PrivDroid: Android Security Code Smells for Privilege Escalation Prevention

Mohammed El Amin TEBIB (LCIS - UGA)

Pascal André (LS2N - NU)

Oum-El-Kheir Aktouf (LCIS-UGA)

Mariem Graa (LCIS-UGA)

DASC 2023 / VELO

14-17 Nov, 2023 - Abu Dhabi, UAE

The 21st IEEE International Conference on Dependable, Autonomic & Secure Computing

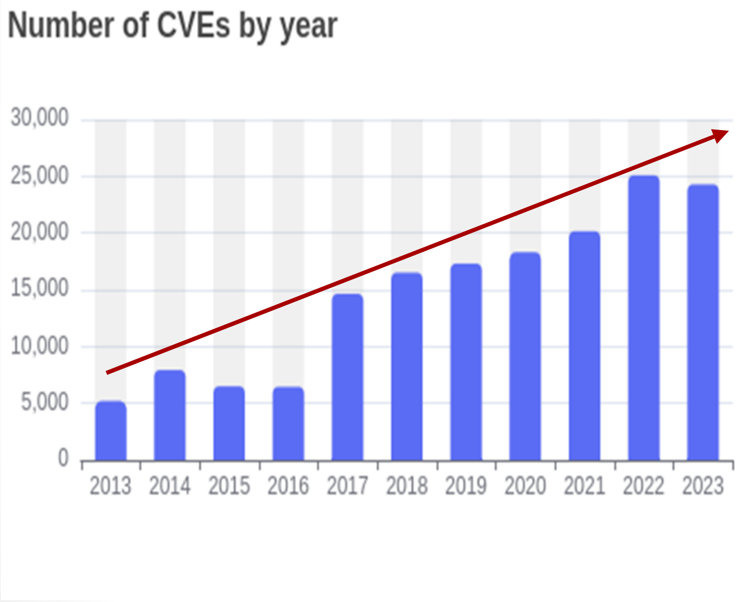
The 21st IEEE International Conference on Dependable, Autonomic & Secure Computing (DASC 2023)

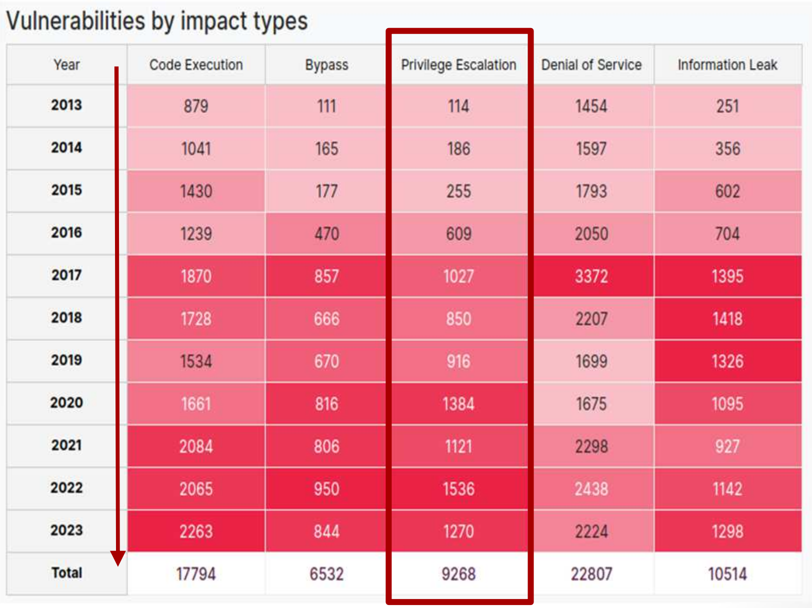
Customised for VELO

Introduction

Android apps security & Vulnerabilities

Security Attacks remain a prominent issue for mobile app users.





Vulnerabilities by impact types

Year	Code Execution	Bypass	Privilege Escalation	Denial of Service	Information Leak
2013	879	111	114	1454	251
2014	1041	165	186	1597	356
2015	1430	177	255	1793	602
2016	1239	470	609	2050	704
2017	1870	857	1027	3372	1395
2018	1728	666	850	2207	1418
2019	1534	670	916	1699	1326
2020	1661	816	1384	1675	1095
2021	2084	806	1121	2298	927
2022	2065	950	1536	2438	1142
2023	2263	844	1270	2224	1298
Total	17794	6532	9268	22807	10514

https://www.cvedetails.com/browse-by-date.php

https://www.cvedetails.com/vulnerabilities-by-types.php

Goal

Automatically detect security design flaws in source code of Android apps

Outline

- Context and contribution
- Proposal
- Experimentation
- PrivDroid
- Conclusion

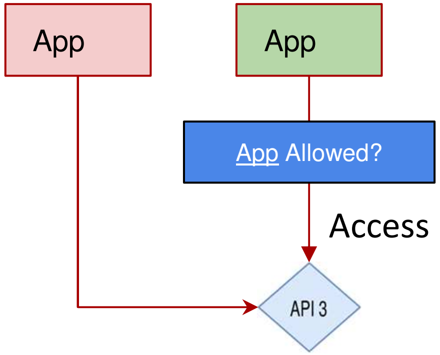
Background

Privilege Escalation

- Privileges are authorization access given to an Android application in order to access system resources (APIs).

- Authorizations in Android are manipulated through the concept of **Permission**

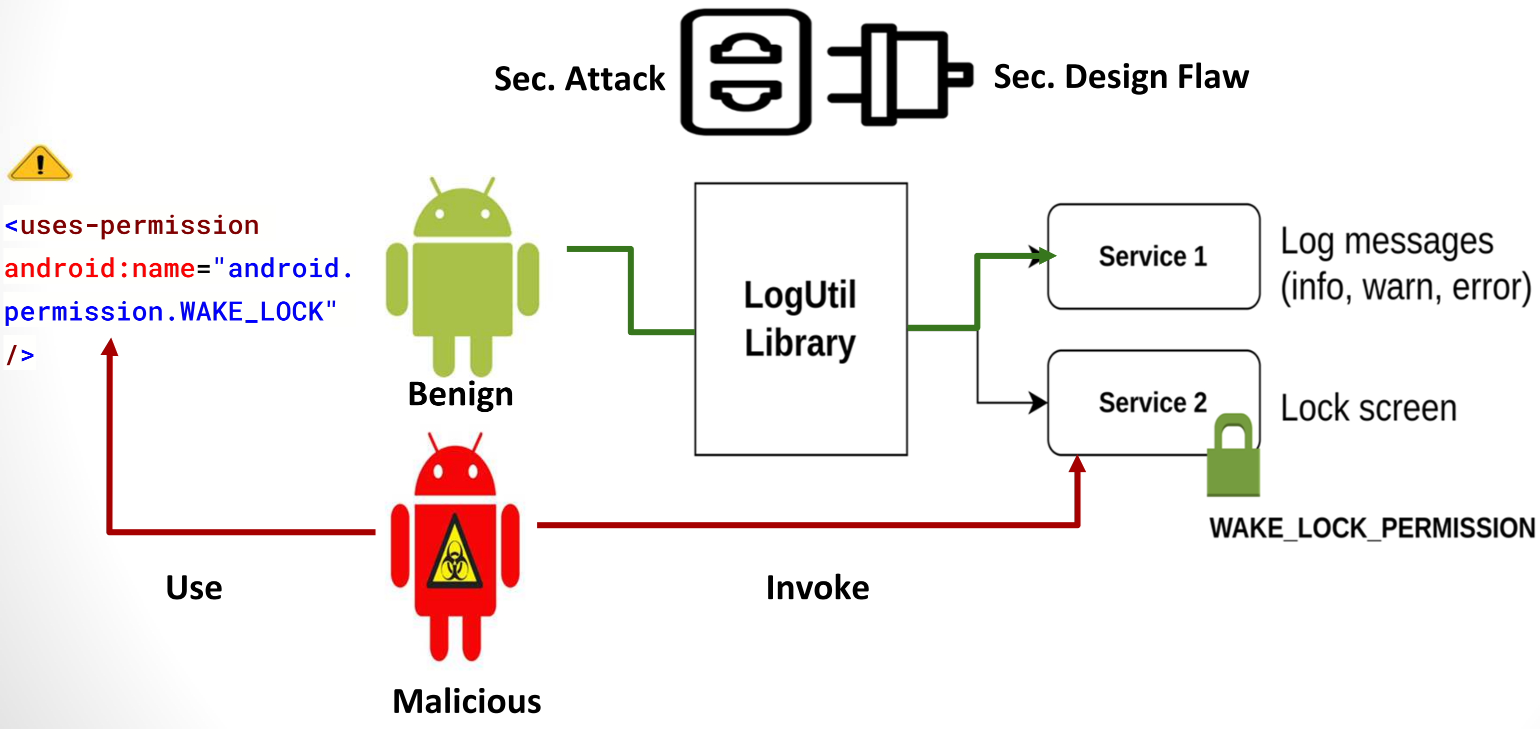
- Privilege Escalation (PE) is a type of security **exploit** in which a user **gains unauthorized permissions** to access resources and carry out **malicious actions.**



Context

From design flaws to privilege escalation attacks

Developers play a crucial role in securing their applications [R. Balebako et al. 2017; Scoccia; SCAM, 2019]

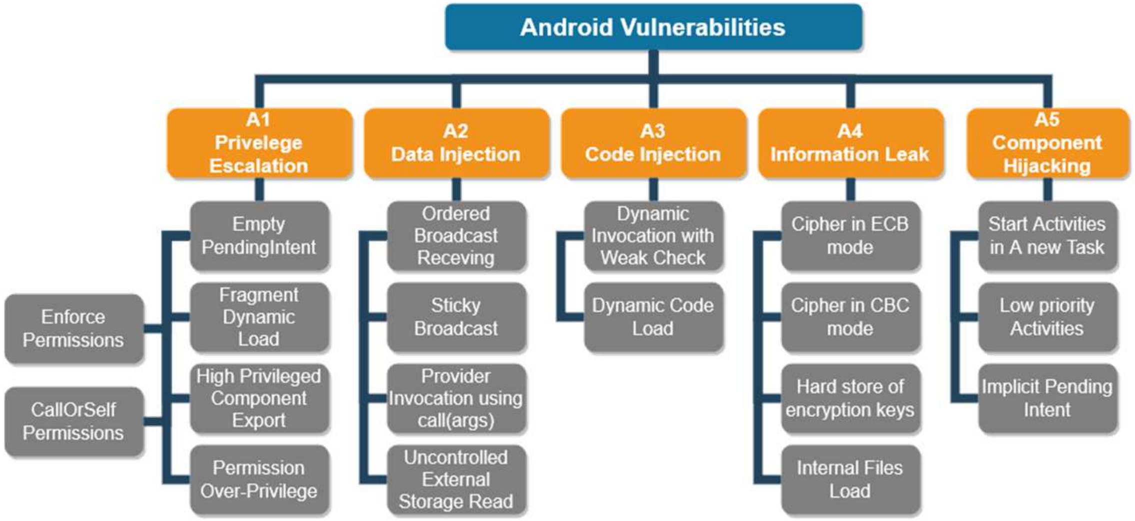


```
<uses-permission
android:name="android.
permission.WAKE_LOCK"
/>
```

State Of the Art

Are the existing ide plugins effective in detecting known vulnerabilities?

16 Tools vs 19 Vulnerabilies



Android Vulnerabilities

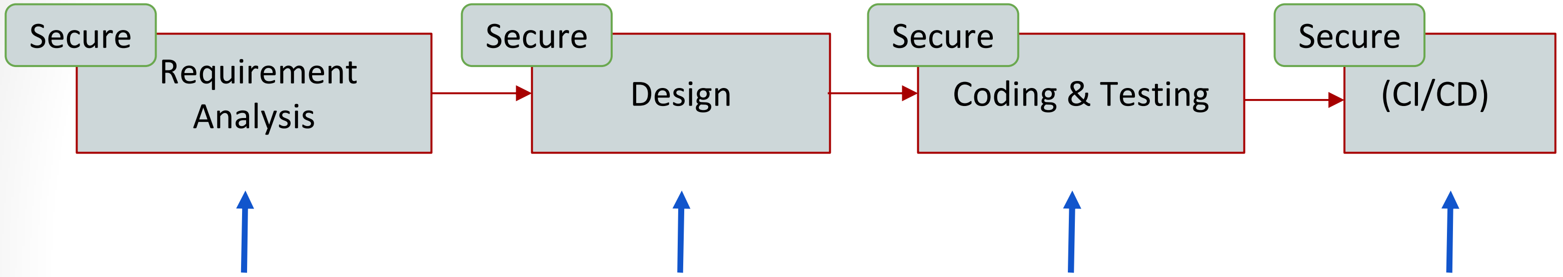
- A1 Privelege Escalation
  - Empty PendingIntent
  - Fragment Dynamic Load
  - High Privileged Component Export
  - Permission Over-Privilege
  - Enforce Permissions
  - CallOrSelf Permissions
- A2 Data Injection
  - Ordered Broadcast Receving
  - Sticky Broadcast
  - Provider Invocation using call(args)
  - Uncontrolled External Storage Read
- A3 Code Injection
  - Dynamic Invocation with Weak Check
  - Dynamic Code Load
- A4 Information Leak
  - Cipher in ECB mode
  - Cipher in CBC mode
  - Hard store of encryption keys
  - Internal Files Load
- A5 Component Hijacking
  - Start Activities in A new Task
  - Low priority Activities
  - Implicit Pending Intent

Open Access

Tebib, M. E. A, et. al. (**2023**). A Survey on Secure Android Apps Development Life-Cycle: Vulnerabilities and Tools. ***International Journal On Advances in Security***, *16*(1 & 2), 54-71.

Context

Secure Software Development Pipeline



Page [rowan2014]

PolDroidAS [slavin2016]

Coconut [li2018]

Sema [mitra2020]

Vandroid [nirumand19]

PerHelper [xu2019]

Curbing [vidas2011]

sonarqube

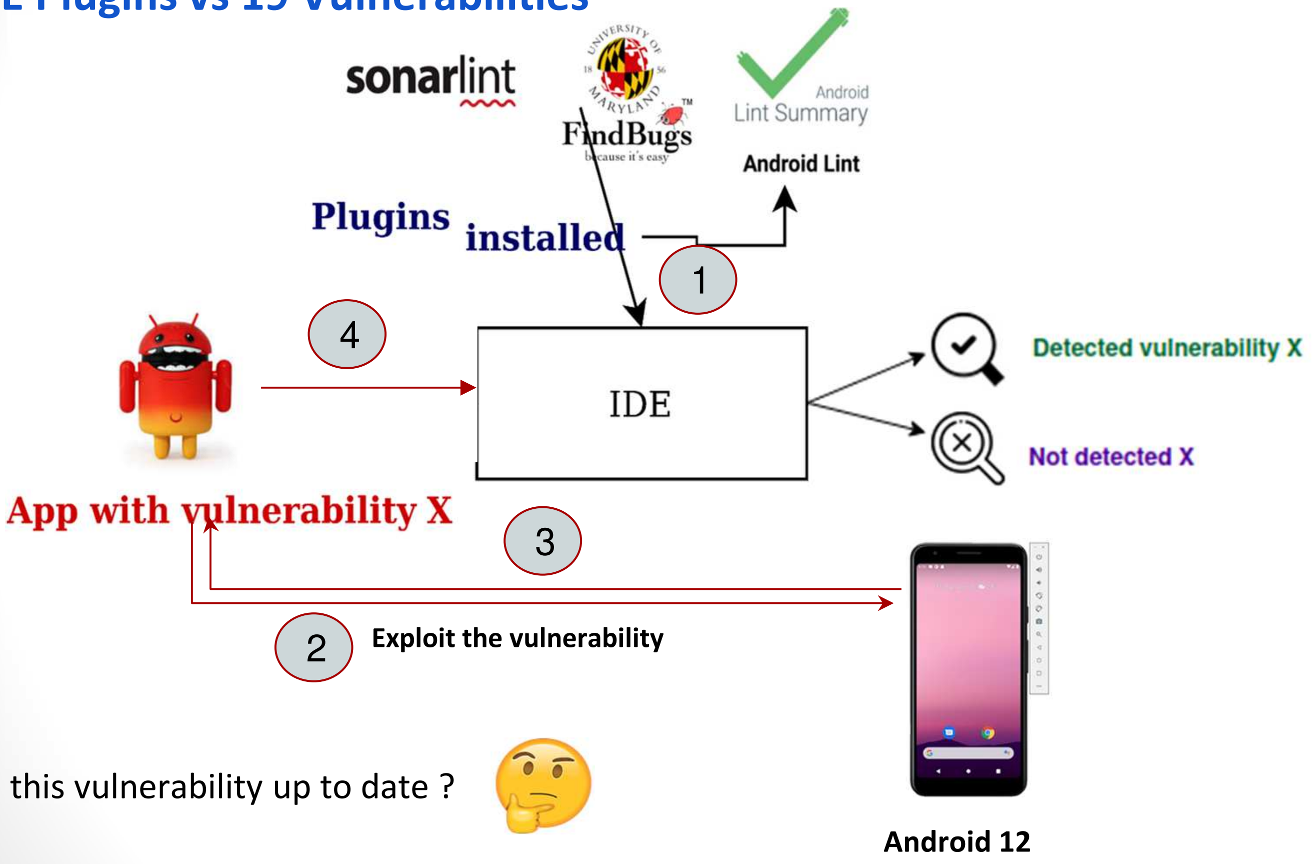
snyk

FindBugs
because it's easy

[Github CodeQL]

Investigation

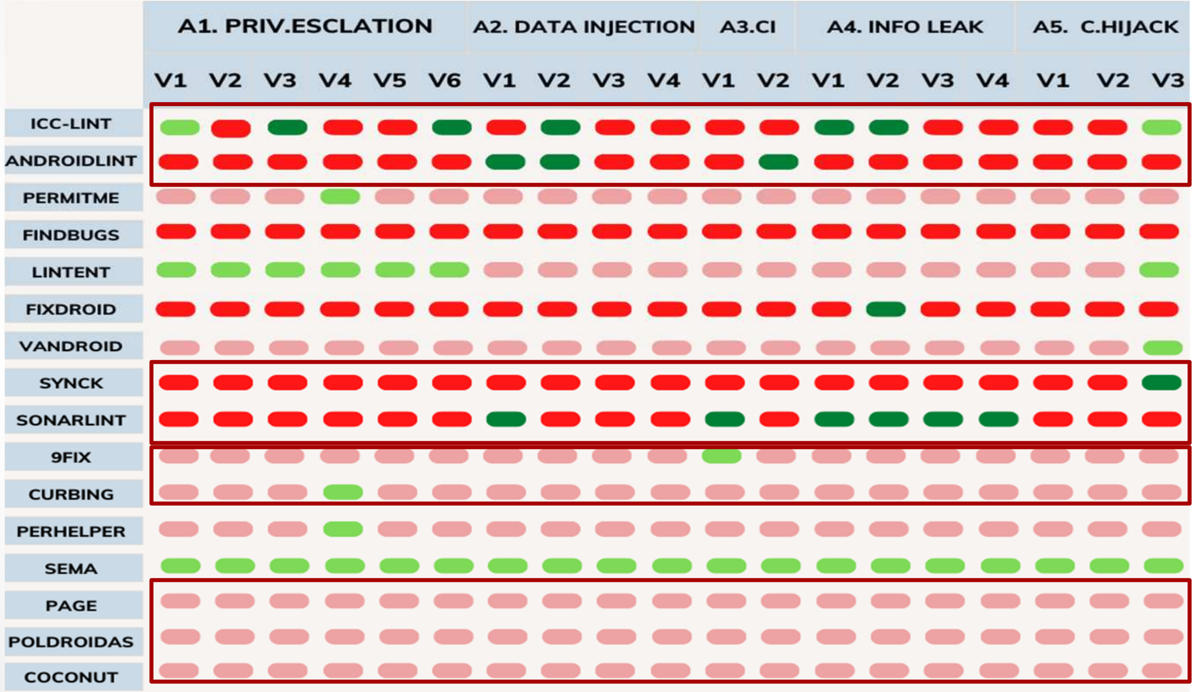
16 IDE Plugins vs 19 Vulnerabilities



Is this vulnerability up to date ? 🤔

State Of the Art

Are the existing ide plugins effective in detecting known vulnerabilities?



	A1. PRIV.ESCLATION						A2. DATA INJECTION				A3.CI		A4. INFO LEAK				A5. C.HIJACK		
	V1	V2	V3	V4	V5	V6	V1	V2	V3	V4	V1	V2	V1	V2	V3	V4	V1	V2	V3
ICC-LINT	🟢	🟥	🟩	🟥	🟥	🟩	🟥	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟥	🟢
ANDROIDLINT	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥
PERMITME	🔴	🔴	🔴	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
FINDBUGS	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
LINTENT	🟢	🟢	🟢	🟢	🟢	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🟢
FIXDROID	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥
VANDROID	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🟢
SYNCK	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩
SONARLINT	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩	🟥	🟩	🟩	🟩	🟩	🟥	🟥	🟥
9FIX	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
CURBING	🔴	🔴	🔴	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
PERHELPER	🔴	🔴	🔴	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
SEMA	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢	🟢
PAGE	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
POLDROIDAS	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
COCONUT	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴

Documentation

🔴

🟢

Experimentation

🟥 False Negative

🟩 Positive

Weak analysis capabilities!

Open Access

Tebib, M. E. A, et. al. (**2023**). A Survey on Secure Android Apps Development Life-Cycle: Vulnerabilities and Tools. ***International Journal On Advances in Security***, *16*(1 & 2), 54-71.

Motivation

Our survey highlighted the following limitations:

- **Lack of availability**. None of these tools is available to be used in real projects
- **Outdated** existing solutions, due to the evolution of permissions and APIs
- **Uncomplete static analysis approaches**: Java Reflection, Native Code, etc.
- **Low Analysis Precision**: existing tools ignore the **api Level** dimension during over-privilege analysis.

Example. API level significance during over-privilege analysis

before > api 29 (Android 10)

```
android.permission.WRITE_EXTERNAL_STORAGE
```

after <= api 29 (Android 10)

```
android.permission.READ_EXTERNAL_STORAGE
or
android.permission.WRITE_EXTERNAL_STORAGE
```

Contribution

We enrich the arsenal of tools used for Android development security.

- **PrivDroid**, an **up to date** and **available** IDE plugin (IntellIj/Android Studio)
- We defined 9 security code smell to reduce the attack surface related to PE
- **PrivDroid** combines static analysis techniques such as **Patterns** for Abstract Syntax Tree (AST) analysis and **Call Graph (CG)** to detect PE vulnerabilities.

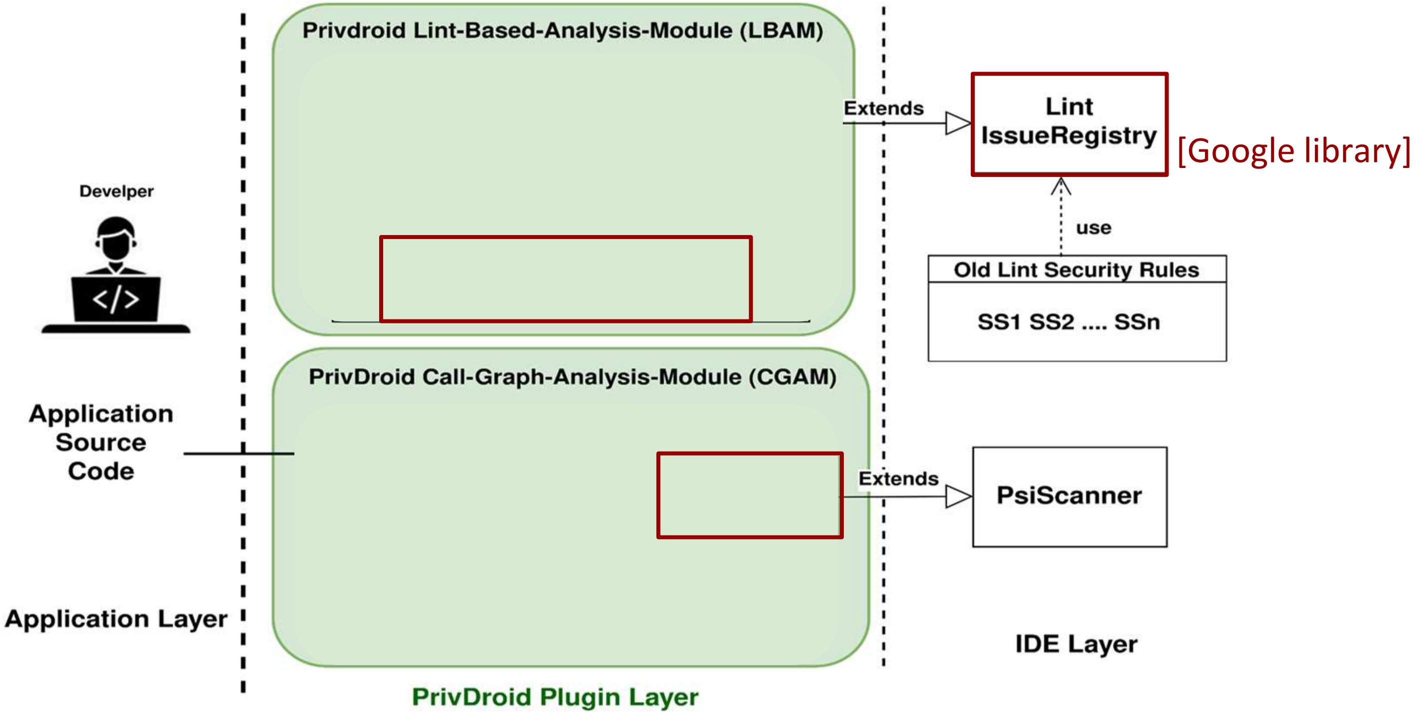
Outline

- Context and contribution
- Proposal
- Experimentation
- PrivDroid
- Conclusion

Proposal

PrivDroid Architecture

Modular Architecture → Ease extensibility



Proposal

PrivDroid Architecture

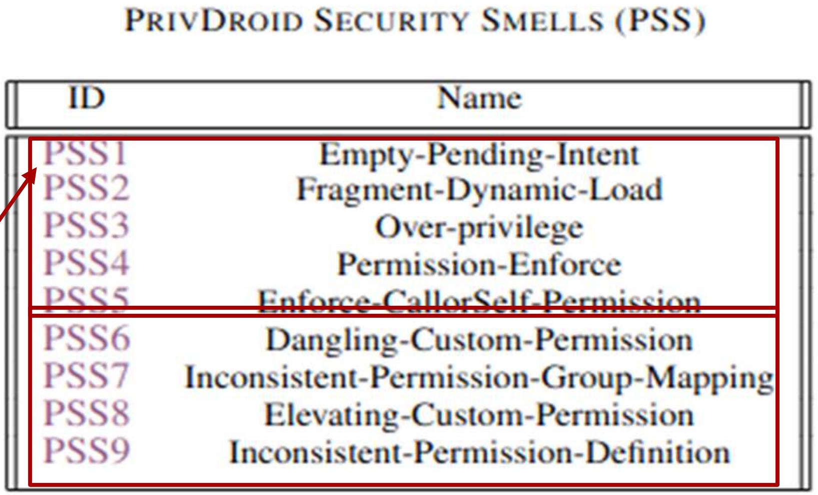
- ➢ **Lint Based Analysis Module**
- ➢ Call Graph Analysis Module

Lint Based Analysis Module

LBAM

Two additional capabilities:

- **Better Analysis Code Coverage** Better Analysis Code Coverage
- Novel Security Code Smells



PRIVDROID SECURITY SMELLS (PSS)

ID	Name
PSS1	Empty-Pending-Intent
PSS2	Fragment-Dynamic-Load
PSS3	Over-privilege
PSS4	Permission-Enforce
PSS5	Enforce-CallorSelf-Permission
PSS6	Dangling-Custom-Permission
PSS7	Inconsistent-Permission-Group-Mapping
PSS8	Elevating-Custom-Permission
PSS9	Inconsistent-Permission-Definition

PSS1. Empty PendingIntent

PendingIntent pendingIntent = PendingIntent.getActivity(context, requestCode, **intent**, flags);

Patterns:

new Inten();	Intent i = new Intent();	Intent i;
		i = new Intent();
			Intent i = new Intent();
			i.setAction('action');
				Inner Class

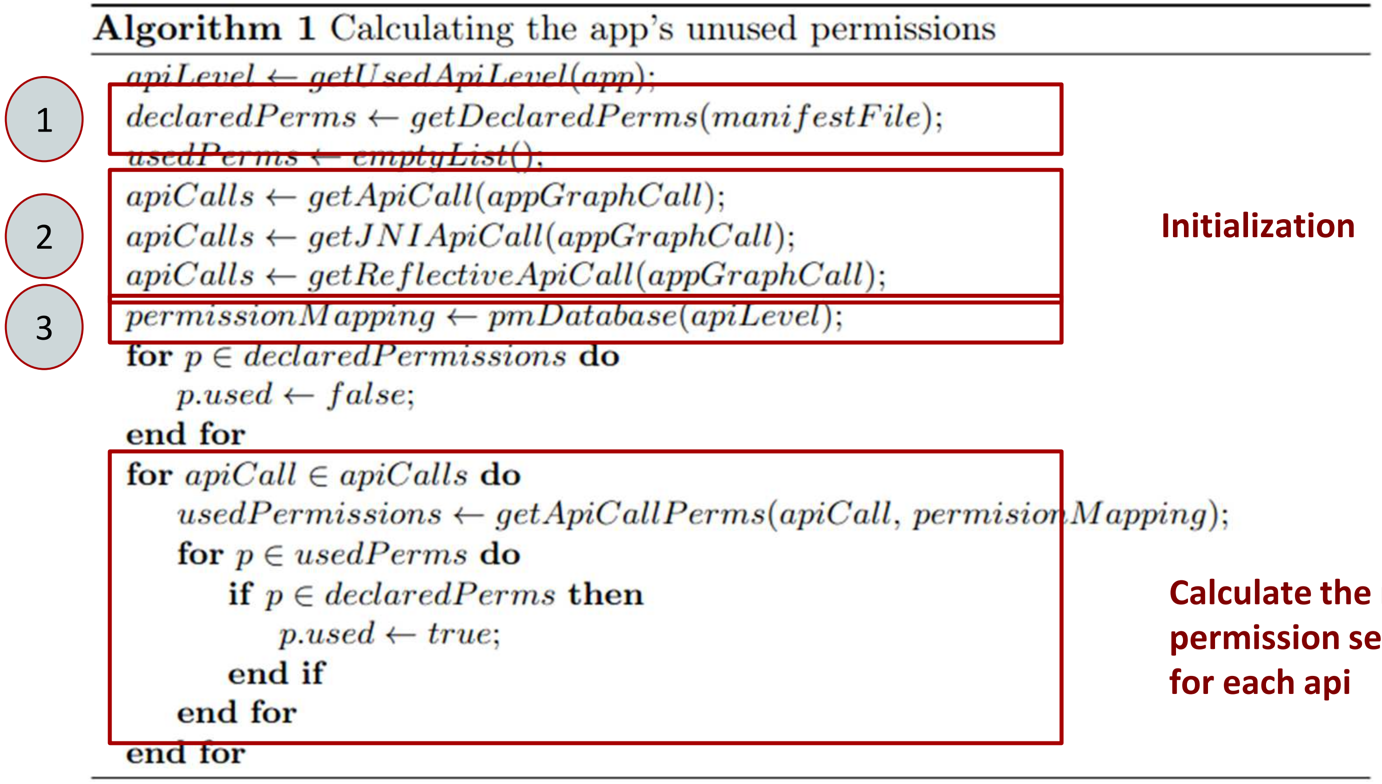
Proposal

PrivDroid Architecture

- ➢ Lint Based Analysis Module
- ➢ **Call Graph Analysis Module**

Call Graph Analysis Module

CGAM: How PrivDroid detects over-privileged applications?



Algorithm 1 Calculating the app's unused permissions

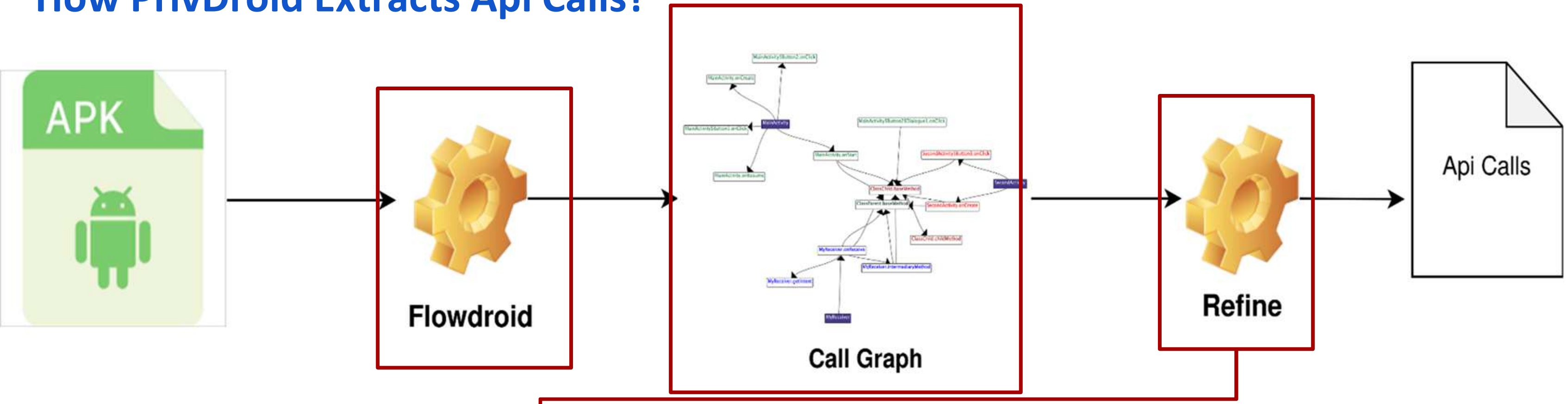
```
$apiLevel \leftarrow getUsedApiLevel(app);$
$declaredPerms \leftarrow getDeclaredPerms(manifestFile);$
$usedPerms \leftarrow emptyList();$
$apiCalls \leftarrow getApiCall(appGraphCall);$
$apiCalls \leftarrow getJNIApiCall(appGraphCall);$
$apiCalls \leftarrow getReflectiveApiCall(appGraphCall);$
$permissionMapping \leftarrow pmDatabase(apiLevel);$
for $p \in declaredPermissions$ do
    $p.used \leftarrow false;$
end for
for $apiCall \in apiCalls$ do
    $usedPermissions \leftarrow getApiCallPerms(apiCall, permisionMapping);$
    for $p \in usedPerms$ do
        if $p \in declaredPerms$ then
            $p.used \leftarrow true;$
        end if
    end for
end for
```

Initialization

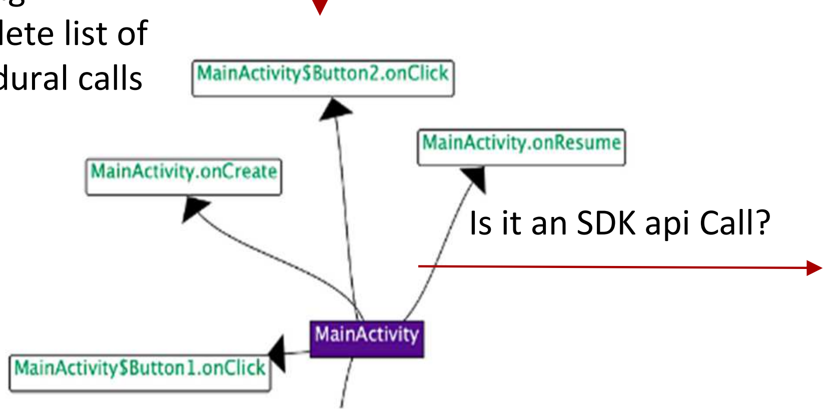
Calculate the minimum permission set required for each api

Call Graph Analysis Module

How PrivDroid Extracts Api Calls?



- Data-Tainting
- More complete list of intra-procedural calls

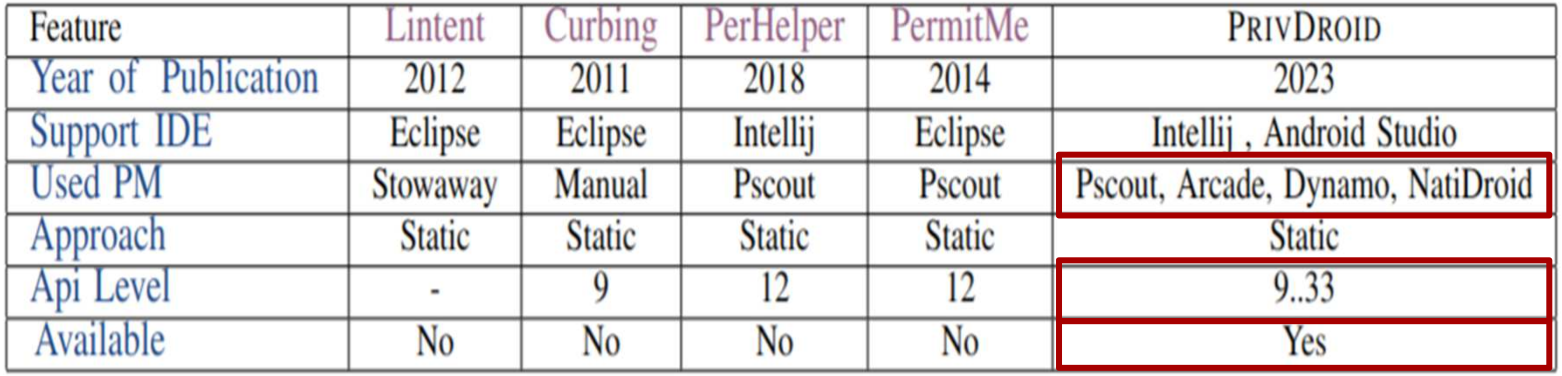


Edge.tgt()

ANDROID_PACKAGE = android.*
ANDROIX_PACKAGE = androidx.*
GOOGLE_ANDROID_PACKAGE=com.google.android.*

Graph Call Analysis Module

CGAM: Additional Analysis Capabilities



Feature	Lintent	Curbing	PerHelper	PermitMe	PrivDroid
Year of Publication	2012	2011	2018	2014	2023
Support IDE	Eclipse	Eclipse	Intellij	Eclipse	Intellij , Android Studio
Used PM	Stowaway	Manual	Pscout	Pscout	Pscout, Arcade, Dynamo, NatiDroid
Approach	Static	Static	Static	Static	Static
Api Level	-	9	12	12	9..33
Available	No	No	No	No	Yes

PrivDroid vs State of the art tools in detecting over-privileges applications

- Update Permission Mapping DB
- PM per api level
- Native and Reflection Calls detection
- Available

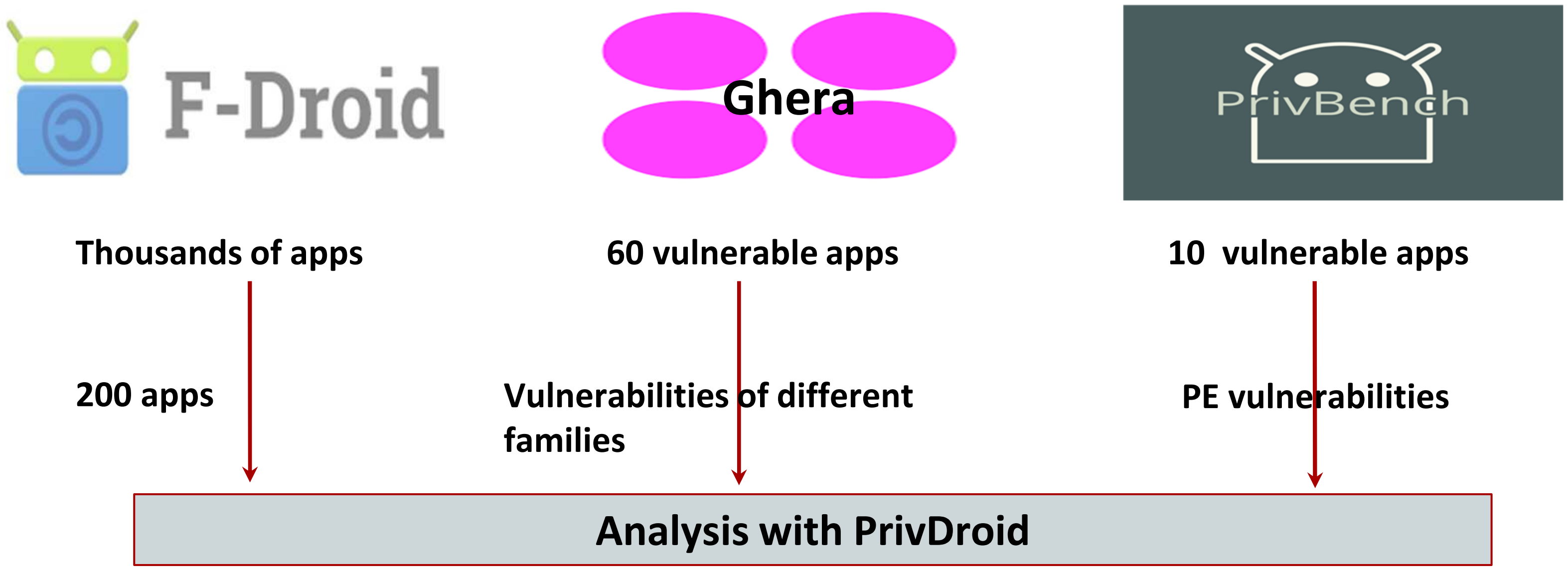
Outline

- Context and contribution
- Proposal
- Experimentation
- PrivDroid

Experimenting PrivDroid

3 datasets: open-source apps analyzable with ide plugins

There are very few established benchmarks available for open-source vulnerable applications!



Thousands of apps

200 apps

60 vulnerable apps

Vulnerabilities of different families

10 vulnerable apps

PE vulnerabilities

Analysis with PrivDroid

14 apps with PE

100% TP

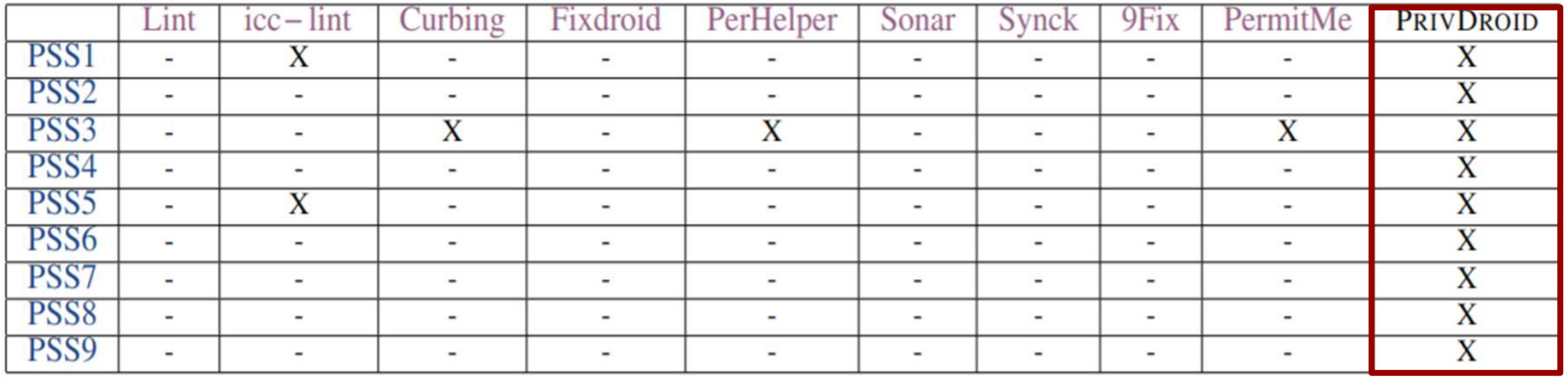
8 apps with PE

100% TP

10 apps with PE

100% TP

Experimenting PrivDroid



	Lint	icc – lint	Curbing	Fixdroid	PerHelper	Sonar	Synck	9Fix	PermitMe	PRIVDROID
PSS1	-	X	-	-	-	-	-	-	-	X
PSS2	-	-	-	-	-	-	-	-	-	X
PSS3	-	-	X	-	X	-	-	-	X	X
PSS4	-	-	-	-	-	-	-	-	-	X
PSS5	-	X	-	-	-	-	-	-	-	X
PSS6	-	-	-	-	-	-	-	-	-	X
PSS7	-	-	-	-	-	-	-	-	-	X
PSS8	-	-	-	-	-	-	-	-	-	X
PSS9	-	-	-	-	-	-	-	-	-	X

PrivDroid analysis code coverage for PE security smells

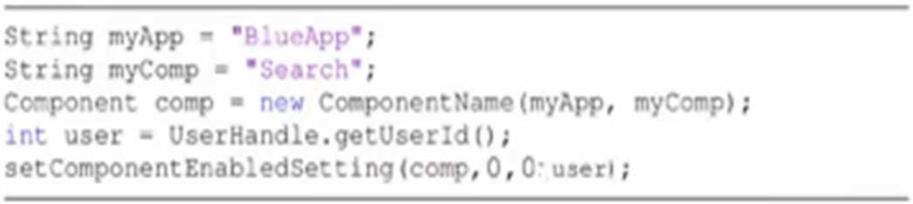
PrivDroid

Permission Mappin Per Api Level

Context of api calls instead of only api calls is required (Permission as a set is imprecise [arcade])

UserChecks

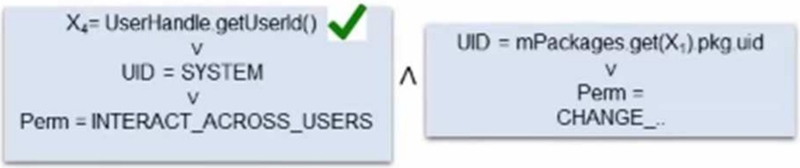
UidChecks



```
String myApp = "BlueApp";
String myComp = "Search";
Component comp = new ComponentName(myApp, myComp);
int user = UserHandle.getUserId();
setComponentEnabledSetting(comp,0,0: user);
```

setComponentEnabledSetting() :: android.permission.CHANGE_COMPONENT_ENABLED_STATE, android.permission.INTERACT_ACROSS_USERS_FULL

aXplorer



X_4= UserHandle.getUserId() ✓
v
UID = SYSTEM
v
Perm = INTERACT_ACROSS_USERS

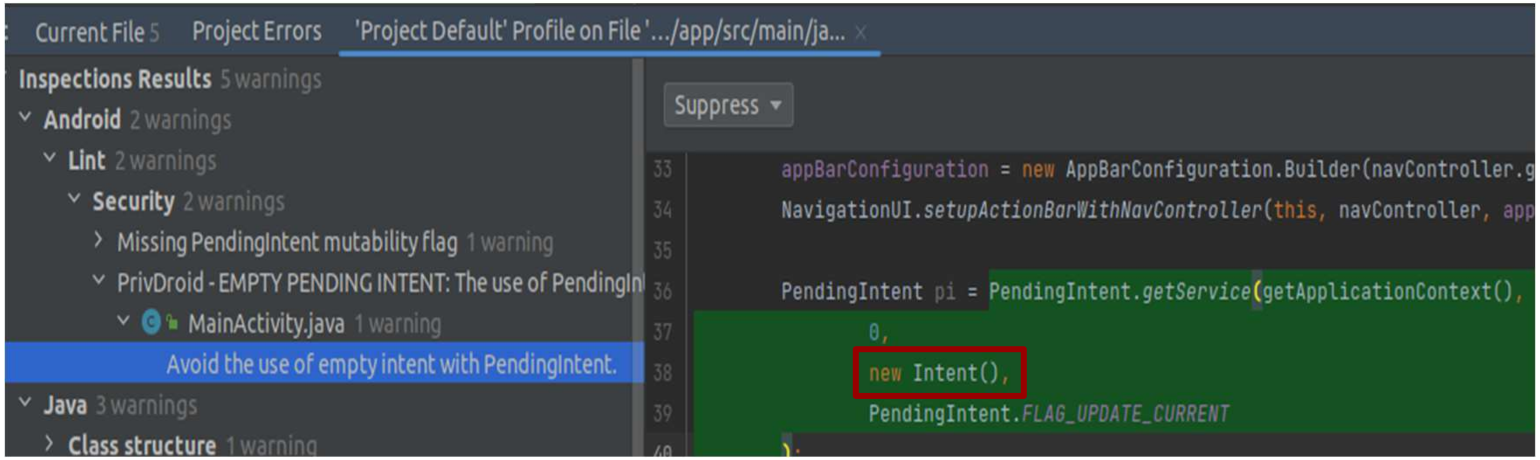
∧

UID = mPackages.get(X_1).pkg.uid
v
Perm =
CHANGE_..

Arcade

[arcade] Aafer, Y., Tao, G., Huang, J., Zhang, X., & Li, N. (2018, October). Precise Android API protection mapping derivation and reasoning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1151-1164).

PrivDroid In Action



Current File 5 | Project Errors | 'Project Default' Profile on File '.../app/src/main/ja...

- Inspections Results 5 warnings
  - Android 2 warnings
    - Lint 2 warnings
      - Security 2 warnings
        - Missing PendingIntent mutability flag 1 warning
        - PrivDroid - EMPTY PENDING INTENT: The use of PendingIn
          - MainActivity.java 1 warning
            - Avoid the use of empty intent with PendingIntent.
  - Java 3 warnings
    - Class structure 1 warning

```
        appBarConfiguration = new AppBarConfiguration.Builder(navController.g
        NavigationUI.setupActionBarWithNavController(this, navController, app

        PendingIntent pi = PendingIntent.getService(getApplicationContext(),
                0,
                new Intent(),
                PendingIntent.FLAG_UPDATE_CURRENT
        );
```

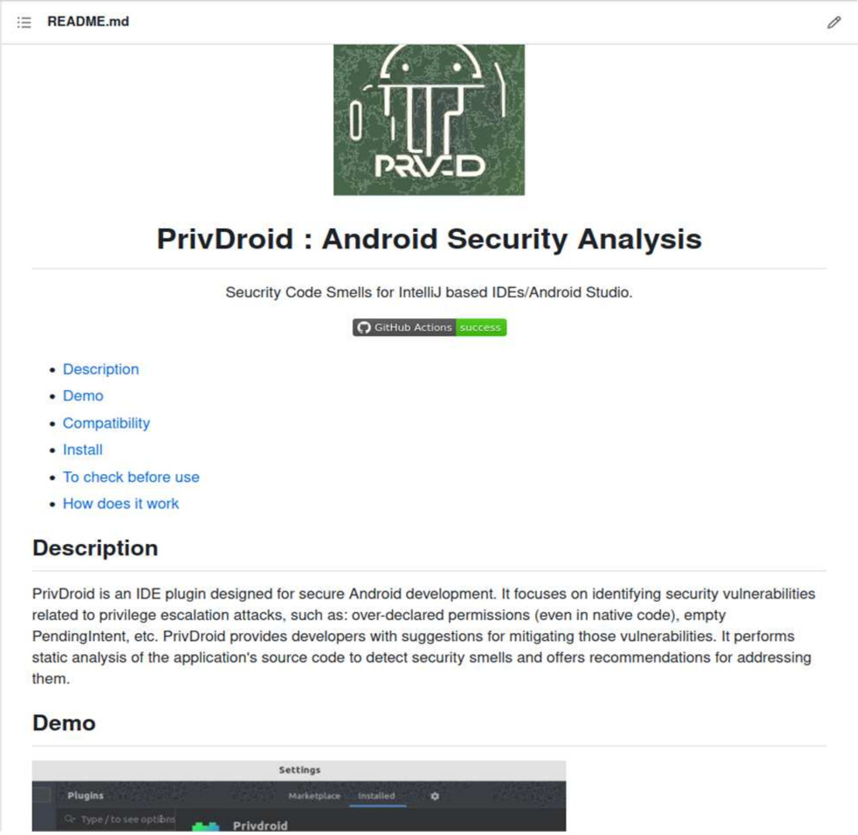
PrivDroid analysis result: a vulnerable app with PSS1 vulnerabilitiy

Outline

- Context and contribution
- Proposal
- Experimentation
- PrivDroid
- Conclusion

PrivDroid is available!

PrivDroid: Android Security Code Smells for Privilege Escalation Prevention



PrivDroid : Android Security Analysis

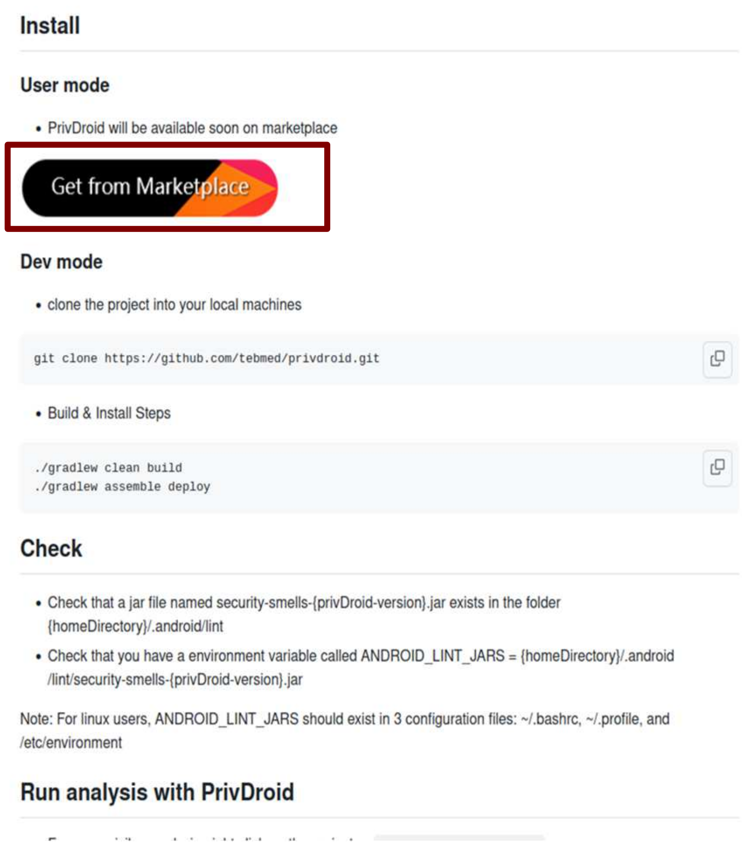
Seucrity Code Smells for IntelliJ based IDEs/Android Studio.

- Description
- Demo
- Compatibility
- Install
- To check before use
- How does it work

Description

PrivDroid is an IDE plugin designed for secure Android development. It focuses on identifying security vulnerabilities related to privilege escalation attacks, such as: over-declared permissions (even in native code), empty PendingIntent, etc. PrivDroid provides developers with suggestions for mitigating those vulnerabilities. It performs static analysis of the application's source code to detect security smells and offers recommendations for addressing them.

Demo



Install

User mode

- PrivDroid will be available soon on marketplace

Dev mode

- clone the project into your local machines

```
git clone https://github.com/tebmed/privdroid.git
```

- Build & Install Steps

```
./gradlew clean build
./gradlew assemble deploy
```

Check

- Check that a jar file named security-smells-{privDroid-version}.jar exists in the folder {homeDirectory}/.android/lint
- Check that you have a environment variable called ANDROID_LINT_JARS = {homeDirectory}/.android /lint/security-smells-{privDroid-version}.jar

Note: For linux users, ANDROID_LINT_JARS should exist in 3 configuration files: ~/.bashrc, ~/.profile, and /etc/environment

Run analysis with PrivDroid

Outline

- Context and contribution
- Proposal
- Experimentation
- PrivDroid
- Conclusion

Conclusion

Summary

- PrivDroid helps developers to secure their applications by identifying potential security risks related to Privilege Escalation
- PrivDroid provides additional analysis capabilities: analysis code coverage, precise analysis of over-privileges application

Future work

- Continue the evaluation process of PrivDroid (large dataset, industrial context)
- Extend Privdroid detection capabilities with new vulnerabilities
- Consider Contextual API Call instead of Api Calls

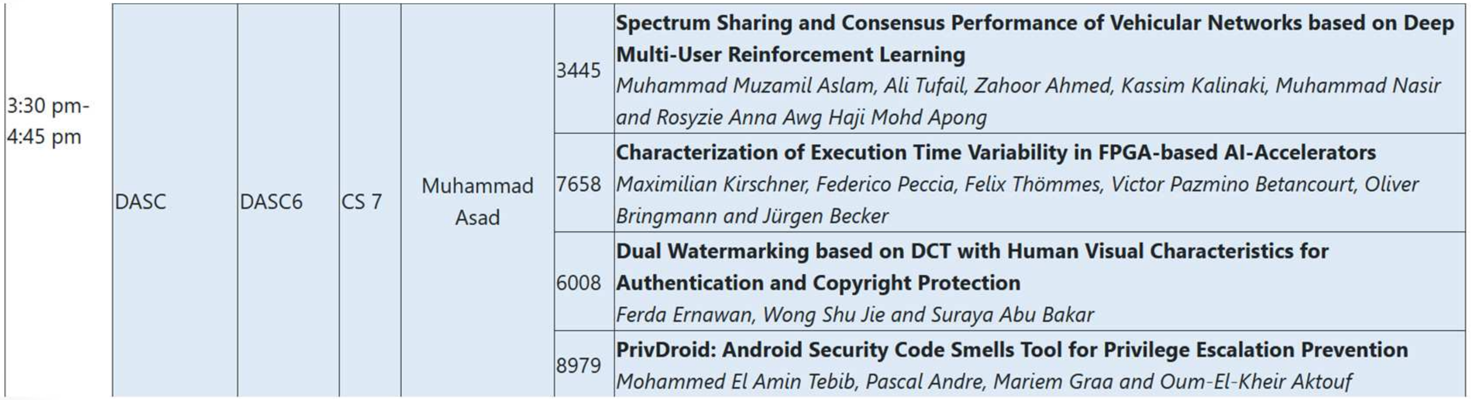
Source Code

https://github.com/tebmed/privdroid

Thanks for your attention!

PrivDroid: Android Security Code Smells for Privilege Escalation Prevention

- Wednesday, November 15th 2023



3:30 pm- 4:45 pm	DASC	DASC6	CS 7	Muhammad Asad	3445	**Spectrum Sharing and Consensus Performance of Vehicular Networks based on Deep Multi-User Reinforcement Learning** *Muhammad Muzamil Aslam, Ali Tufail, Zahoor Ahmed, Kassim Kalinaki, Muhammad Nasir and Rosyzie Anna Awg Haji Mohd Apong*
					7658	**Characterization of Execution Time Variability in FPGA-based AI-Accelerators** *Maximilian Kirschner, Federico Peccia, Felix Thömmes, Victor Pazmino Betancourt, Oliver Bringmann and Jürgen Becker*
					6008	**Dual Watermarking based on DCT with Human Visual Characteristics for Authentication and Copyright Protection** *Ferda Ernawan, Wong Shu Jie and Suraya Abu Bakar*
					8979	**PrivDroid: Android Security Code Smells Tool for Privilege Escalation Prevention** *Mohammed El Amin Tebib, Pascal Andre, Mariem Graa and Oum-El-Kheir Aktouf*

Lint

- Améliorer votre code avec des vérifications lint

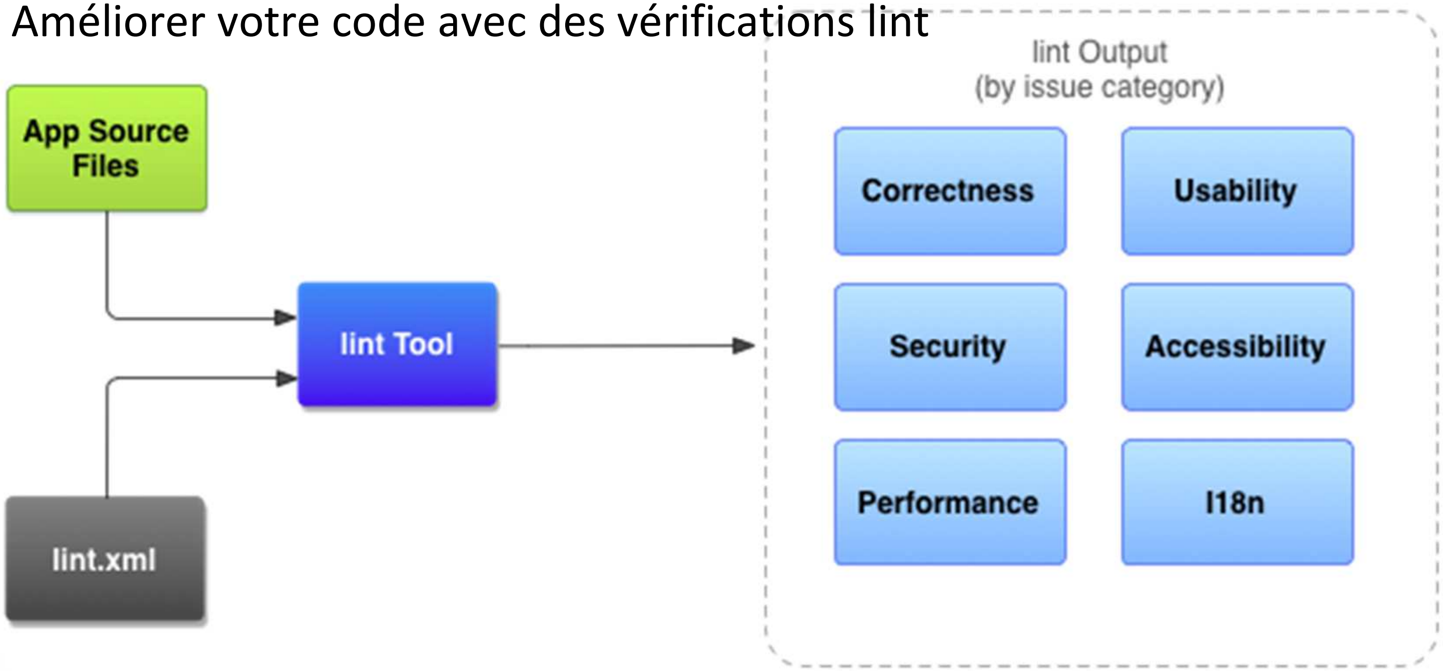
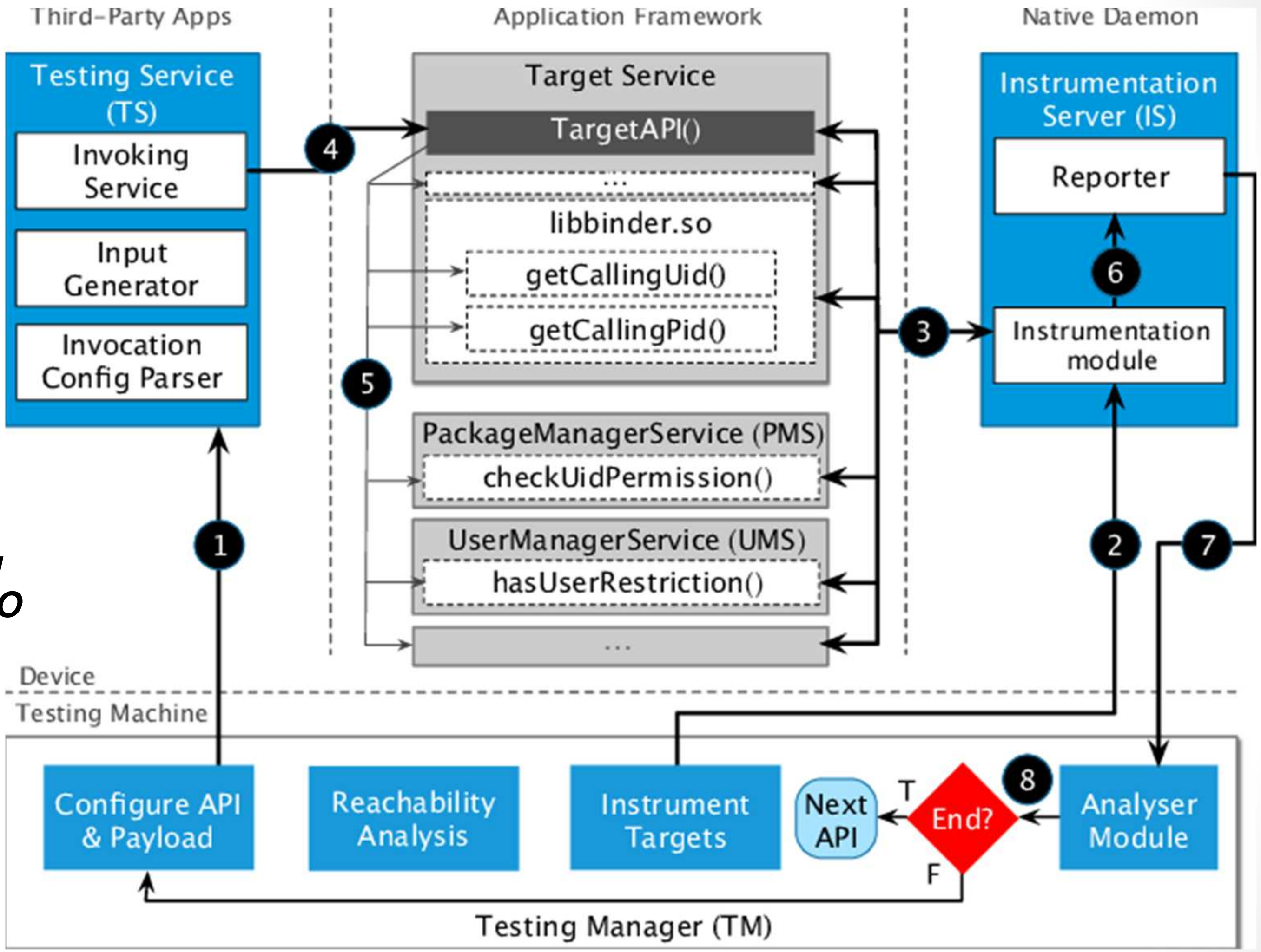


Figure 1. Workflow de lecture de code avec lint

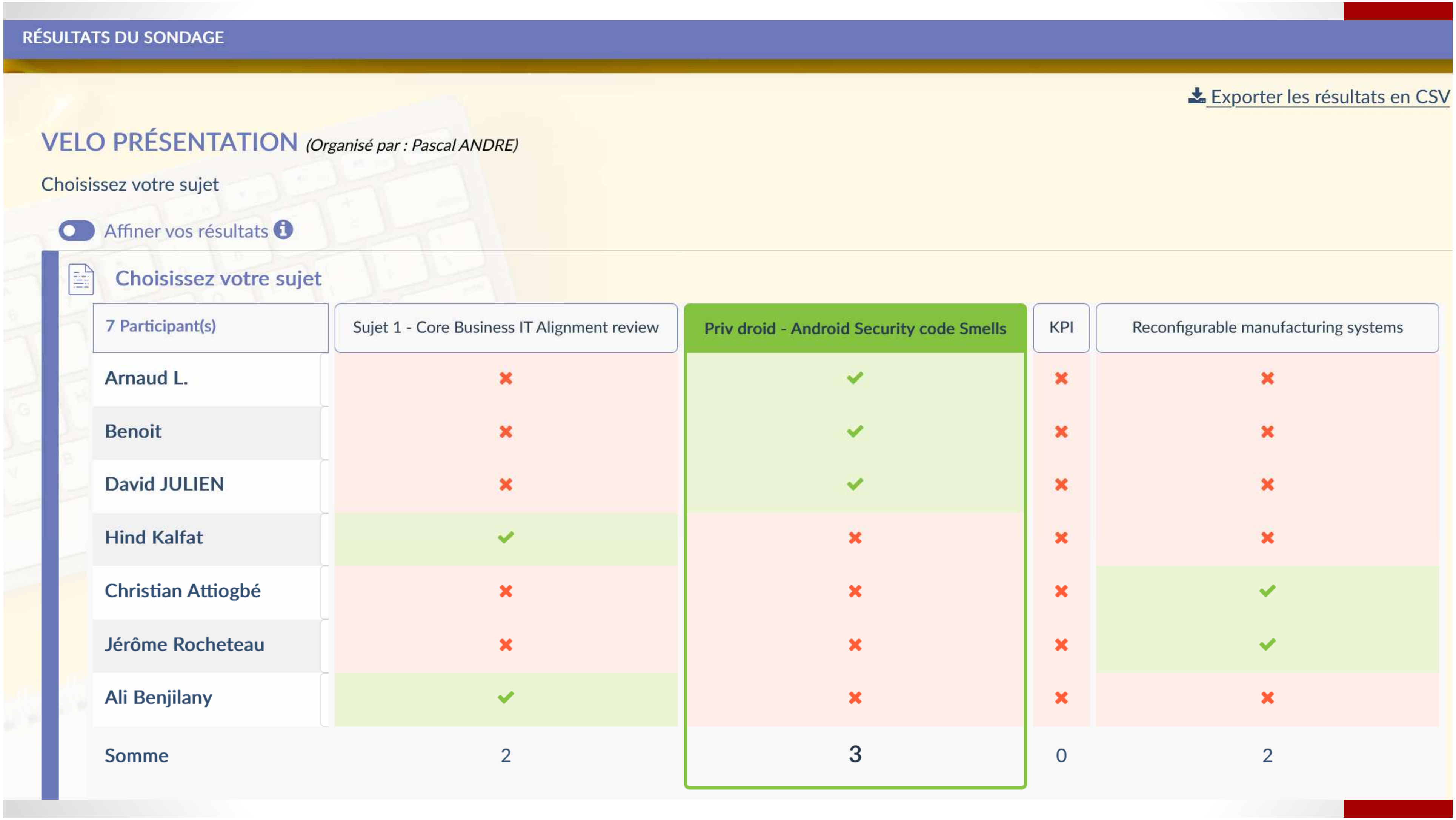
https://developer.android.com/studio/write/lint?hl=fr

DYNAMO



Steps of one testing iteration by DYNAMO

Abdallah Dawoud



RÉSULTATS DU SONDAGE

Exporter les résultats en CSV

VELO PRÉSENTATION *(Organisé par : Pascal ANDRE)*

Choisissez votre sujet

Affiner vos résultats

Choisissez votre sujet

7 Participant(s)	Sujet 1 - Core Business IT Alignment review	Priv droid - Android Security code Smells	KPI	Reconfigurable manufacturing systems
Arnaud L.	✗	✓	✗	✗
Benoit	✗	✓	✗	✗
David JULIEN	✗	✓	✗	✗
Hind Kalfat	✓	✗	✗	✗
Christian Attiogbé	✗	✗	✗	✓
Jérôme Rocheteau	✗	✗	✗	✓
Ali Benjilany	✓	✗	✗	✗
Somme	2	3	0	2