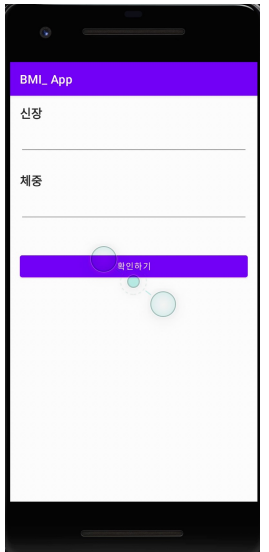
https://sites.google.com/view/custom-permission

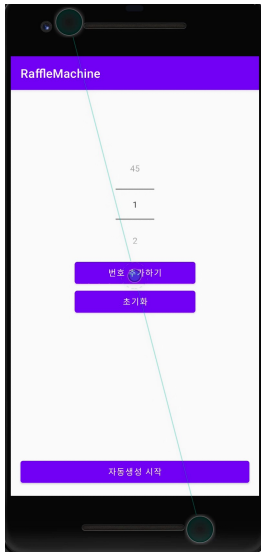
Android-java

- thread, animation
- Networking
  (Socket, Web Request)
- Database

Android-kotlin

- Learning Kotlin Grammar
- Simple Project
  - BMI Calculator
  - Calculator App
  - Lotto Number Drawer





Decompile

1. 내부 파일을 확인만 해볼 경우
apk -> jar 파일 변환
jar 파일 확인

2. 파일을 수정하고 앱 재빌드 하는 경우
Apk 압축 풀기
dex -> jar 파일 변환
jar 파일 확인
파일 수정
다시 빌드하기
서명 및 설치

- ApkTool
- Dex2Jar
- JD-GUI

Plans

- Kotlin programming (small project)
- Reversing analysis
- Audio Preprocessing vulnerabilities
- Reading paper