Android Permission assignment from packages to processes

Conflicts between Manifest, PackageManager, System information and Processus concepts

Julien Thomas

julien.thomas@protektoid.com

Protektoid Project

June 24, 2017 - Paris

Outline

Outline

Objectives of this talk

- Objectives
  - explain how permissions work and are implemented
  - explain where vulnerabilities were found
    - illustrate where other vulnerabilities could be found
  - stick to overdue or “working as Intended” submits

Objectives of this talk

- Objectives
  - explain how permissions work and are implemented
  - explain where vulnerabilities were found
    - illustrate where other vulnerabilities could be found
  - stick to overdue or "working as Intended" submits
- Origin
  - Protektoid project
    - can ask more about it after talk
  - One techincal issue (undecidability)
    - how to safely display the name of the app that wanted to connect to Internet (VPN based firewall)?

Objectives of this talk

- Objectives
  - explain how permissions work and are implemented
  - explain where vulnerabilities were found
    - illustrate where other vulnerabilities could be found
  - stick to overdue or "working as Intended" submits
- Origin
  - Protektoid project
    - can ask more about it after talk
  - One techincal issue (undecidability)
    - how to safely display the name of the app that wanted to connect to Internet (VPN based firewall)?
  - .. really far from what we are doing and talking about now

Permission and permission level

- Permission display and control

Before Marshmallow

- permissions are listed
- inherited permissions are listed
- all permissions are granted (*AppOp)

Since Marshmallow

- permissions are listed
- system-assigned permissions are granted*
- non system-assigned permissions are controlable

Permission and permission level

- Permission display and control

Before Marshmallow

- permissions are listed
- inherited permissions are listed
- all permissions are granted (*AppOp)

Since Marshmallow

- permissions are listed
- system-assigned permissions are granted*
- non system-assigned permissions are controlable

- Permission level
  - normal permissions are granted on install
  - dangerous permissions are granted on user requests and limited
  - signature permissions are granted on install if signature matches

Already known issues

- SharedUserID
  - well known concept
    - <manifest .. android:sharedUserId="protektoid.ndhxv">
    - applications share same UID
  - well known issues
    - Android Public (test) key signed sytem apps (android.uid.system)

Already known issues

- SharedUserID
  - well known concept
    - <manifest .. android:sharedUserId="protektoid.ndhxv">
    - applications share same UID
  - well known issues
    - Android Public (test) key signed sytem apps (android.uid.system)
- custom permissions: “First one in wins“
  - INSTALL_FAILED_DUPLICATE_PERMISSION

Already known issues

- SharedUserID
  - well known concept
    - <manifest .. android:sharedUserId="protektoid.ndhxv">
    - applications share same UID
  - well known issues
    - Android Public (test) key signed sytem apps (android.uid.system)
- custom permissions: “First one in wins“
  - INSTALL_FAILED_DUPLICATE_PERMISSION
- Talk about permissions but not about these known issues
  - sharedUserID: algorithm logic (failure)
  - permission management flaws
  - custom permissions: a tool for vector of attacks

Guess it

- Install app getThemAll
- Install app Ring1 and Ring2
  - No permission asked

Guess it

- Install app getThemAll
- Install app Ring1 and Ring2
  - No permission asked
- Let's update app Ring1 and Ring2
  - No permission asked or mentionned?

Guess it

- Install app getThemAll
- Install app Ring1 and Ring2
  - No permission asked
- Let's update app Ring1 and Ring2
  - No permission asked or mentionned?
- Question 1: if we reopen getThemAll ..
  - what PackageManager says about getThemAll?
  - what SystemInfo says about getThemAll?

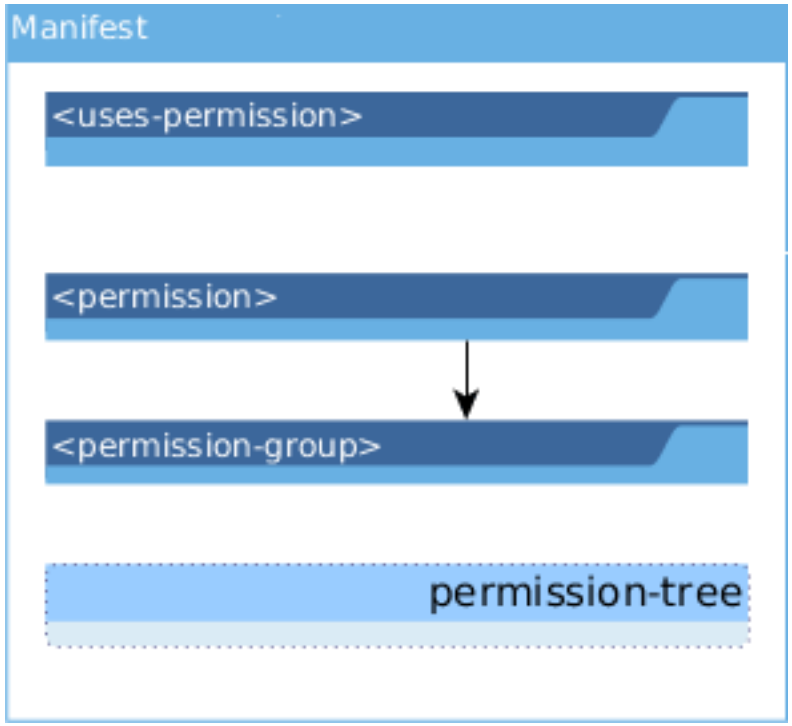
Guess it

- Install app getThemAll
- Install app Ring1 and Ring2
  - No permission asked
- Let's update app Ring1 and Ring2
  - No permission asked or mentionned?
- Question 1: if we reopen getThemAll ..
  - what PackageManager says about getThemAll?
  - what SystemInfo says about getThemAll?
- Question 2: what can do getThemAll?

Outline

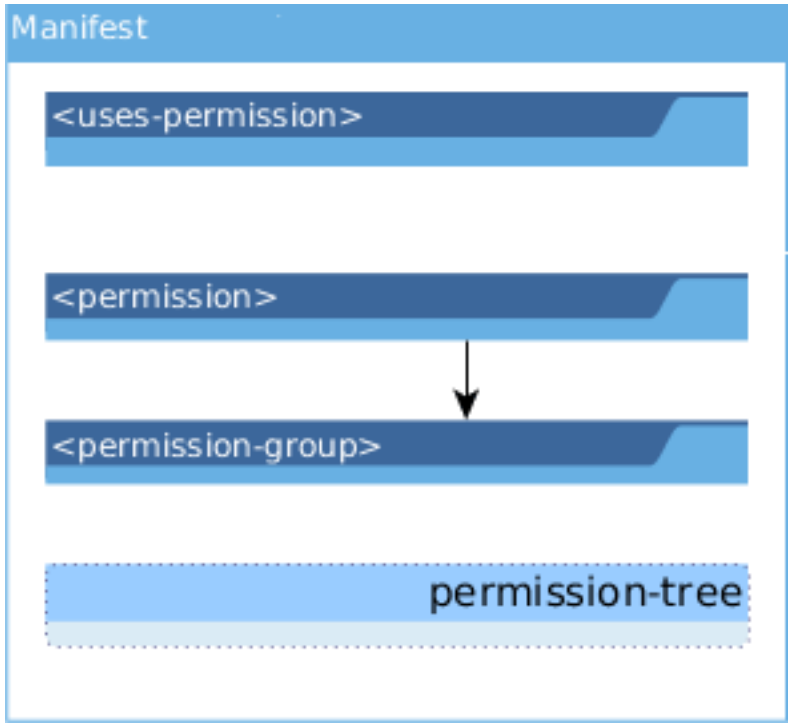
Manifest definition: user controled

- Permission usage: <permission-uses>
  - asks to uses system permissions
  - asks to uses other app permissions



Manifest definition: user controled

- Permission usage: <permission-uses>
  - asks to uses system permissions
  - asks to uses other app permissions
- Permission definition: <permission>
  - defines permission
  - uses normal permissions
  - asks to uses dangerous permissions
  - can be grouped



System and process definition: android logic

- Manifest definition at install
  - assigns UID
  - allows custom permissions
  - generates (rewritten) manifest in database
    - /data/system/packages.xml - root Protected

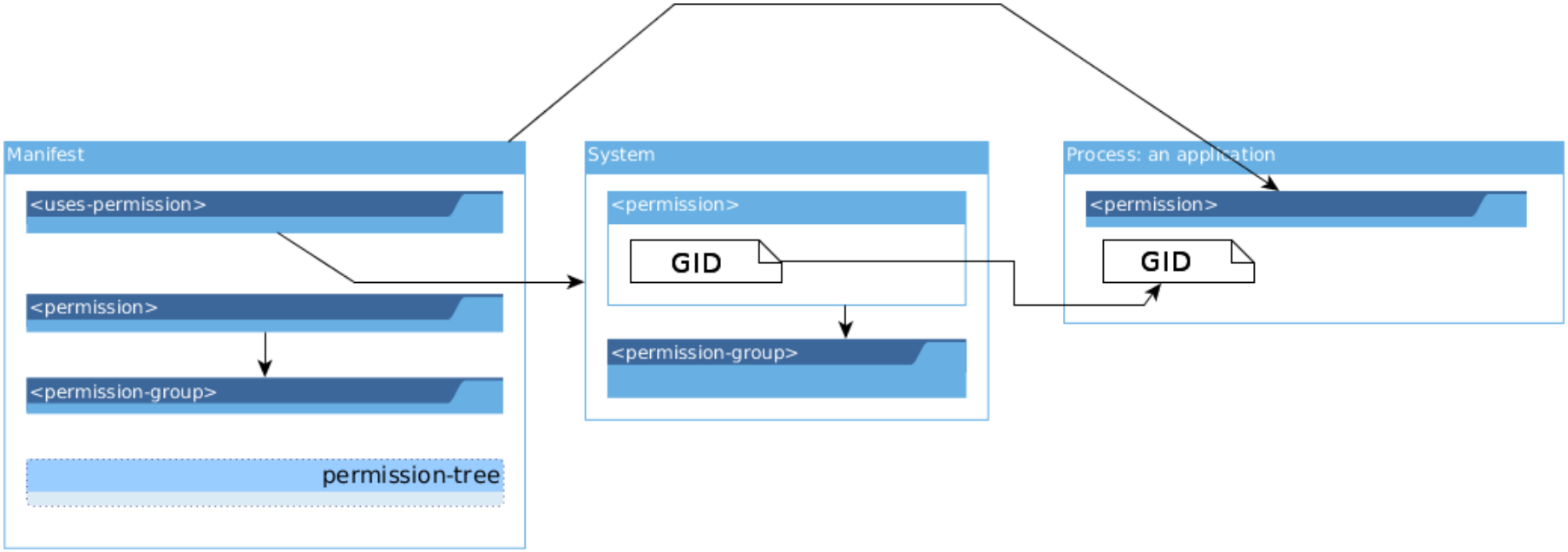
System and process definition: android logic

- Manifest definition at install
  - assigns UID
  - allows custom permissions
  - generates (rewritten) manifest in database
    - /data/system/packages.xml - root Protected
- System permissions
  - define permissions on the system
    - /etc/permissions/platform.xml - root Protected
    - OEM data - root Protected
  - map system permission to GID

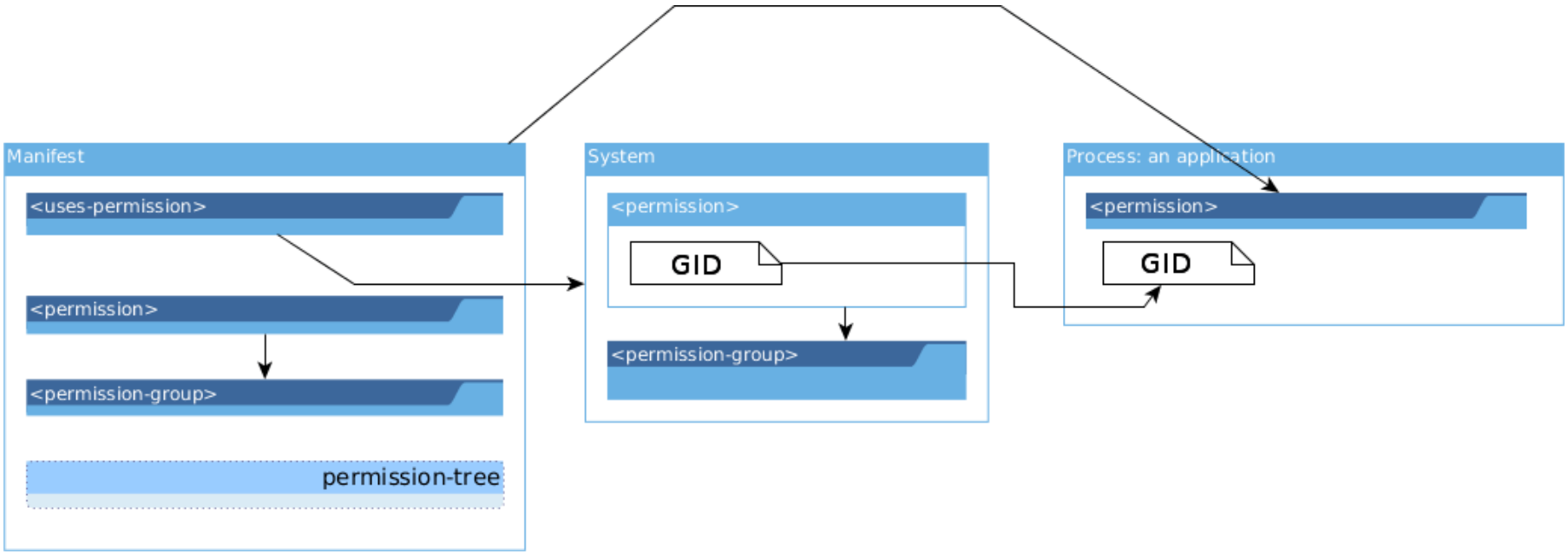
System and process definition: android logic

- Manifest definition at install
  - assigns UID
  - allows custom permissions
  - generates (rewritten) manifest in database
    - /data/system/packages.xml - root Protected
- System permissions
  - define permissions on the system
    - /etc/permissions/platform.xml - root Protected
    - OEM data - root Protected
  - map system permission to GID
- Process launch
  - assigned a UID
  - assigned a set of permissions
  - assigned set of GIDs (Zygote)

Permission model overview: the complete model



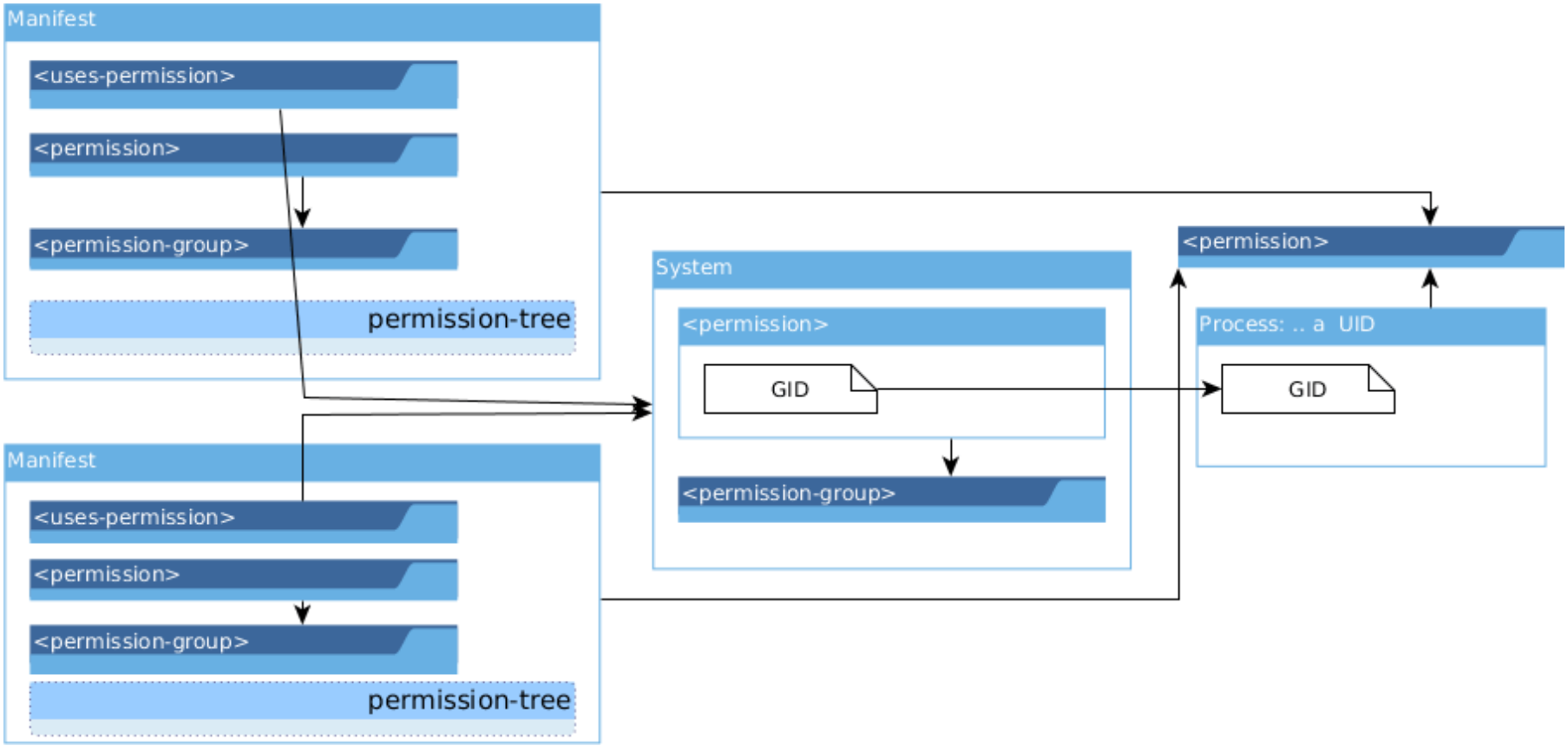
Permission model overview: the complete model



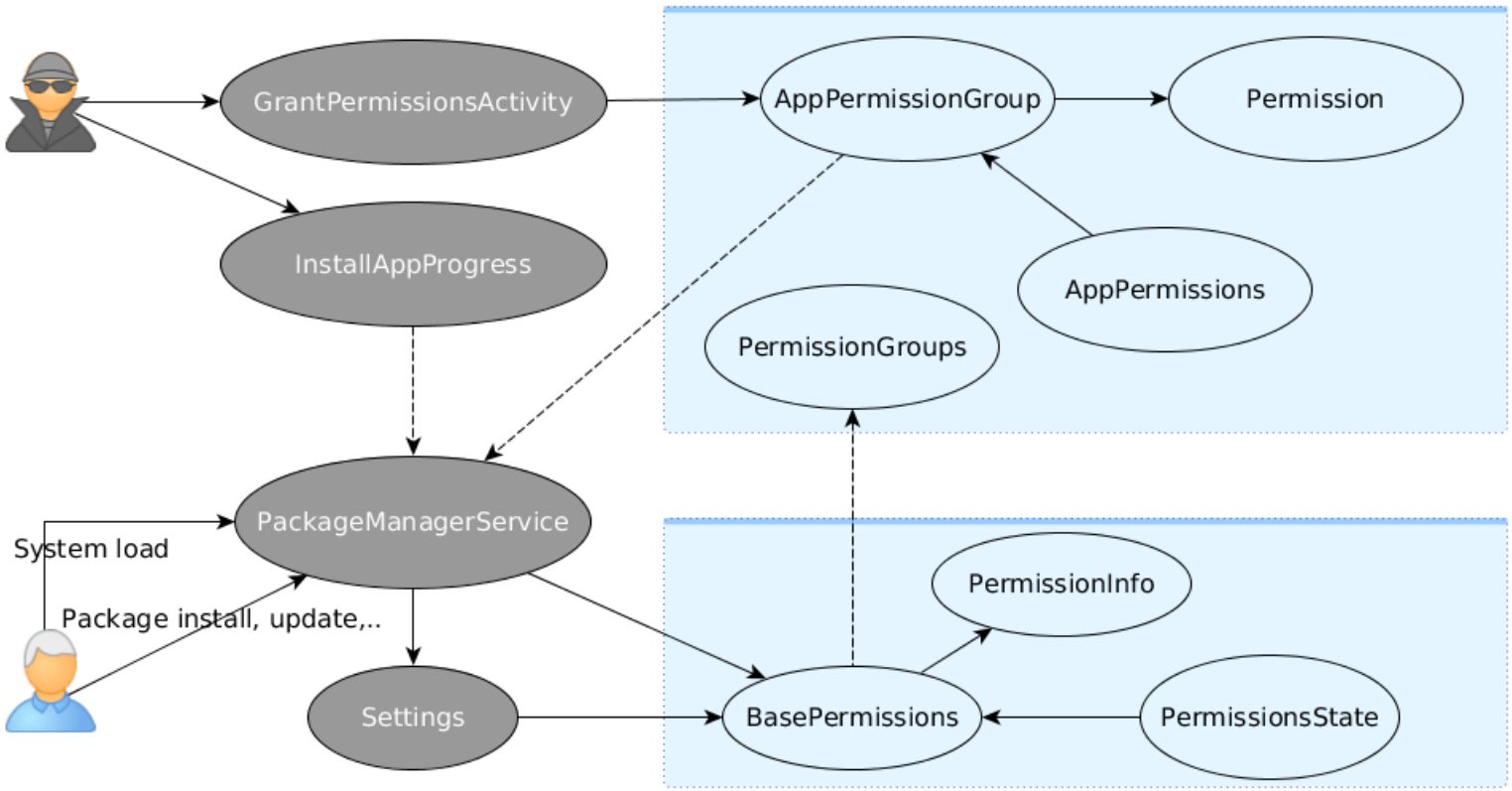
- Is it that simple?

Permission model overview: the complete model

- Is it that simple?
- Applications can share UID ⇒ model more complex



Permission implementation overview



Permission list build

- System launch: PacakgeManagerService.writeLPr

Permission list build

- System launch: PacakgeManagerService.writeLPr
- App launch
  - Zygote (startProcessLocked): load app (create new process), set GID

Permission list build

- System launch: PacakgeManagerService.writeLPr
- App launch
  - Zygote (startProcessLocked): load app (create new process), set GID
- Application update /removal
  - PacakgeManagerService.updatePermissionsLPw
  - removePackageDataLIF $\rightarrow$ mSettings.removePackageLPw

Permission list build

- System launch: PacakgeManagerService.writeLPr
- App launch
  - Zygote (startProcessLocked): load app (create new process), set GID
- Application update /removal
  - PacakgeManagerService.updatePermissionsLPw
  - removePackageDataLIF $\rightarrow$ mSettings.removePackageLPw
- Is the removal algorithm sound compared to reboot / install-no-update?
  - before M: no
  - since M: no

Permission state computation

- Permission state is targetSdkVersion dependant to match “app can not be disabled/enabled” “issue” before M.
  - application update requires recomputation of permission status
  - application install requires recomputation of permission status
    - PacakgeManagerService.grantPermissionsLPw

Permission state computation

- Permission state is targetSdkVersion dependant to match “app can not be disabled/enabled” “issue” before M.
  - application update requires recomputation of permission status
  - application install requires recomputation of permission status
    - PacakgeManagerService.grantPermissionsLPw
- User set and read on States are targetSdkVersion dependant
  - set: relies on AppOpManager if <M or PermissionState if ≥M
  - read: relies on PermissionState (and AppOpManager if <M)

Permission state computation

- Permission state is targetSdkVersion dependant to match “app can not be disabled/enabled” “issue” before M.
  - application update requires recomputation of permission status
  - application install requires recomputation of permission status
    - PacakgeManagerService.grantPermissionsLPw
- User set and read on States are targetSdkVersion dependant
  - set: relies on AppOpManager if <M or PermissionState if ≥M
  - read: relies on PermissionState (and AppOpManager if <M)
- Is backward compatility correctly handled?
- Is logic sound?

Permission state computation

- Permission state is targetSdkVersion dependant to match "app can not be disabled/enabled" "issue" before M.
  - application update requires recomputation of permission status
  - application install requires recomputation of permission status
    - PacakgeManagerService.grantPermissionsLPw
- User set and read on States are targetSdkVersion dependant
  - set: relies on AppOpManager if <M or PermissionState if ≥M
  - read: relies on PermissionState (and AppOpManager if <M)
- Is backward compatility correctly handled?
- Is logic sound?
  - .. not really but "not promised" ☹

Permission Grouping

- Permission may be grouped

Permission Grouping

- Permission may be grouped
- If a permission is granted, all other permission are granted-by-request without user consent
  - relies on GrantPermissionsActivity
  - what if 2 permissions are in different level but same group?
  - what if a permission is signature level?

Permission Grouping

- Permission may be grouped
- If a permission is granted, all other permission are granted-by-request without user consent
  - relies on GrantPermissionsActivity
  - what if 2 permissions are in different level but same group?
  - what if a permission is signature level?

- Can we still hijack the grouping concept to grant permissions without user consent?
  - yes ☺.. but were two late by 3 months ☹

Permission displays

- System Info
  - displays package permissions based on PackageInfo and PackageParser (AppPermissions.loadPermissionGroups)

Permission displays

- System Info
  - displays package permissions based on PackageInfo and PackageParser (AppPermissions.loadPermissionGroups)
    - before Marshmallow, used AppSecurityPermission

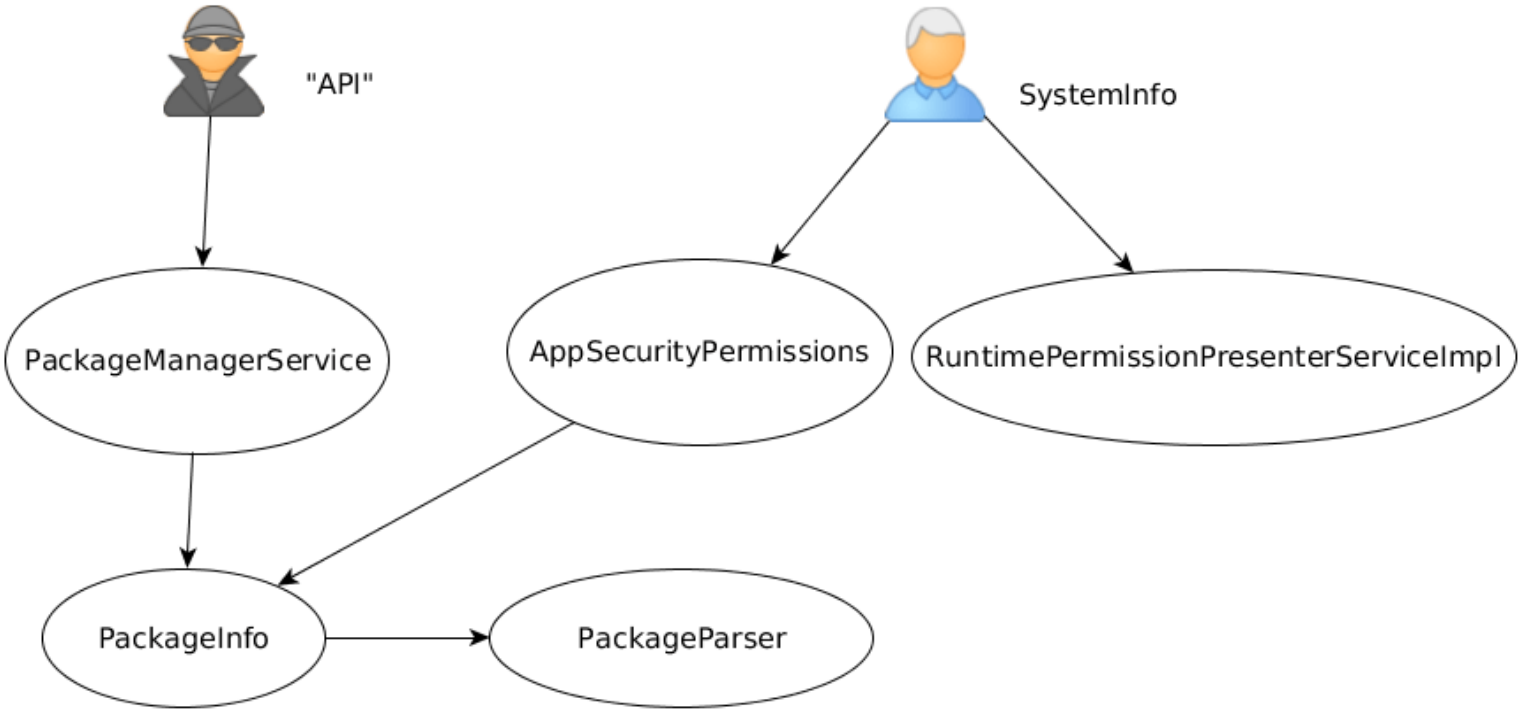
Permission displays

- System Info
  - displays package permissions based on PackageInfo and PackageParser (AppPermissions.loadPermissionGroups)
    - before Marshmallow, used AppSecurityPermission
  - displays package permission status directly from system
    - RuntimePermissionPresenterServiceImpl

Permission displays

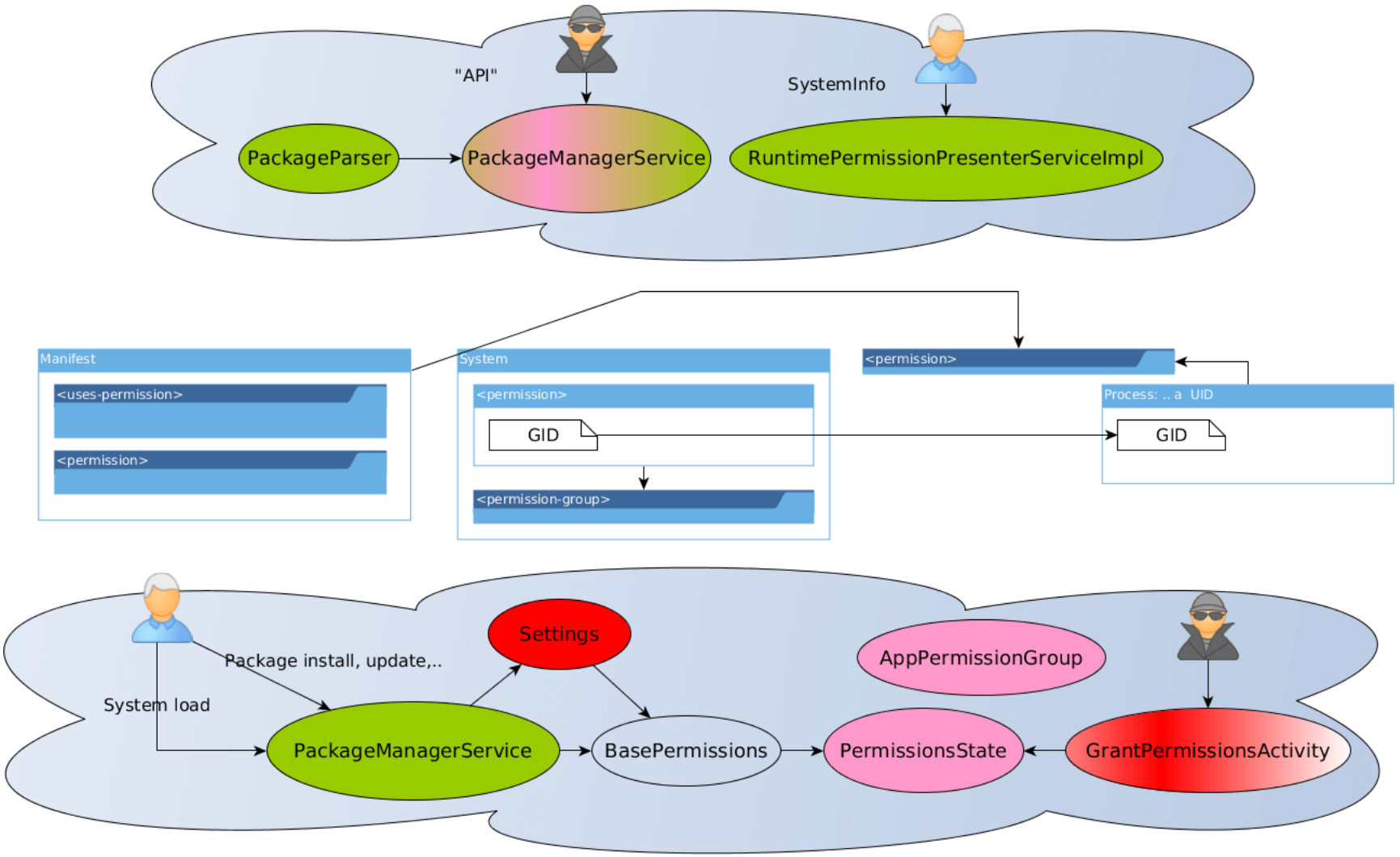
- System Info
  - displays package permissions based on PackageInfo and PackageParser (AppPermissions.loadPermissionGroups)
    - before Marshmallow, used AppSecurityPermission
  - displays package permission status directly from system
    - RuntimePermissionPresenterServiceImpl
- PackageManager "API" (for "security" apps)
  - displays package permissions list based on PackageInfo and PackageParser
  - displays package permissions status based on PackageInfo and PackageParser

Permission Display Overview



Outline

Overview


"API"
SystemInfo
PackageParser
PackageManagerService
RuntimePermissionPresenterServiceImpl
Manifest
<uses-permission>
<permission>
System
<permission>
GID
<permission-group>
<permission>
Process: .. a UID
GID
Settings
Package install, update,..
System load
PackageManagerService
BasePermissions
AppPermissionGroup
PermissionsState
GrantPermissionsActivity

Hiding permissions from Android

- Submission 37151831 coupled with 37601324: abusing of the security information (restricted to Manifest) vs Process logic to hide permission
  - PackageParser vs real permissions
  - mapping logic issue

Hiding permissions from Android

```
//Marshmallow as example, SystemInfo
private void loadPermissionGroups() {
  List groups = new ArrayList<>();
  if (mPackageInfo.requestedPermissions == null) {
    return;
  }
  for (int i = 0; i <
        mPackageInfo.requestedPermissions.length;
        i++) { [...] }
}
```

```
//fixing system, but PackageManager API?
private PackageInfo
      generatePackageInfo(PackageSetting ps, int
      flags, int userId) {
  [...]
  final Set permissions =
        ArrayUtils.isEmpty(p.requestedPermissions)
          ? Collections.emptySet() :
                permissionsState.getPermissions(userId);
  return PackageParser.generatePackageInfo(p, gids,
        [...], permissions, state, userId);
}
```

```
//Kitkat as example, SystemInfo
private void getAllUsedPermissions(int sharedUid,
      Set pS){
  String sPL[] = mPm.getPackagesForUid(sharedUid);
  if(sPL == null || (sPL.length == 0)) {
    return;
  }
  for(String sharedPkg : sPL) {
    getPermissionsForPackage(sharedPkg, pS);
  }
}

private void getPermissionsForPackage(String
      packageName, Set<MyPermissionInfo> permSet) {
  try {
    PackageInfo pkgInfo =
          mPm.getPackageInfo(packageName,
          PackageManager.GET_PERMISSIONS);
    extractPerms(pkgInfo, permSet, pkgInfo);
  } catch (NameNotFoundException e) {
    [...]
  }

}
```

Hiding permissions from Android

```
//Marshmallow as example, SystemInfo
private void loadPermissionGroups() {
  List groups = new ArrayList<>();
  if (mPackageInfo.requestedPermissions == null) {
    return;
  }
  for (int i = 0; i <
        mPackageInfo.requestedPermissions.length;
        i++) { [...] }
}
```

```
//fixing system, but PackageManager API?
private PackageInfo
      generatePackageInfo(PackageSetting ps, int
      flags, int userId) {
  [...]
  final Set permissions =
        ArrayUtils.isEmpty(p.requestedPermissions)
          ? Collections.emptySet() :
                permissionsState.getPermissions(userId);
  return PackageParser.generatePackageInfo(p, gids,
        [...], permissions, state, userId);
}
```

```
//Kitkat as example, SystemInfo
private void getAllUsedPermissions(int sharedUid,
      Set pS){
  String sPL[] = mPm.getPackagesForUid(sharedUid);
  if(sPL == null || (sPL.length == 0)) {
    return;
  }
  for(String sharedPkg : sPL) {
    getPermissionsForPackage(sharedPkg, pS);
  }
}
```

```
private void getPermissionsForPackage(String
      packageName, Set<MyPermissionInfo> permSet) {
  try {
    PackageInfo pkgInfo =
          mPm.getPackageInfo(packageName,
          PackageManager.GET_PERMISSIONS);
    extractPerms(pkgInfo, permSet, pkgInfo);
  } catch (NameNotFoundException e) {
    [...]
  }

}
```

Hiding permissions from Android

- Submission 37151831 coupled with 37601324: abusing of the security information (restricted to Manifest) vs Process logic to hide permission
  - PackageParser vs real permissions
  - mapping logic issue
- Steps
  - install app app1v2 (same as demo)
  - install app app2v1 (same as demo)
  - grant contact to app1v2 (that could be bypassed)

Hiding permissions from Android

- Submission 37151831 coupled with 37601324: abusing of the security information (restricted to Manifest) vs Process logic to hide permission
  - PackageParser vs real permissions
  - mapping logic issue
- Steps
  - install app app1v2 (same as demo)
  - install app app2v1 (same as demo)
  - grant contact to app1v2 (that could be bypassed)
- App app2v1 can acces to contact list too yet not listed neither on “API” nor system Info
  - flagged as not a security issue (37151831) and Intended Behavior (37601324)

Hiding permission states from SystemInfo

- Submission 38493552: make permissions flagged as not enabled yet enabled
  - abuse of the AppOp vs Permission.isGranted logic on AppPermissionGroup
  - backward compatility logic issue
  - partial explanation $\Rightarrow$ not full disclosure today

Hiding permission states from SystemInfo

```
public boolean areRuntimePermissionsGranted(){
  [...]
  final int permissionCount =
        mPermissions.size();
  for (int i = 0; i < permissionCount; i++) {
    Permission permission =
          mPermissions.valueAt(i);
    if (mAppSupportsRuntimePermissions) {
      if (permission.isGranted()) {
        return true;
      }
    } else if (permission.isGranted() &&
          permission.getAppOp()
      != AppOpsManager.OP_NONE &&
            permission.isAppOpAllowed()) {
      return true;
    }
  }
  return false;
}

//
```

```
public boolean revokeRuntimePermissions(boolean fixedByTheUser){
  final boolean isSharedUser = mPI.sharedUserId!=null;
  final int uid = mPI.applicationInfo.uid;
  for (Permission p : mPermissions.values()) {
    if (mAppSupportsRuntimePermissions) {
      if (p.isGranted()) {
        p.setGranted(false);
        mPM.revokeRuntimePermission( mPI.packageName,
              p.getName(), mUserHandle);
      } [...]
    } else {
      if (p.isAppOpAllowed()) {
        p.setAppOpAllowed(false);
        if (isSharedUser) {
          String[] pNames=mPM.getPackagesForUid(uid);
          for (String packageName : pNames ) {
            mAppOps.setUidMode(p.getAppOp(), uid,
              AppOpsManager.MODE_IGNORED);
          }
        } else {
          mAppOps.setUidMode(p.getAppOp(), uid,
                AppOpsManager.MODE_IGNORED);
        } [...]
  } } }
  return true;
}
```

Hiding permission states from SystemInfo

```
public boolean areRuntimePermissionsGranted(){
  [...]
  final int permissionCount =
        mPermissions.size();
  for (int i = 0; i < permissionCount; i++) {
    Permission permission =
          mPermissions.valueAt(i);
    if (mAppSupportsRuntimePermissions) {
      if (permission.isGranted()) {
        return true;
      }
    } else if (permission.isGranted() &&
          permission.getAppOp()
      != AppOpsManager.OP_NONE &&
            permission.isAppOpAllowed()) {
      return true;
    }
  }
  return false;
}

//
```

```
public boolean revokeRuntimePermissions(boolean fixedByTheUser){
  final boolean isSharedUser = mPI.sharedUserId!=null;
  final int uid = mPI.applicationInfo.uid;
  for (Permission p : mPermissions.values()) {
    if (mAppSupportsRuntimePermissions) {
      if (p.isGranted()) {
        p.setGranted(false);
        mPM.revokeRuntimePermission( mPI.packageName,
              p.getName(), mUserHandle);
      } [...]
    } else {
      if (p.isAppOpAllowed()) {
        p.setAppOpAllowed(false);
        if (isSharedUser) {
          String[] pNames=mPM.getPackagesForUid(uid);
          for (String packageName : pNames ) {
            mAppOps.setUidMode(p.getAppOp(), uid,
              AppOpsManager.MODE_IGNORED);
          }
        } else {
          mAppOps.setUidMode(p.getAppOp(), uid,
                AppOpsManager.MODE_IGNORED);
        } [...]
  } } }
  return true;
}
```

Hiding permission states from SystemInfo

- Submission 38493552: make permissions flagged as not enabled yet enabled
  - abuse of the AppOp vs Permission.isGranted logic on AppPermissionGroup
  - backward compatility logic issue
  - partial explanation ⇒ not full disclosure today
- Steps (combined with 37150554 for auto granting)
  - install app demo2_app1 (Kitkat)
  - install app demo2_app2
  - ungrant Contact from demo2_app1

Hiding permission states from SystemInfo

- Submission 38493552: make permissions flagged as not enabled yet enabled
  - abuse of the AppOp vs Permission.isGranted logic on AppPermissionGroup
  - backward compatility logic issue
  - partial explanation ⇒ not full disclosure today
- Steps (combined with 37150554 for auto granting)
  - install app demo2_app1 (Kitkat)
  - install app demo2_app2
  - ungrant Contact from demo2_app1
- App demo2_app1 can access to contact list yet not listed on System
  - confusing results with demo2_app2
  - bug under review

Keeping un-requested permissions even after permission-owned is removed

- Submission 37150440: how permissions are managed on app removal?
  - PackageManager → mSettings
  - incorrect check of PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED
  - logic issue
  - partial explanation ⇒ not full disclosure today

Keeping un-requested permissions even after permission-owned is removed

```
int updateSharedUserPermsLPw( PackageSetting
      deletedPs, int userId) {
  [...]
  // Update permissions
  for (String eachPerm :
        deletedPs.pkg.requestedPermissions) {
    BasePermission bp = mPermissions.get(eachPerm);
    if (bp == null) { continue; }
    // Check if another package in the shared user
          needs the permission.
    boolean used = false;
    for (PackageSetting pkg : sus.packages) {
      if (pkg.pkg != null &&
        !pkg.pkg.packageName.equals(
              deletedPs.pkg.packageName)&&
         pkg.pkg.requestedPermissions.contains(eachPerm))
               {
        used = true; break;
      }
    }

    if (used) { continue;}
    PermissionsState permissionsState =
          sus.getPermissionsState();
    [...]
    // Try to revoke as an install permission which is
          for all users.
    permissionsState.updatePermissionFlags(bp, userId,
          PackageManager.MASK_PERMISSION_FLAGS, 0);
    if (permissionsState.revokeInstallPermission(bp)
          == PermissionsState.
          PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED){
      return UserHandle.USER_ALL;
    }
    // Try to revoke as an install permission which is
          per user.
    if (permissionsState.revokeRuntimePermission(bp,
          userId) == PermissionsState.
          PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED){
      return userId;
    }
  }
  return UserHandle.USER_NULL;
}
```

Keeping un-requested permissions even after permission-owned is removed

```
int updateSharedUserPermsLPw( PackageSetting
      deletedPs, int userId) {
  [...]
  // Update permissions
  for (String eachPerm :
        deletedPs.pkg.requestedPermissions) {
    BasePermission bp = mPermissions.get(eachPerm);
    if (bp == null) { continue; }
    // Check if another package in the shared user
          needs the permission.
    boolean used = false;
    for (PackageSetting pkg : sus.packages) {
      if (pkg.pkg != null &&
        !pkg.pkg.packageName.equals(
              deletedPs.pkg.packageName)&&
         pkg.pkg.requestedPermissions.contains(eachPerm))
               {
        used = true; break;
      }
    }

    if (used) { continue;}
    PermissionsState permissionsState =
          sus.getPermissionsState();
    [...]
    // Try to revoke as an install permission which is
          for all users.
    permissionsState.updatePermissionFlags(bp, userId,
          PackageManager.MASK_PERMISSION_FLAGS, 0);
    if (permissionsState.revokeInstallPermission(bp)
          == PermissionsState.
          PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED){
      return UserHandle.USER_ALL;
    }
    // Try to revoke as an install permission which is
          per user.
    if (permissionsState.revokeRuntimePermission(bp,
          userId) == PermissionsState.
          PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED){
      return userId;
    }
  }
  return UserHandle.USER_NULL;
}
```

Keeping un-requested permissions even after permission-owned is removed

- Submission 37150440: how permissions are managed on app removal?
  - PackageManager → mSettings
  - incorrect check of PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED
  - logic issue
  - partial explanation ⇒ not full disclosure today
- Steps (combined with 37150554 just because we can)
  - install app demo3_app1
  - install app demo3_app2
  - remove app demo3_app1

Keeping un-requested permissions even after permission-owned is removed

- Submission 37150440: how permissions are managed on app removal?
  - PackageManager → mSettings
  - incorrect check of PERMISSION_OPERATION_SUCCESS_GIDS_CHANGED
  - logic issue
  - partial explanation ⇒ not full disclosure today
- Steps (combined with 37150554 just because we can)
  - install app demo3_app1
  - install app demo3_app2
  - remove app demo3_app1
- App demo3_app2 can send your contact list to a remote server. Not possible without demo3_app1
  - bug under review

Auto-granting permissions

- Submission 36887724: Auto granting dangerous permissions by application upgrade since M
  - logic issue
  - initial submit report: 3386074
  - not the owner $\Rightarrow$ not full disclosure today

Auto-granting permissions

- Submission 36887724: Auto granting dangerous permissions by application upgrade since M
  - logic issue
  - initial submit report: 3386074
  - not the owner $\Rightarrow$ not full disclosure today
- Steps: presented in introduction demo
  - install app app1v1
  - update app to app1v2

Auto-granting permissions

- Submission 36887724: Auto granting dangerous permissions by application upgrade since M
  - logic issue
  - initial submit report: 3386074
  - not the owner ⇒ not full disclosure today
- Steps: presented in introduction demo
  - install app app1v1
  - update app to app1v2
- App app1v2 can send your contact to Internet

Some more

- Some useful ones
  - keeping permission on upgrade (kitkat)
  - preventing app pemission from being ungranted
  - auto granting dangerous as install perm (kitkat+Marshmallow)

Some more

- Some useful ones
  - keeping permission on upgrade (kitkat)
  - preventing app pemission from being ungranted
  - auto granting dangerous as install perm (kitkat+Marshmallow)
- Some non-useful ones
  - preventing app pemission from being granted
  - displaying permissions are enabled yet disabled (API)

Outline

4 Analyzes
- Google feedbacks
- Android market distribution
- Android security Apps

Google feedbacks

- 1 inital submit: labeled as Working as intendeed ☹

Google feedbacks

- 1 inital submit: labeled as Working as intendeed ☹
- 10 additionnal submits

Google feedbacks

- 1 inital submit: labeled as Working as intendeed ☹
- 10 additionnal submits
- 4 months later ☺

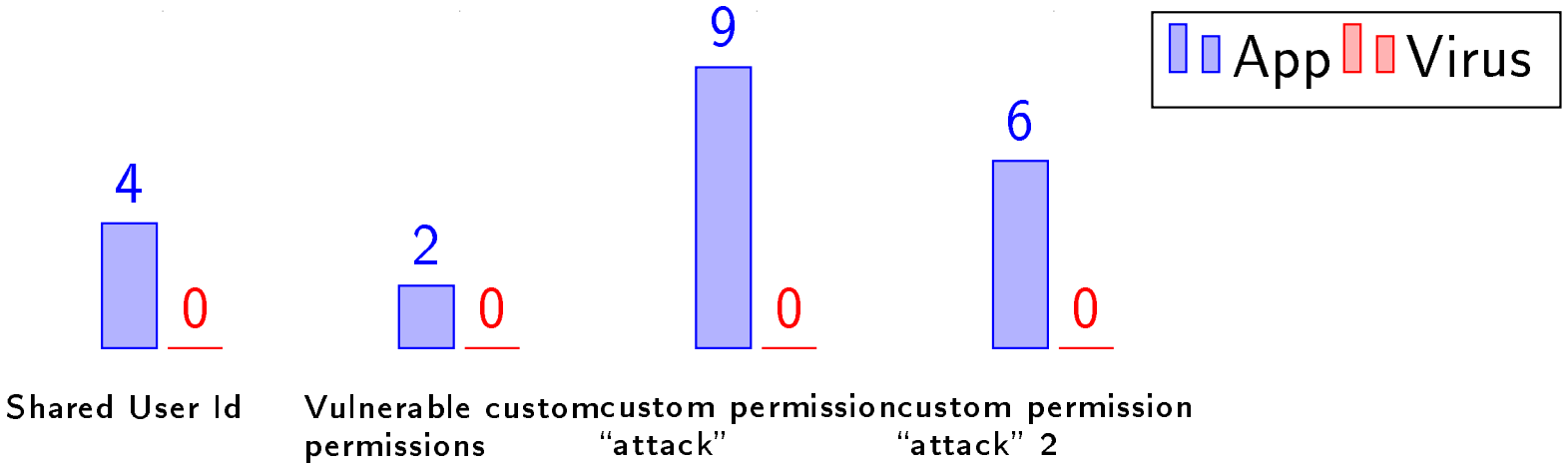
Labeled	Quantity	Remark
Working as intendeed	1 3	agree: 37324271 disagree: 37150554 3715054,37601324
feature request, not a security issue	1	disagree: 36989024
medium security issue	1	36899497
waiting for feedbacks	3	-
duplicate	1	36887724

Android market distribution

- 150 analyzed apps
  - online stores top 100
  - protektoid top 100
  - virus top 33 (thanks koodous for the APKs ☺)

Android market distribution

- 150 analyzed apps
  - online stores top 100
  - protektoid top 100
  - virus top 33 (thanks koodous for the APKs ☺)
- some GIYF filters
  - “android:permissionGroup inurl:github”. (some false positives)
  - “<permission-group inurl:github”



- what if they start doing it?

Android security Apps

- 13 security app compared (working on Nougat)

Android security Apps

- 13 security app compared (working on Nougat)
  - 11 ignore permission inheritance (37601324)
  - 77% do not care about permission status

Q	perm status	inheritance detection	inherited perm status	system grouped custom perm inherited
1*	✓	✓	✓	✓
1	✓	✗	✗	✗
1	✓	✗	✗	✓
1	✗	✗	✗	✓
1	✗	✗	✗	conditional
1	✗	✗	✗	✓ no perm search
1	✗	✓	✗	✗
1	✗	✓ not clear	✗	✗
4	✗	✗	✗	✗
1	✗ hidden if not enabled	✗	✗	✗

Outline

Conclusion

- Android permission model is really good but still some issues
  - update path not always sound
  - conflicts when considering SharedUserId, rarely used

Conclusion

- Android permission model is really good but still some issues
  - update path not always sound
  - conflicts when considering SharedUserId, rarely used
- Rebooting device fix most of the problems
  - but who will (TV, watches, ..)?

Conclusion

- Android permission model is really good but still some issues
  - update path not always sound
  - conflicts when considering SharedUserId, rarely used
- Rebooting device fix most of the problems
  - but who will (TV, watches, ..)?
- Security application are still needed
  - providing they care to define what they want
  - providing they dig deeper into AOSP soure code
  - even with new services such as Play Protect

Conclusion

- Android permission model is really good but still some issues
  - update path not always sound
  - conflicts when considering SharedUserId, rarely used
- Rebooting device fix most of the problems
  - but who will (TV, watches, ..)?
- Security application are still needed
  - providing they care to define what they want
  - providing they dig deeper into AOSP soure code
  - even with new services such as Play Protect
- Thank you!