안드로이드 보안

취약점 분석과 보안 아키텍처 확장

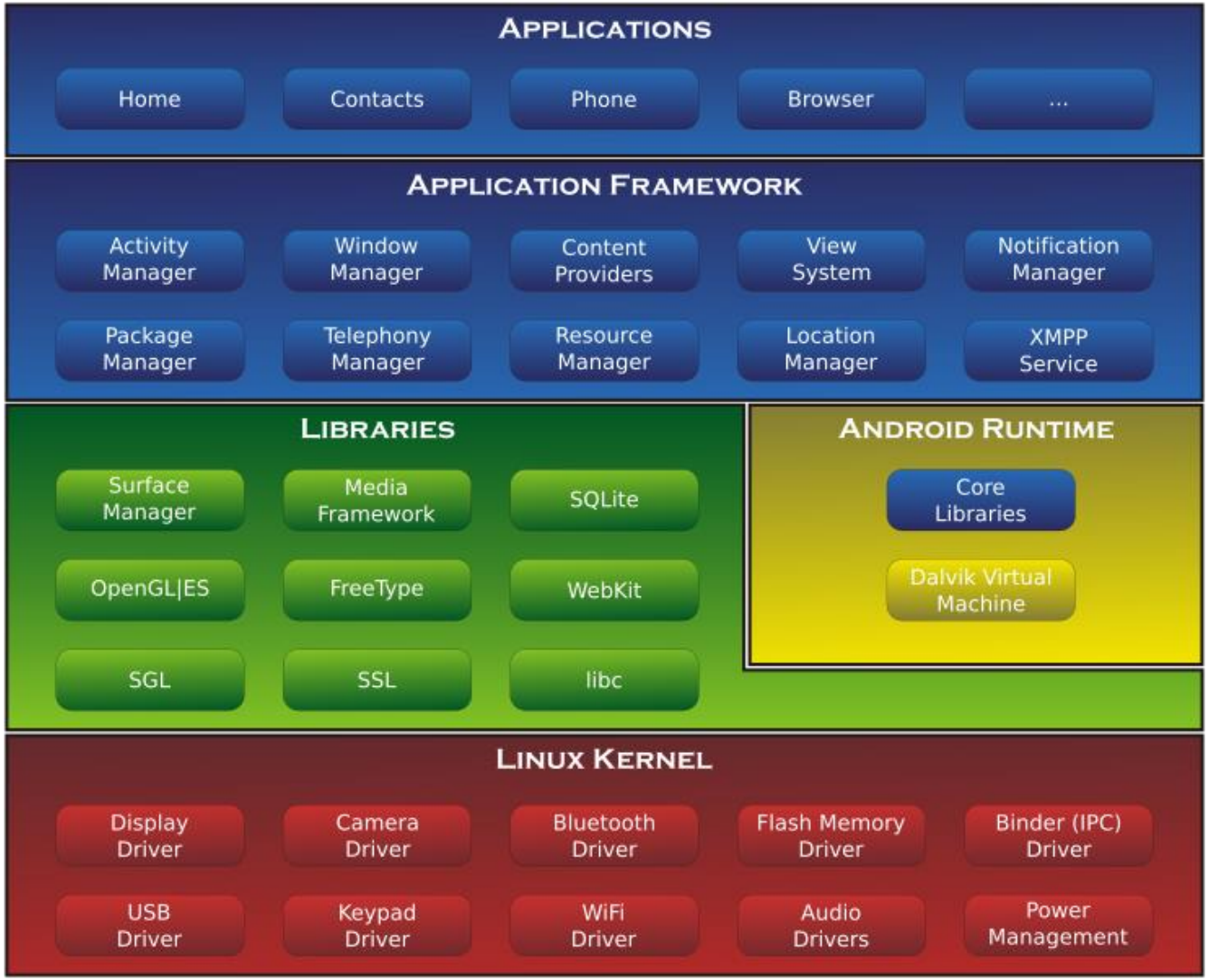
신인식

Android Security Architecture

Android Permission & Linux GID

Android Architecture

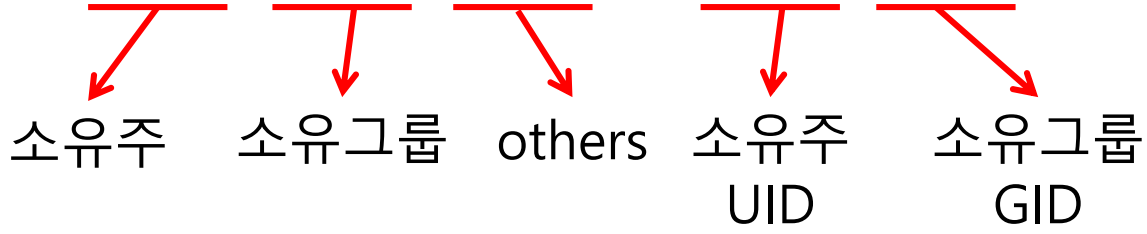
- Android는 linux 기반의 모바일 OS로, 내부적으로 linux의 access control 메커니즘을 활용하고 있음



Linux의 access control 메커니즘

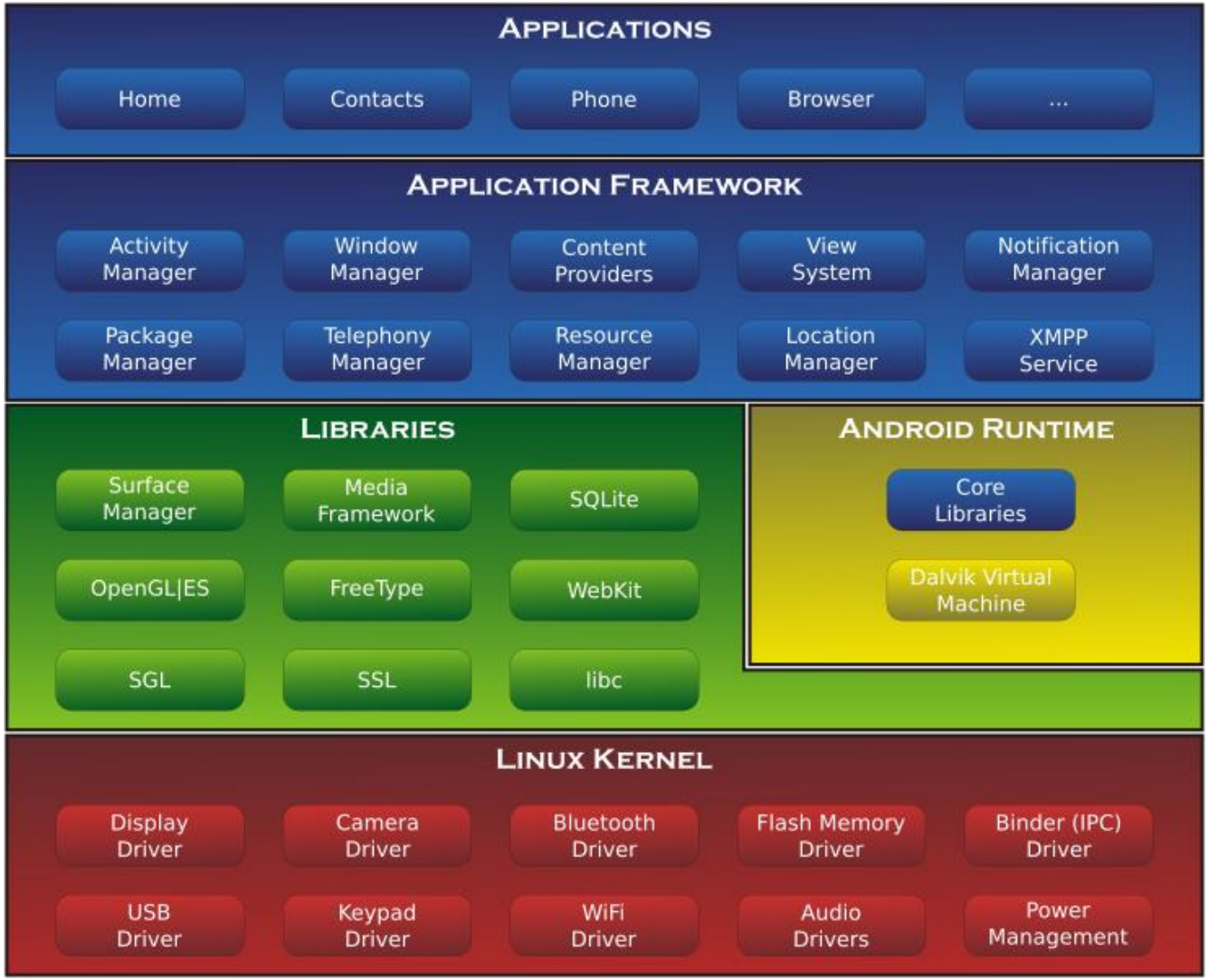
- Linux에서는 user 식별자인 **UID**와 user가 속해 있는 group 식별자인 **GID**를 기반으로 user가 접근할 수 있는 파일이나 디렉토리 자원들에 대한 access control을 수행함

- 파일을 생성한 user가 해당 파일의 소유주가 되며, user는 소유주, 소유그룹, others에 대한 권한을 설정함으로써, 다른 user들에 대한 파일 접근을 제어할 수 있음
  - ex) - r w x r – x r - - 1 root root 1024 Feb 8 13:01 bin/



Android Architecture

- Android는 linux 기반의 모바일 OS로, 내부적으로 linux의 access control 메커니즘을 활용하고 있음
  - 각 앱은 고유한 UID를 지님으로써, 다른 앱들로부터 각자의 파일 자원들을 보호함
  - 동일한 GID를 지니는 다른 앱들에게는 자원을 공유하기도 함



Android permission

- 하지만 Android에서 Linux UID / GID를 직접 활용하기에는 어려움이 존재
  - Android에서는 파일 외에도 다양한 자원들이 존재하고, 이들에 대한 access control이 필요
    - ex) App의 service, activity
  - 하나의 자원 사용을 위해 다수 개의 GID가 필요한 경우, 이에 대한 매핑이 필요

- 이를 위해 Android에서는 한 단계 더 추상화된 컨셉으로 permission을 정의함

Android permission

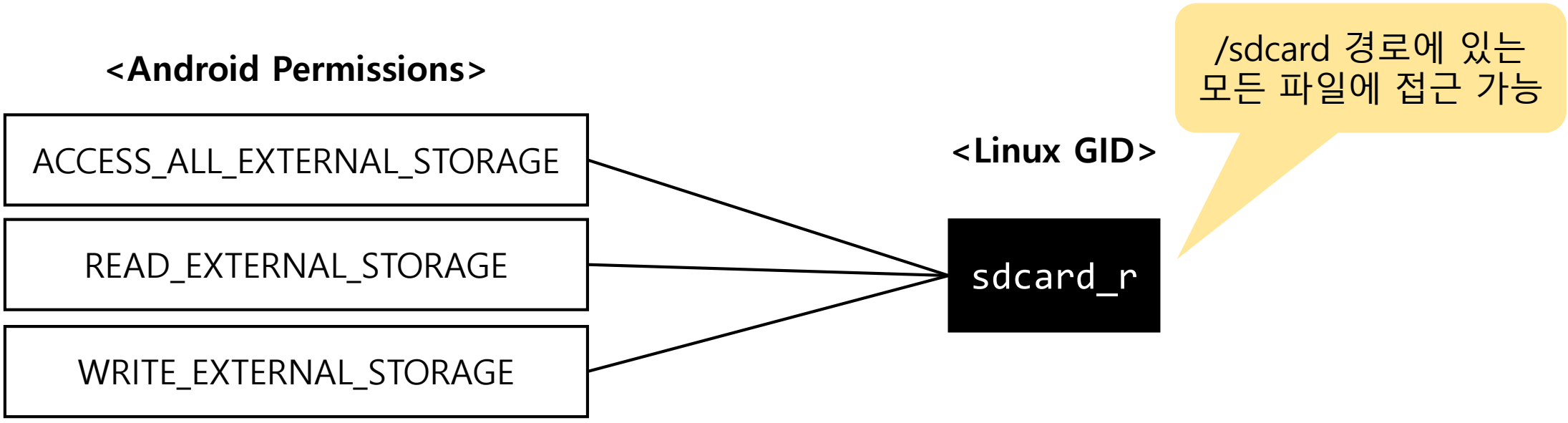
- Android는 여러 resource에 대한 permission들을 정의하고 있으며, 적절한 permission을 지닌 앱만이 해당 resource에 접근할 수 있도록 설계되어 있음
  - System permission: Android 플랫폼에서 정의된 permission으로, 시스템 자원들에 대한 access control을 담당
    - ex) 카메라, 외부 저장소, SMS 메시지 등에 대한 접근
  - Custom permission: Third-party 앱들이 정의한 permission으로, 앱 자원들에 대한 access control을 담당
    - ex) 앱의 Service, Content provider 등에 대한 접근

Android Permission List

READ_CALENDAR	RECORD_AUDIO	READ_EXTERNAL_STORAGE
WRITE_CALENDAR	READ_PHONE_STATE	WRITE_EXTERNAL_STORAGE
CAMERA	CALL_PHONE	ACCESS_WIFI_STATE
READ_CONTACTS	BODY_SENSORS	CHANGE_NETWORK_STATE
WRITE_CONTACTS	SEND_SMS	INTERNET
GET_ACCOUNTS	RECEIVE_SMS	NFC
ACCESS_FINE_LOCATION	READ_SMS	SET_TIME
•••	•••	•••

Linux GID mapping

- Android permission들은 Linux Group ID (GID)와 매핑되어 있음
  - 앱이 GID에 매핑되어 있는 permission들 중 하나라도 얻으면 해당 GID를 가질 수 있음
  - 이때, 앱은 해당 GID로만 접근할 수 있는 파일들을 read/write할 수 있게 됨



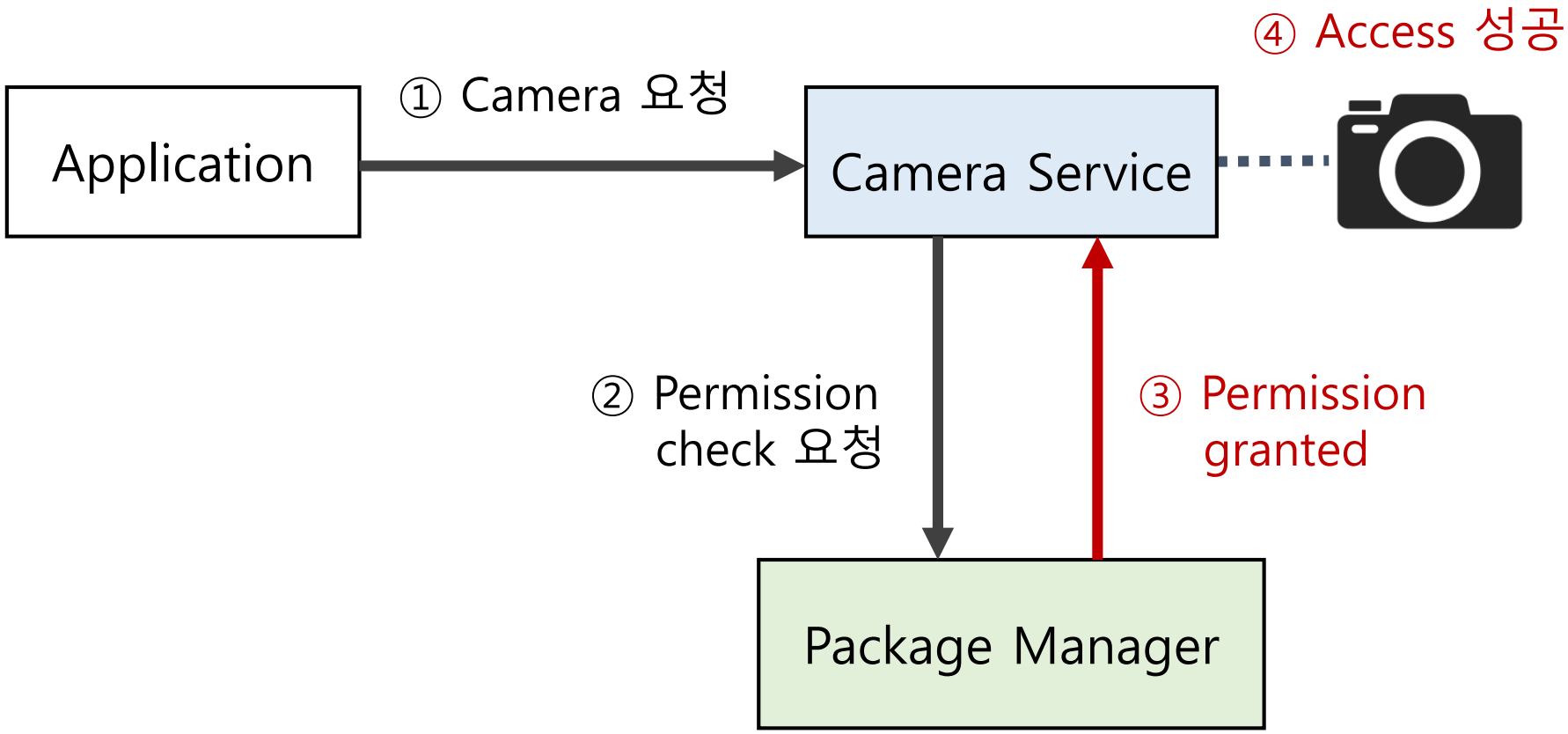
Permission management

- 앱이 요구하는 permission 정보들은 Manifest 파일에 선언되어 있으며, 이 정보들은 앱이 설치될 때 Package Manager (PM)에 저장되고, 관리됨
- 이때, 각 permission마다 다른 protection level을 지니게 함으로써 앱이 요구할 수 있는 permission 범위에 제한을 둠

Protection Level	Description to obtain the Permission
Normal	사용자의 명확한 동의 불필요
Dangerous	사용자의 명확한 동의 요구됨
Signature / SystemOrSignature	해당 permission을 정의한 주체와 동일한 인증서로 서명되어야 함

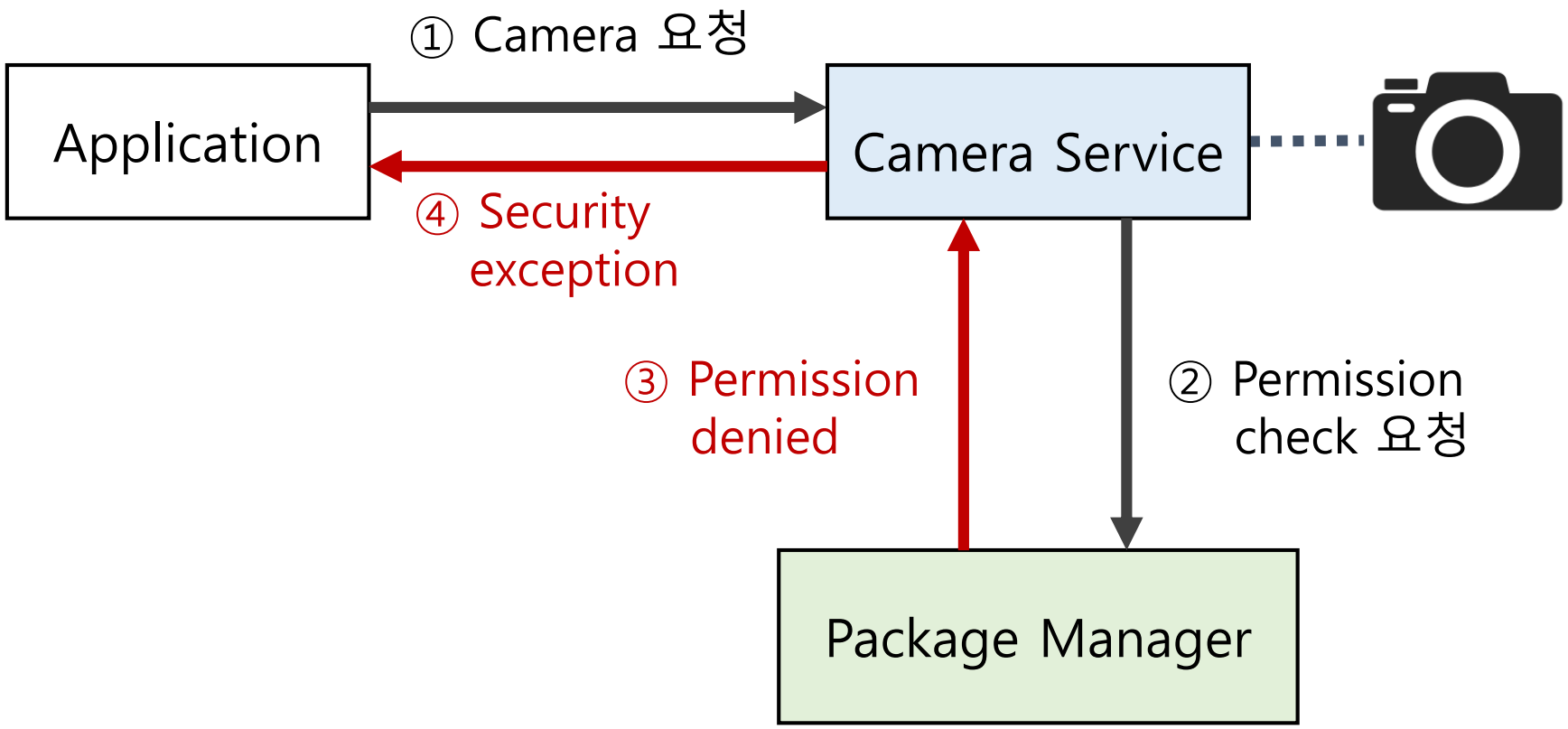
Permission checking

- Android는 앱이 각 resource에 접근할 때마다 PM을 통해 해당 앱의 permission을 확인함
  - 적절한 permission이 있다면 해당 resource에 대한 접근을 허용



Permission checking

- Android는 앱이 각 resource에 접근할 때마다 PM을 통해 해당 앱의 permission을 확인함
  - 적절한 permission이 없다면 security exception을 일으키고, 해당 접근을 막음



SELinux (Security-Enhanced Linux)

- Android는 4.3 버전부터 SELinux를 적용하여, 더욱더 세분화된 access control 제공
- SELinux vs. Linux UID / GID system

		Linux UID / GID	SELinux
Access control 방법		Discretionary Access Control (DAC) Owner에게 자유재량권 부여	**Mandatory Access Control (MAC)** 미리 정해진 rule에 따라, 오직 필요한 기능에 대해서만 사용 권한 부여
Access control 단위	주체	User	**Role** 예: shell, init, kernel, mediaserver, system_server, appdomain 등
	객체	File / directory	**Type - Class** 예: Kernel type의 file, dir, process, security, socket class
Access permission 종류		Read / write / execute	모든 종류의 operation 예: read / … / create / open / listen / lock

SELinux (Security-Enhanced Linux)

- SELinux vs. Linux UID / GID system (계속)
  - 악성 앱이 악의적으로 root (super user) UID 획득 (루팅)에 성공한 경우,
    - **Linux UID / GID**: super user이므로, 아무런 제한 없이 다른 user의 data에 접근 가능

    - **SELinux**: super user이더라도, OS에서 type에 따라 access control
      - User 임의로 type를 변경하는 것은 불가능

SELinux policy management

- SELinux policy는 source (주체)와 target (객체)에 관한 rule들로 구성됨
  - SELinux policy rule 포맷

    allow [SOURCE_TYPE] [TARGET_TYPE]:[CLASS] [PERMISSION];
    - 예: allow shell shell_data_file:file {ioctl read write create getattr setattr lock append … }
  - 특정 source와 target간 rule이 정의되어 있지 않은 경우, 접근이 허용되지 않은 것으로 간주 (whitelist 방식)

- SELinux policy rule들은 Android 프레임워크와 함께 빌드됨
  - 제조사들은 device-specific rule들을 추가하기 위해, 해당 rule들을 포함하고 있는 .te파일들을 프레임워크와 같이 빌드함
  - Runtime에 사용자가 임의로 rule을 추가하는 것이 불가능

SELinux policy management

- SELinux policy는 적용되는 방법에 따라, 두 가지 모드로 구분됨
  - Enforcing 모드
    - SELinux policy에 따른 access control이 적용되며, 이에 대한 log도 같이 남김
  - Permissive 모드
    - SELinux policy에 따른 access control이 적용되지 않지만,
      policy에 어긋난 operation이 수행된 경우 log를 남김
  - Android는 5.0 버전부터 enforcing mode를 기본 모드로 사용하여 보안을 강화함
    - Enforcing -> permissive 모드로의 전환 또한 일반 사용자 (root 사용자 포함)에게는 제한됨

Android 스마트폰 보안 취약점 분석
Permission-GID 매핑

목표

- Motivation
  - 각 스마트폰 제조사들은 자신들의 제품에 맞게, 많은 부분의 안드로이드 소스를 수정해서 사용함
  - 이 과정에서 다음과 같은 이유로 여러 보안 취약점이 발생할 수 있음
    - 안드로이드는 그 구조가 매우 복잡함
    - 짧은 개발 기간 (약 6개월 미만) 안에 안드로이드를 customizing함
  - 따라서 제조사 기기 (custom) 프레임워크와 AOSP 프레임워크 간의 비교를 통해 제조사 기기에 잠재되어 있는 보안 취약점을 발견할 수 있음

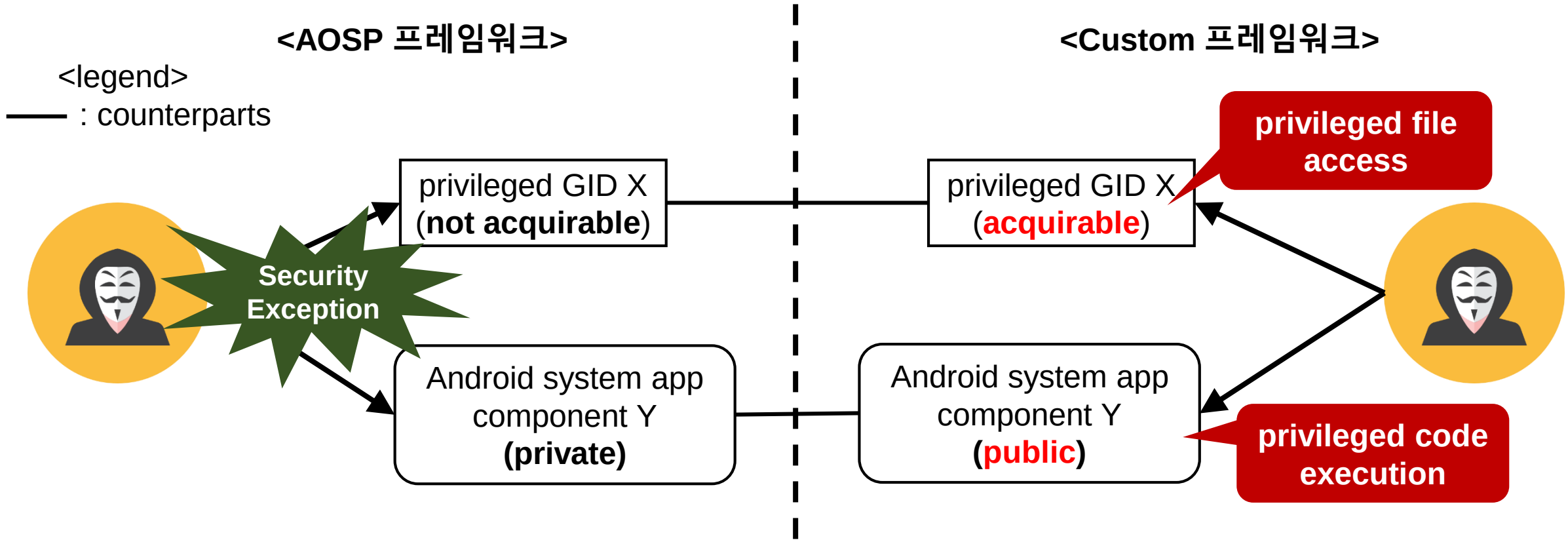
- **목표**
  - **ANDDIFF**: 프레임워크 비교 분석 도구 설계 및 개발
    - AOSP와의 비교를 통해 custom 프레임워크에 잠재되어 있는 보안 취약점을 자동으로 발견 및 분석

타겟 보안 취약점들

- ANDDIFF는 크게 두 가지 타입의 보안 취약점들을 발견할 수 있음

  V1. 이전에 얻을 수 없었던 **privileged Linux GID**를 custom 프레임워크에서 얻을 수 있는 취약점

  V2. 이전에 접근할 수 없었던 **App component**에 custom 프레임워크에서는 접근할 수 있는 취약점



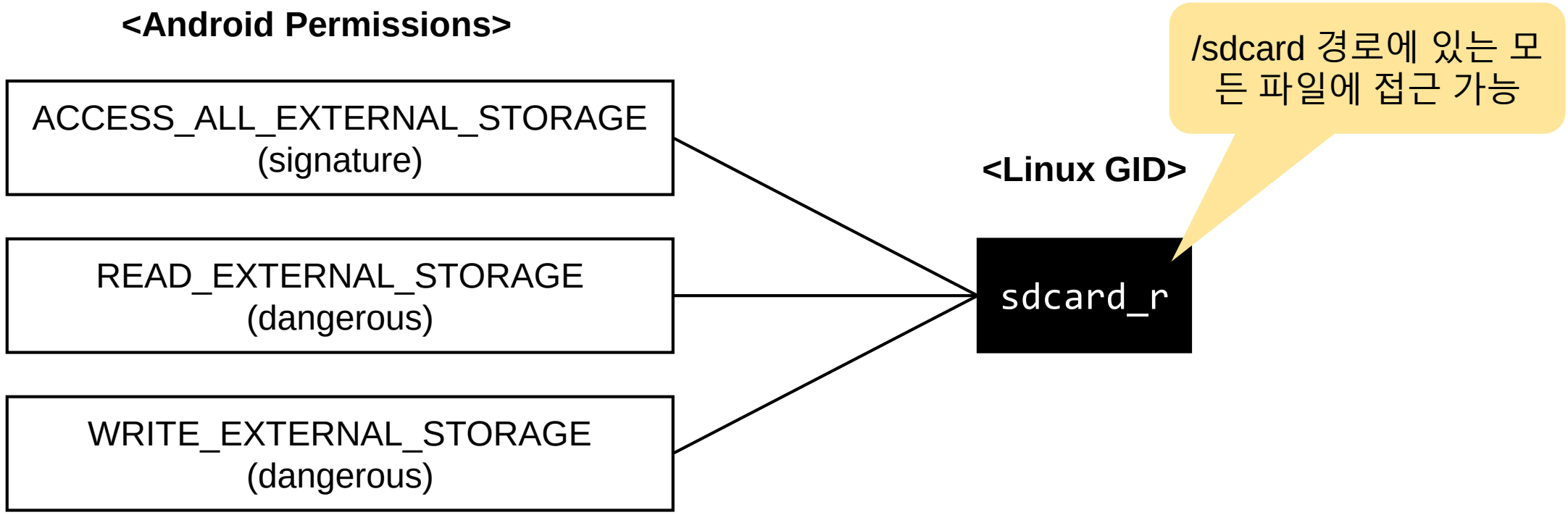
V1. 보안 수준이 약해진 privileged GID 분석 방법 (1/4)

- Android permission의 protection level
  - Permission마다 각각 다른 protection level을 지님

Protection Level	Description to Obtain the Permission
Normal	사용자의 명확한 동의 불필요
Dangerous	사용자의 명확한 동의 요구됨
Signature, SystemOrSignature	앱은 퍼미션을 정의한 앱과 동일한 인증서로 서명되어야 함

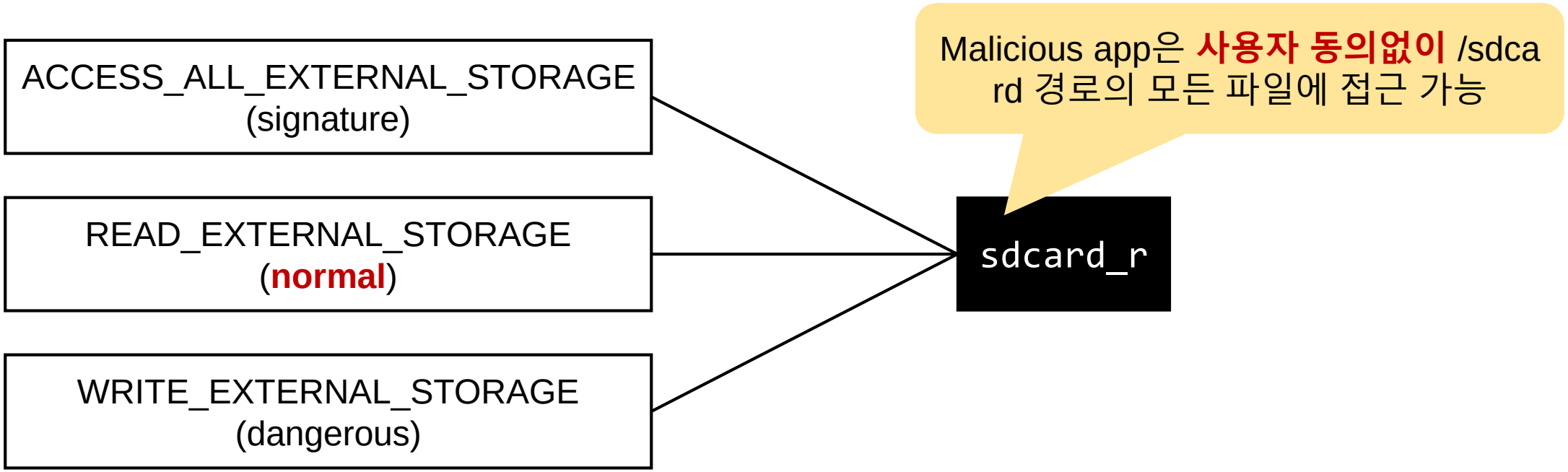
V1. 보안 수준이 약해진 privileged GID 분석 방법 (2/4)

- Android permission과 Linux Group ID (GID) 사이의 매핑
  - 앱이 Linux GID에 매핑되어 있는 permission들 중 하나라도 얻으면 해당 GID를 가질 수 있음
  - 어떤 GID를 가지면 해당 GID로만 접근할 수 있는 파일들을 read/write할 수 있음



V1. 보안 수준이 약해진 privileged GID 분석 방법 (3/4)

- 발생 가능한 보안 취약점
  - customizing 과정에서 특정 GID를 얻을 수 있는 permission들의 protection level이 이전보다 낮게 설정된다면 이로 인해 보안 취약점이 생길 수 있음
  - e.g., READ_EXTERNAL_STORAGE의 protection level을 dangerous에서 normal로 낮춘 경우



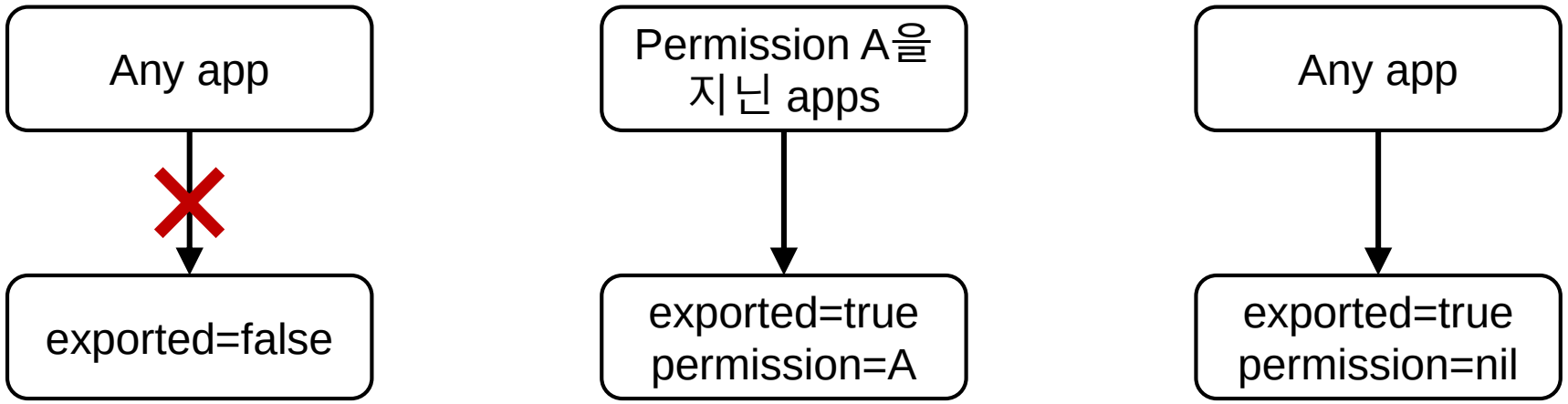
V1. 보안 수준이 약해진 privileged GID 분석 방법 (4/4)

- 분석 목표
  - protection level이 낮아진 permission을 탐색하고, 해당 permission으로 얻을 수 있는 GID 파악
  - GID를 통해 접근할 수 있는 파일 목록 파악

- 분석 방법
  - 이를 위해 프레임워크 이미지에 포함된 설정 파일들 (.xml)을 분석
    - /etc/permissions/platform.xml
      - Android permission과 Linux GID 사이의 매핑 관계가 정의되어 있음
    - /framework/base/core/res/AndroidManifest.xml
      - 각 Android permission이 정의되어 있으며, 어떤 protection level로 설정되어 있는지 알 수 있음
  - File system의 경로 별 group permission 목록 추출

V2. 보안 수준이 약해진 앱 컴포넌트 분석 방법 (1/5)

- Android component의 보안 설정 방식
  - 안드로이드 컴포넌트의 종류: Activity, Service, ContentProvider, BroadcastReceiver
  - 다음과 같은 보안 설정을 지님
    - exported, permission, readPermission/writePermission



V2. 보안 수준이 약해진 앱 컴포넌트 분석 방법 (2/5)

- 발생 가능한 보안 취약점
  - 앱 컴포넌트의 보안 설정이 AOSP보다 다르게 설정되어 있는 경우, 이로 인해 보안 취약점이 생길 수 있음
  - e.g., exported 값이 false에서 true로 변경된 경우, 외부 malicious app이 해당 컴포넌트에 쉽게 접근하여, 특별한 권한 없이도 컴포넌트의 코드를 실행시킬 수 있음

V2. 보안 수준이 약해진 앱 컴포넌트 분석 방법 (3/5)

- 분석 목표
  - AOSP 프레임워크와의 비교를 통해 보안 설정이 이전보다 낮게 변경된 컴포넌트들 파악
    - 이때, AOSP 프로임워크와 custom 프레임워크에서 동일한 이름을 지닌 컴포넌트들을 매핑하여 비교 분석
    - e.g., AOSP의 ClockProvider와 custom의 ClockProvider의 보안 설정 비교

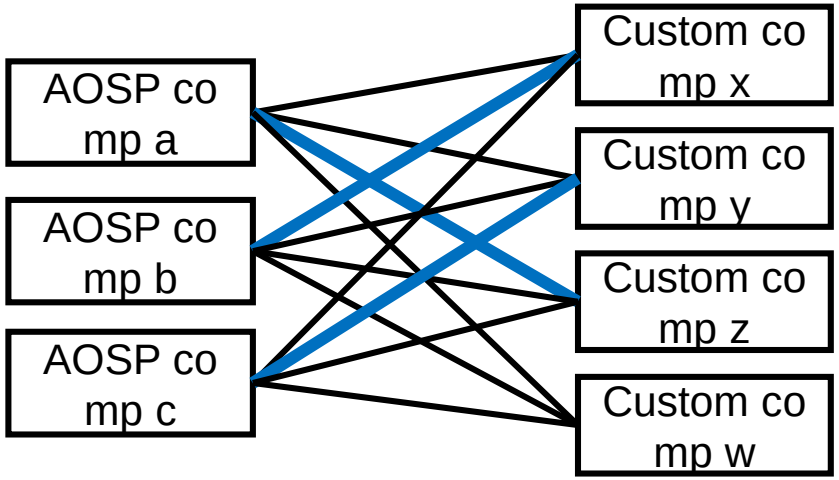
- Challenge
  - Customizing 과정에서 많은 컴포넌트들의 이름이 변경될 수 있음
    - e.g., AOSP의 ClockProvider가 custom에서는 AlarmProvider로 개명될 수 있음
  - 이 경우, **개명된 컴포넌트들까지 추적하여 AOSP 프레임워크와 비교 분석이 가능해야 함**

V2. 보안 수준이 약해진 앱 컴포넌트 분석 방법 (4/5)

- 분석 방법
  1. 컴포넌트의 이름이 바뀌지 않은 경우
     - 해당 앱의 manifest 파일 (AndroidManifest.xml) 분석
       - 각 컴포넌트의 보안 설정이 정의되어 있음

V2. 보안 수준이 약해진 앱 컴포넌트 분석 방법 (5/5)

- 분석 방법
  2. 컴포넌트의 이름이 바뀐 경우
     - AOSP 프레임워크에서 동일한 컴포넌트를 찾기 위해 컴포넌트들의 코드 유사도 비교
       - 만약 두 컴포넌트들 간의 코드가 비슷하다면 두 컴포넌트들이 동일할 가능성이 높다고 가정함
     - 최대 이분 매칭 알고리즘을 사용하여 최대 유사도를 지닌 컴포넌트 매핑을 찾음
     - 이후 매핑된 컴포넌트들에 대해 manifest 파일의 보안 설정 분석



	x	y	z	w
a	50	70	100	90
b	95	40	30	60
c	10	70	10	10

행렬로 표현한 각 노드 사이의 가중치(유사도)

V1. 보안 수준이 약해진 privileged GID 결과 (1/3)

	ZTE Kis3 max	ZTE Blade G	THL W200
Protection level이 낮아진 Permission	ACCESS_MTK_MMHW	ACCESS_LOCATION_API	ACCESS_MTK_MMHW
획득 가능한 GID	• system • media • camera	• system • qcom_oncrpc • net_raw • qcom_diag • gps	• media • camera

V1. 보안 수준이 약해진 privileged GID 결과 (2/3)

- 결과 분석
  - **system GID (**ZTE Kis3 max & ZTE Blade G)
    - ZTE Kis3 max
      - /data/system/locksettings.db --> 잠금화면 패턴인증 초기화
      - /data/system/packages.xml --> 안드로이드 퍼미션 획득 및 악성앱 신분 위조
      - /dev/block/mmcblk0 --> 플래시 메모리 (eMMC) 내의 모든 정보 유출
      - /dev/block/mmcblk0boot0 --> 플래시 메모리 (eMMC) 내의 모든 정보 유출
    - ZTE Blade G
      - ACCESS_LOCATION_API에 대한 실제 정의 부분이 존재하지 않아서 system GID를 획득할 수 없음
      - 또한 ACCESS_LOCATION_API에 매핑되어 있는 다른 GID들 역시 획득할 수 없음
        - qcom_oncrpc, net_raw, qcom_diag, gps

V2. 보안 수준이 약해진 앱 컴포넌트 결과

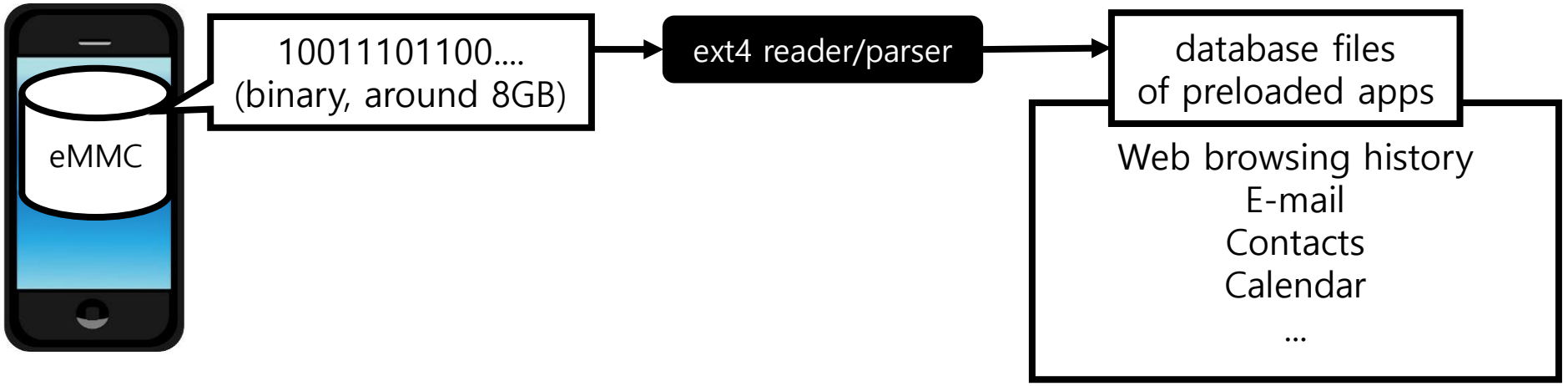
		ZTE Kis3 max	ZTE Blade G	THL W200
개명되지 않은 컴포넌트	Activity	-	-	-
	Service	4	2	-
	ContentProvider	1	3	2
	BroadcastReceiver	-	2	-
개명된 컴포넌트	Activity	-	-	-
	Service	-	1	20
	ContentProvider	1	2	3
	BroadcastReceiver	-	2	15
총 합		6	12	35

다섯 범주의 공격 방식과 공격 예제

공격 방식	예시 공격
A1. 플래시 메모리의 정보 유출	안드로이드 내부 저장소(`/data/data/*`)에 있는 시스템 데이터베이스 파일 확인
A2. 안드로이드 퍼미션 획득	`signature` protectionLevel의 `REBOOT` 시스템 퍼미션 획득
A3. 악성 앱 신분 위조	악성 앱의 유저 ID를 크롬(Chrome)의 것으로 위조, 크롬 데이터에 접근
A4. 잠금화면패턴인증 초기화	PIN, 패턴인증 등의 사용자 인증 방식 무력화
A5. 스마트폰 모듈 제어	WiFi 모듈을 비활성화하는 서비스 거부 공격

<Demo 1> A1. 플래시 저장소 내의 모든 정보 유출

- 악성 앱이 ACCESS_MTK_MMHW 퍼미션을 요구하여 system GID를 획득
- system GID가 있으면 플래시 메모리를 관리하는 디바이스 드라이버에 접근 가능하며, 이를 통해 플래시 메모리에 저장된 사용자의 다양한 개인 정보를 획득할 수 있음
  - 웹 브라우저의 쿠키와 히스토리
  - Email 클라이언트 앱이 다운로드한 이메일들의 내용
  - 핸드폰에 있는 연락처



<Demo 3> 잠금 화면 패턴 초기화

- 악성 앱이 ACCESS_MTK_MMHW 퍼미션을 요구하여 system GID를 획득
- system GID가 있으면 잠금 화면 패턴을 저장해 놓은 파일인 /data/system/locksettings.db에 접근 가능
- 이 파일을 삭제하면 잠금 화면 패턴을 초기화할 수 있음

Mobile Platform Security

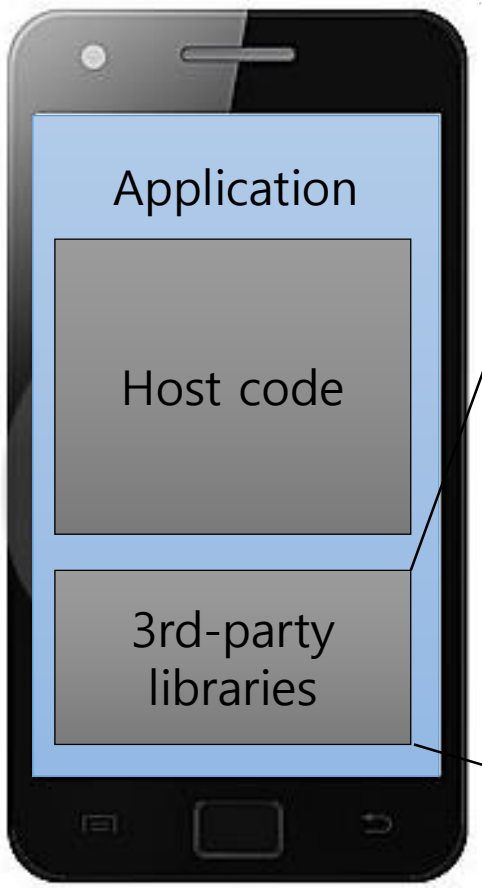
FlexDroid: Enforcing In-app Privilege Separation

(NDSS 2016)

- **Jaebaek Seo***, Daehyeok Kim*, Donghyun Cho*, Taesoo Kim†, Insik shin*
- * KAIST † Georgia Institute of Technology




3rd-party libraries become popular on Android



Ad, Analytics, Game engine, Billing, Social

admob

FLURRY

unity

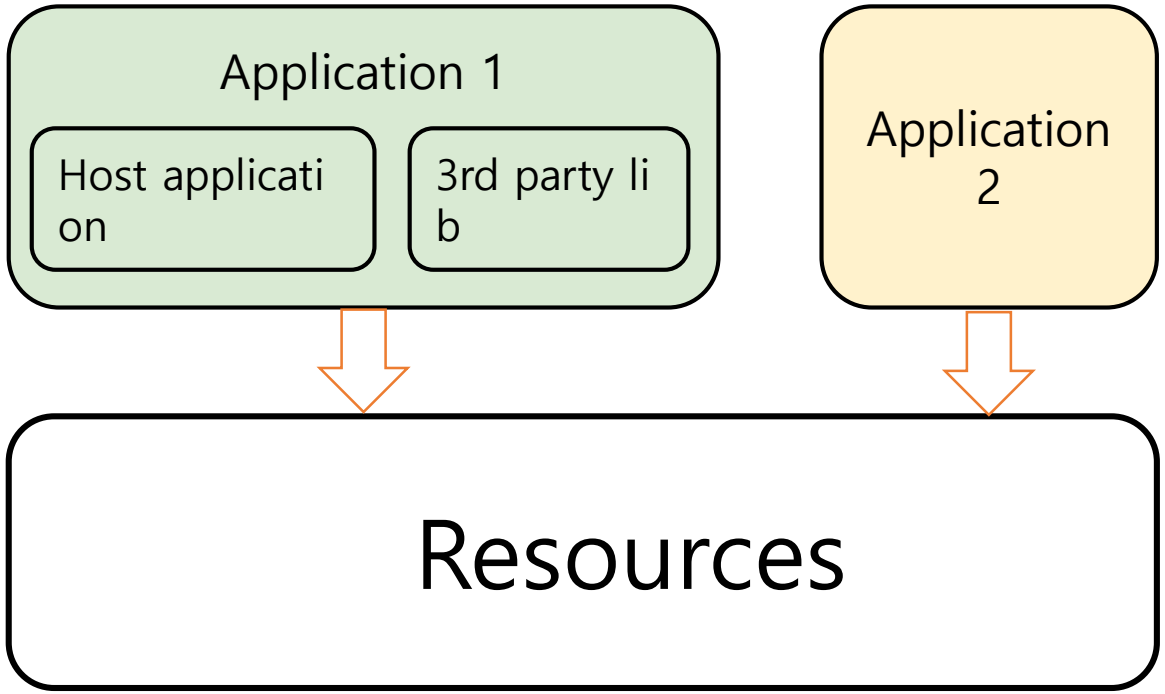
Parse
The Cloud Application Platform

PayPal

facebook

Can we trust them?

Unit of Trust on Android



- The unit of trust in Android is an app
- All components including third-party libraries in an app have the same permissions to acces s resources

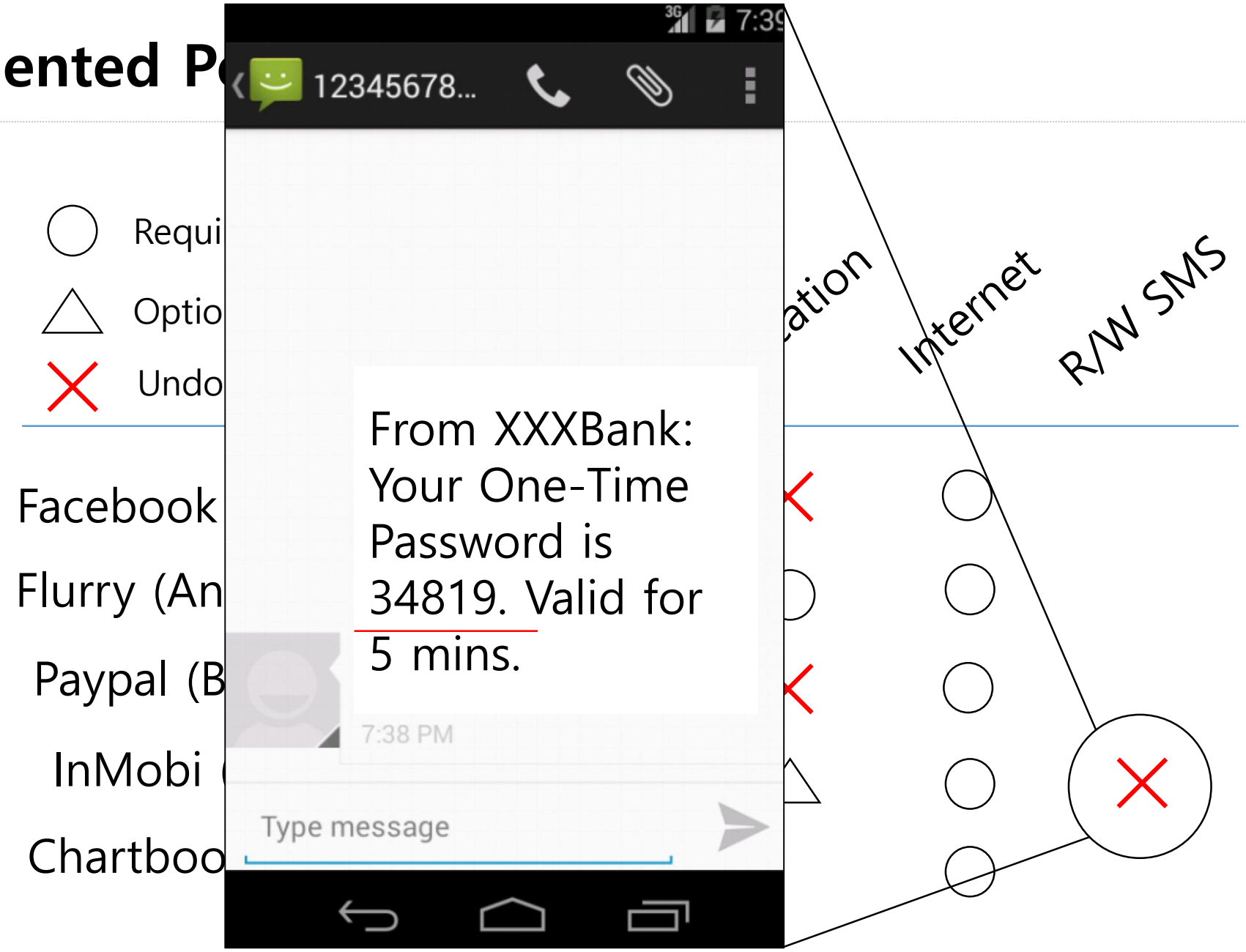
Over-privileged Third-party Libraries

Permissions used by popular third-party libraries

Name	Category	Accounts	Phone information	Internet	Read SMS	Write SMS	Read Calender	Write Calender	Write Settings	Get Tasks	Read Bookmark	Record Audio	Location
Facebook	Social	·	×	O	·	·	·	·	×	·	·	·	×
Flurry	Analytics	·	×	O	·	·	·	·	·	·	·	·	O
RevMob	Advertising	×	△	O	·	·	·	·	·	·	·	·	·
Chartboost	Advertising	·	×	O	·	·	·	·	·	·	·	·	·
InMobi	Advertising	·	·	O	×	×	△	△	·	·	·	·	△
Millennialmedia	Advertising	·	·	O	·	·	·	·	·	·	·	O	×
Paypal	Billing	·	×	O	·	·	·	·	·	·	·	·	×
Umeng	Analytics	·	O	O	·	·	·	·	×	×	·	·	×
AppLovin	Advertising	△	O	O	·	·	·	·	·	·	·	·	·
Pushwoosh	Notification	·	O	O	·	·	·	·	·	·	·	·	×
Tapjoy	Advertising	·	O	O	·	·	×	×	·	·	·	·	×
AppFlood	Advertising	·	△	O	·	·	·	·	·	·	·	·	△
OpenFeint	Social	O	O	O	·	·	·	·	·	·	·	·	×
Airpush	Advertising	×	△	O	·	·	·	·	·	·	×	·	×
Youmi	Advertising	·	O	O	·	·	·	·	·	×	·	·	×
Cauly	Advertising	·	·	O	·	·	·	·	·	×	·	·	△
Socialize	Social	·	△	O	·	·	·	·	·	·	·	·	△
Domob	Advertising	·	O	O	·	·	·	·	·	·	·	·	×
Leadbolt	Advertising	×	△	O	·	·	×	×	·	·	·	·	△
MobFox	Advertising	·	×	O	·	·	·	·	·	·	·	·	△

◯ Required △ Optional ╳ Undocumented

Undocumented P



Threat Model

- **Third-party libraries are potentially malicious**
  - Their code and logic are not directly visible to app developers (e.g., obfuscated)
  - Can use dynamic features of the Java language (e.g., JNI, reflect ion, multi-threading)
- **App developers explicitly know what third-party libraries a re for**
  - Given high-level functional description, app developers should be able to adjust the manifest and seamlessly integrate them w ithout compromising usability

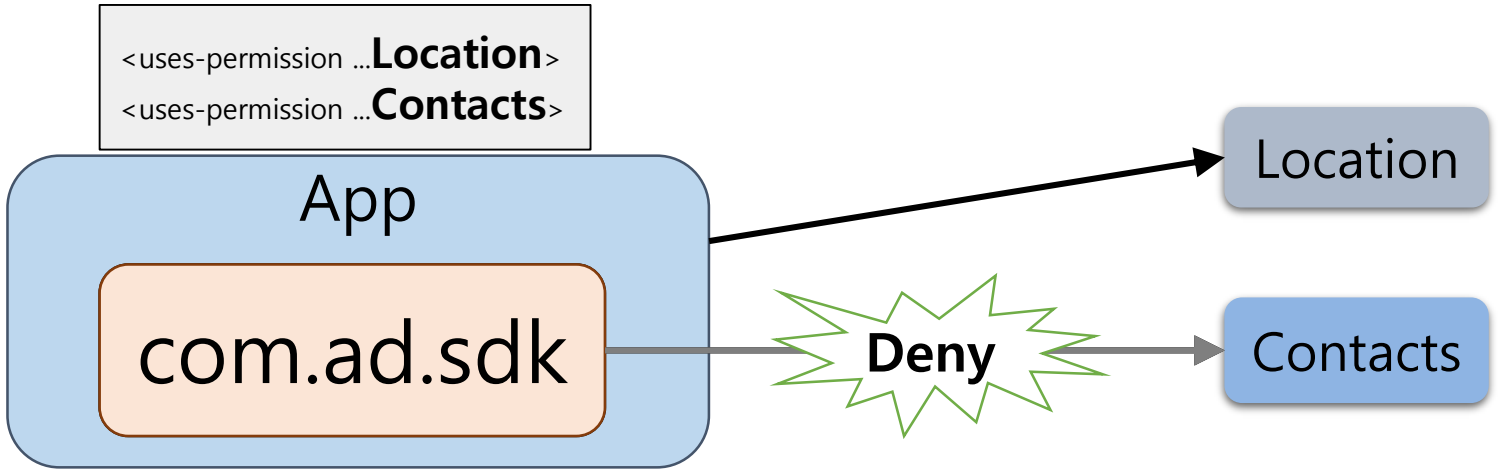
Goal

Separating the privilege of third-party library from the privil ege of its host application

- Preventing third-party libraries from accessing resources out of its pr ivilege

Overview of FLEXDROID

Specifying the package name and its permissions in ***AndroidManifest.xml***




```
<flexdroid android:name="com.ad.sdk" >
        <allow ...Location>
</flexdroid>
```

Challenges

- Control-flow and data dependency
  - Between host application and third-party libraries
- Dynamic runtime behavior
  - JNI, reflection, multi-threading

Technique	Out of 295 libs
Class Loading	27.9%
Reflection	49.6%
Class Inheritance	71.5%
JNI	17.1%

Java language techniques used in third-party libraries.

Related Works

- Protecting apps from privacy-unaware third-party libraries.
  - Running ads in separate processes or system services
  - Drawback: unable to handle **control-flow & data dependency** between host and library

  AdSplit[1], AdDroid[2]

- Detecting in-app security/privacy risks.
  - Static and dynamic analysis to detect resource access
  - Drawback: unable to detect malicious behavior of **dynamically generated code**

  Brahmastra[3], Livshits et al.[4]

[1] S. Shekhar, M. Dietz, and D. S. Wallach. Adsplit: Separating smartphone advertising from applications. In Presented as part of the 21st USENIX Security Symposium, 2012.

[2] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. AdDroid: Privilege separation for applications and advertisers in android. In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012.

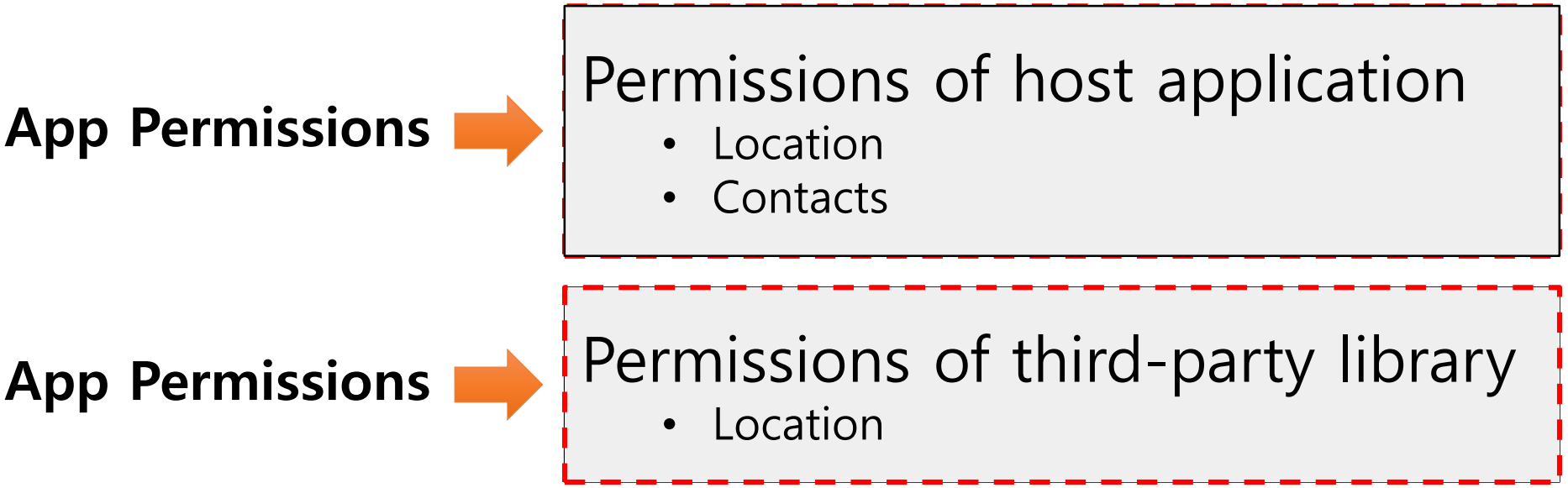
[3] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall. Brahmastra: Driving apps to test the security of third-party components. In 23rd USENIX Security Symposium, Aug. 2014.

[4] B. Livshits and J. Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In Presented as part of the 22nd USENIX Security Symposium, 2013.

Dynamic Permission Adjustment

When executing the host application’s code

When executing the 3rd-party library’s code



Dynamically adjusting the permission of an app based on the current context

Identification of Executed Code

1. Identify the principal using stack inspection
2. Apply the stack inspection to Android
3. Protect the integrity of call stack information against various attacks such as
   - JNI
   - Reflection
   - Multi-threading

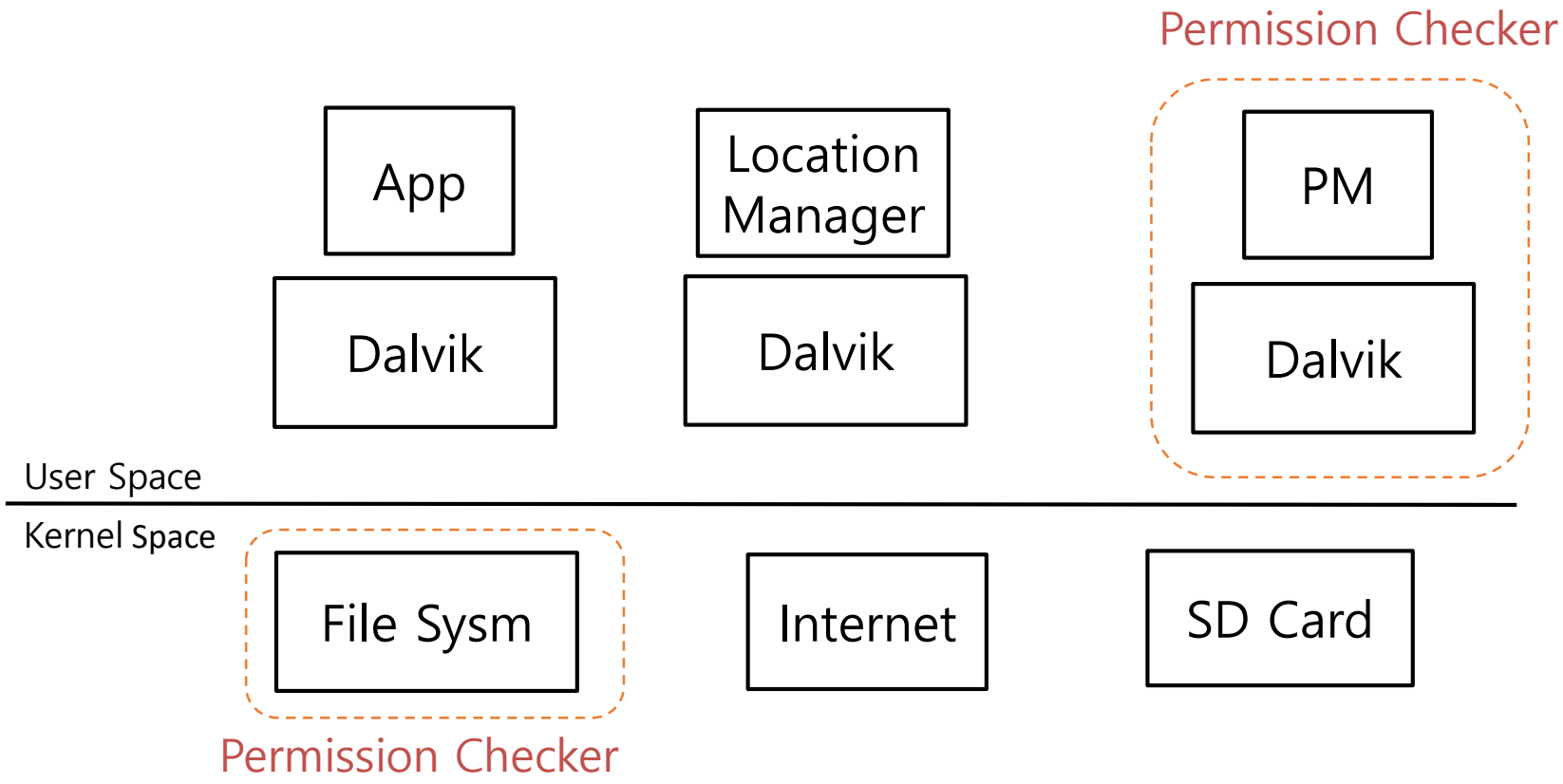
Stack Inspection in Security Context

Process of determining the permissions allowed to the current thread according to principals shown in the call stack

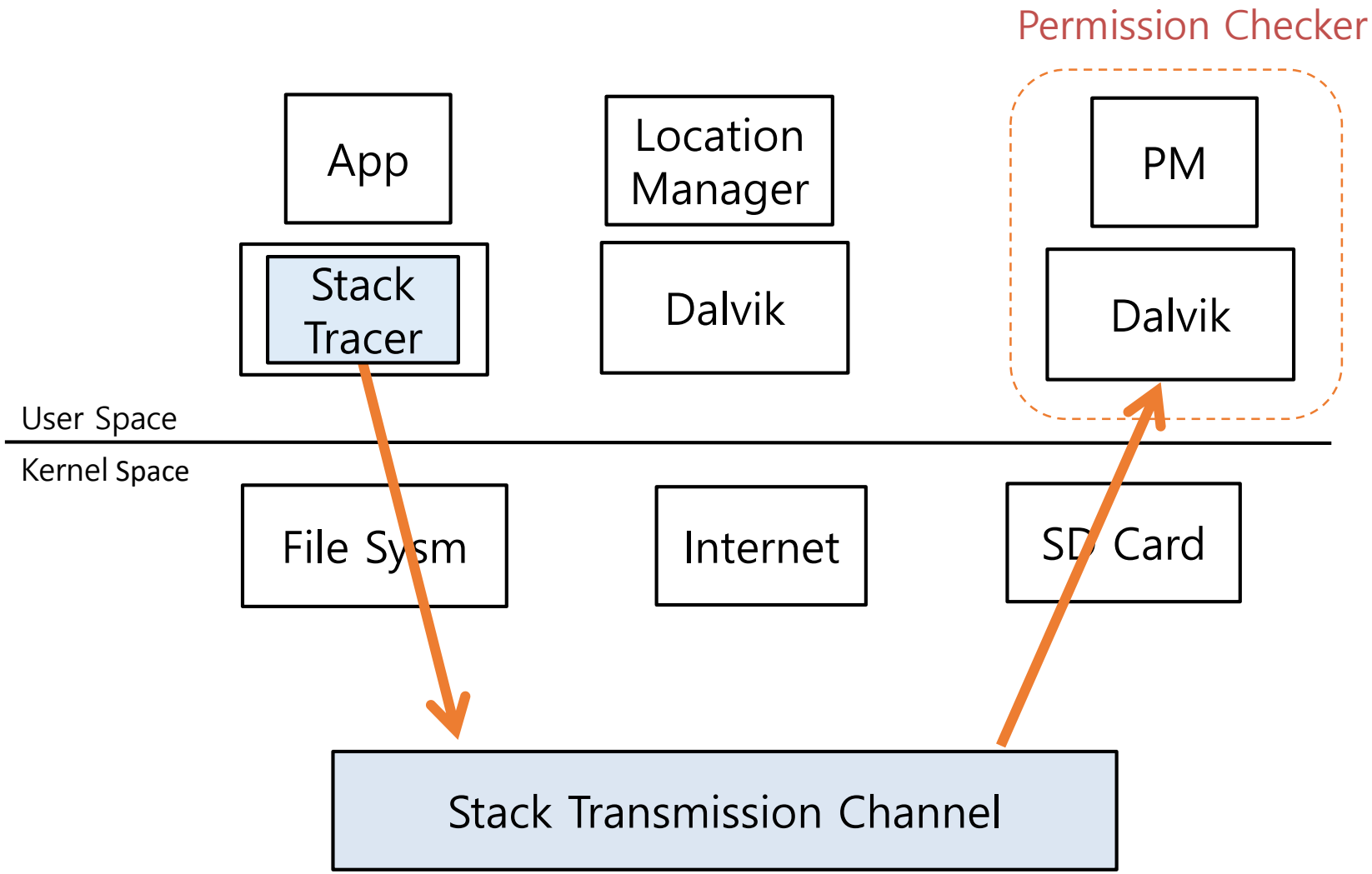
	P	Call stack
⬇	A	com.A.functionA
	B	com.B.functionB
	C	com.C.functionC

$$\text{Perm} = \text{Perm}(A) \cap \text{Perm}(B) \cap \text{Perm}(C)$$

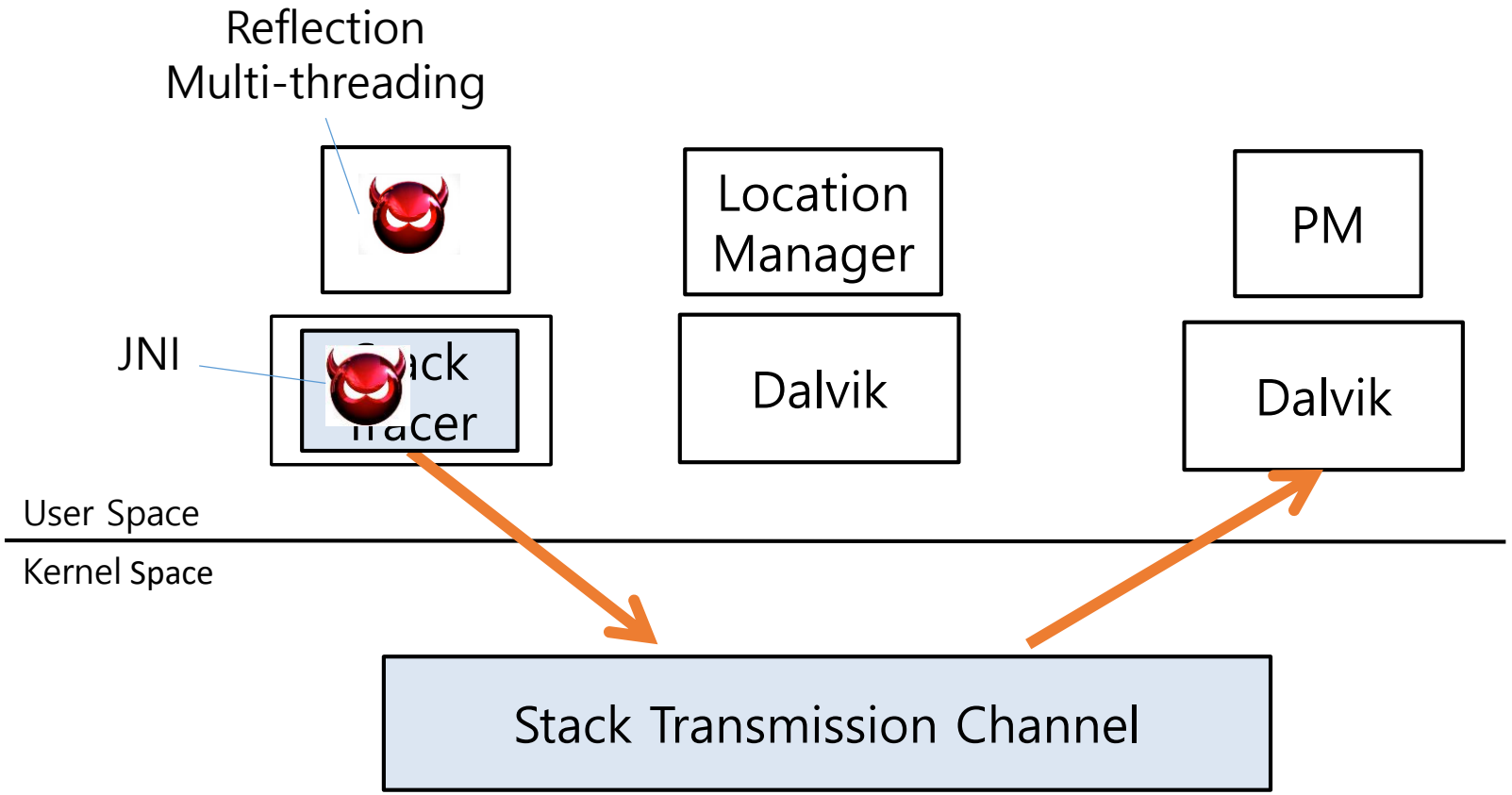
Inter-process Stack Inspection



Inter-process Stack Inspection



Potential Attack Surface



Potential Attack Surface

- Compromising stack tracer ← **JNI**
- Manipulating Dalvik call stack ← **JNI, Reflection, Multi-threading**
- Hijacking the control data ← **JNI**

  e.g., code injection on Dalvik functio ns, manipulating code pointers

Protecting Integrity of Call Stack

- JNI Sandbox
- Defense mechanism against attacks via reflection
- Defense mechanism against attacks via multi-threading

Protecting Integrity of Call Stack

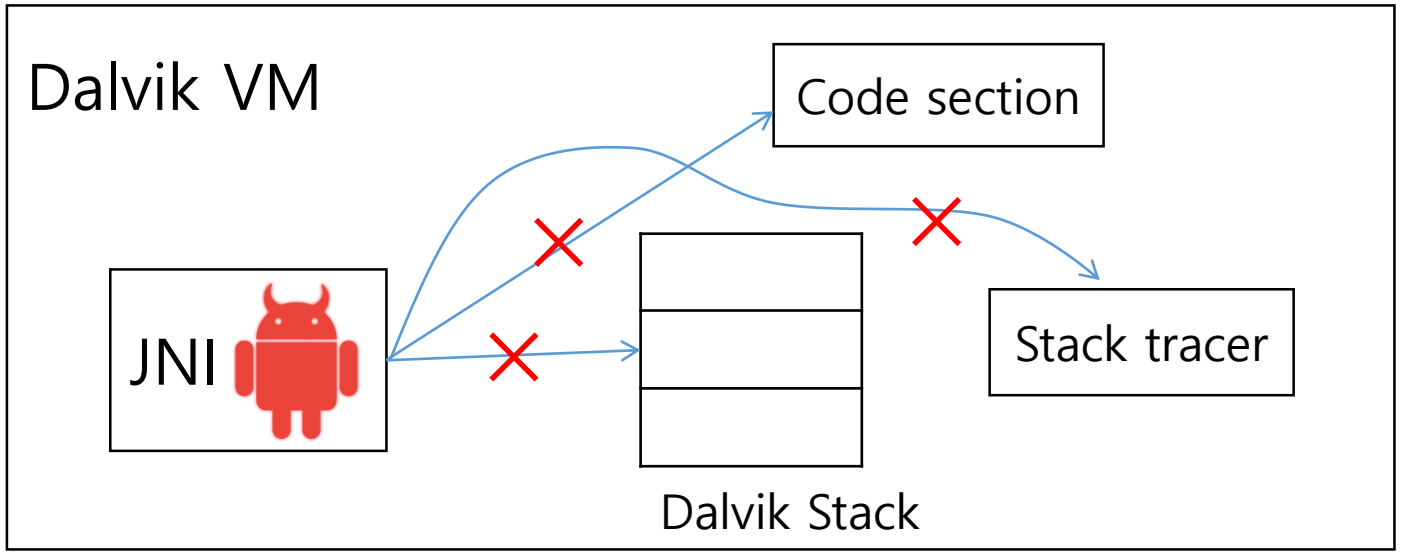
- JNI Sandbox

- Defense mechanism against attacks via reflection

- Defense mechanism against attacks via multi-threading

Defense Against Native Code Execution

- **Hardware based Fault Isolation**
  - FlexDroid enforces JNI to be executed in an isolated area under the same process
  - When JNI accesses memory out of its domain, a fault occurs



Performance Evaluation

Experiment environment: Nexus 5 / Android 4.4.4 / Kernel 3.4.0

User scenario	Android	FlexDroid	Over.
Launch an application †	39.13 ms	39.73 ms	1.55%
Launch a service	3.76 ms	3.95 ms	5.22%
Download 1.5MB file	328.58 ms	328.96 ms	0.11%
Take a photo	562.34 ms	564.89 ms	0.45%
Send an email †	4659 ms	4671 ms	0.25%
Read 6MB file via JNI	203.89 ms	203.98 ms	0.04%

† Conducted on the K-9 email app

1.04% overhead according to Antutu benchmark

Conclusion

- FlexDroid is a first mobile system that provides in-app privilege separation against JNI and dynamic runtime behavior e.g., reflection, multi thread, runtime code loading
  - However, FlexDroid cannot prevent a malicious third-party library from accessing data without access control