Android Compat Analysis Summary

Original Data Source:

A. Provided sets of installation and runtime logs.

1. Installation log – Benign 2018-2019, Malware 2018-2019 *(/media/SeagateBack/xqfu/InstallationReSults/ on the server 10.99.1.192)*
2. Runtime log – Benign 2018-2019, Malware 2010-2019 *(at /home/xqfu/runtimeResults on the server 10.99.1.192)*

B. Provided sets of source apks

1. /pool/home/zyzhang/myDownloads on 10.99.1.191
2. /media/SeagateBack/xqfu/myDownloads on 10.99.1.192
3. /storage_array/home/xqfu/andro-data/myDownloads on 10.99.1.190
4. /storage_array/home/xqfu/andro-data/AndroZoo on 10.99.1.190

C. Provided .csv file, each csv contains three apk identification and one app name that they belong to.

1. Benign 2018-2019 and Malware 2018-2019 (at /pool/home/zyzhang/myDownloads on 10.99.1.191)
   (NOTE that we should also need .csv 2010-2019 for both benign and malware)

NOTICE: $for\ all\ apk\ in\ A\ must\ \subseteq\ for\ all\ apk\ in\ B$, so to guarantee 100% coverage for SSPS of A, so far the coverage of SSPS of A << 100%.

Filtered Data Set:

D. ./InstallResult

- Contains a set of text files, each of which with the identity {apkYear, apiLevel}, that describe the total apks that are examined, the number of apks that are compatible, and the number of apks that are incompatible. Each compatible or incompatible apk is listed with its original identification number and further that each incompatible apk listed with its failure message.
- How do I obtain these text files? **NEED A.1 original source file**
  - bash ClassifierScript/runInstallTraces.sh **InstallationLogAddress**
  - will then saved a set of text files ./InstallResult (create folder if not existed)

E. ./RuntimeResult

- Contains a set of text files, each of which with the identity {apkYear, apilevel}, that describe the total apks that are examined, the number of apks that are compatible, and the number of apks that are incompatible. Each compatible or incompatible apk is listed with its original identification number and further that

each incompatible apk listed with its failure message.

- How do I obtain these txt file? **NEED A.2 original source file**
    - python ClassifierScript/runtime_classify.py **RuntimeLogAddress**
    - will then saved a set of text files ./RuntimeResult (create folder if not existed)

F. ./DataParse/malware-minsdk
./DataParse/benign-minsdk

- Each folder contains set of text files, each of which with identity {apkType, apkYear}, describe each apk with its minSDKlevel.
- How do I obtain these text file? **NEED B original source file**
    - python DataParse/parseMinSdk.py FolderContainsApks "apkType" "apkYea"
    - will then saved the results in a folder at ./DataParse/

G. ./MultuCompatResult

- Contains a set of text files that describe apps that are multi-compatible (meaning at least some of its belonging apks are compatible with covered API19, 20-27) or multi-incompatible (all of its belonging apks did not compatible cover API19, 20-27).
- How do I obtain these text file? **NEED C and (D or E)**
    - python MultiClassify .csvDir InstallResult/RuntimeResultDir
    - will then saved the results in a folder at ./MultiCompatResult/

Tables and Figures

H. Data/Tables.xlsx

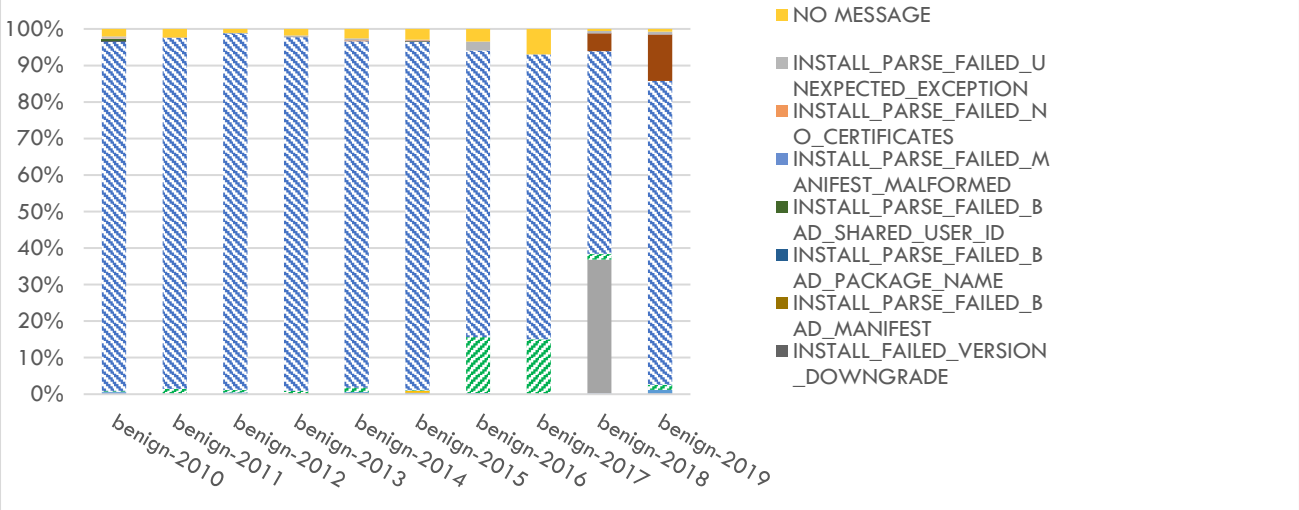
- How do I obtain these data?
    - Python TableOneStat.py A.1orA.2dir
    - Print to console then manually collect

Benign Data use	number of samples from each year within 2010-2019										Total
	2010	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	
Installation-time incompatibility study	16835	9977	10991	9688	5300	5406	2431	2266	37838	30236	130968
Run-time incompatibility study	1531	2020	2054	1750	1335	327	1548	1680	13171	11787	37203

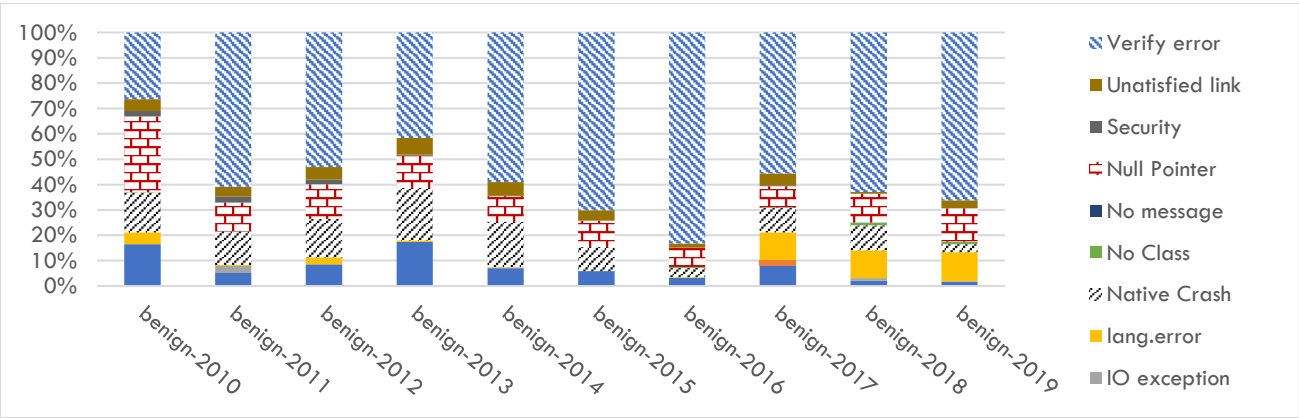
Malware Data use	number of samples from each year within 2010-2019										Total
	2010	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	
Installation-time incompatibility study									32523	20270	52793
Run-time incompatibility study	16173	11226	11731	12094	14418	11411	10845	11926	12245	12205	124274

I. Data/Bar(completed)-Benign.xlsx

- How do I obtain these data?
  - Python BarStatIns.py InstallResult
  - Print to console then manually collect

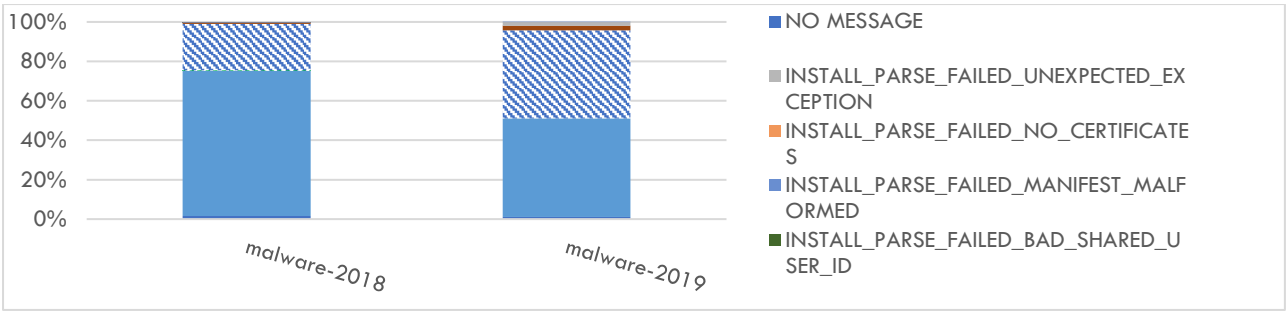


- How do I obtain these data?
  - Python BarStatRuntime.py RuntimeResult
  - Print to console then manually collect

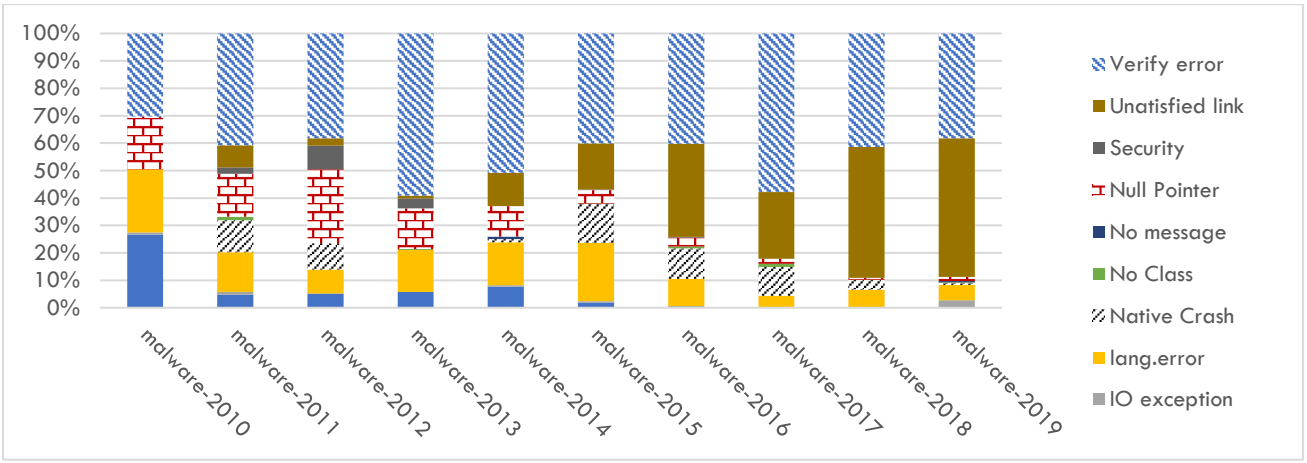


J. Data/Bar(completed)-Malware.xlsx

- How do I obtain these data?
  - Python BarStatIns.py InstallResult
  - Print to console then manually collect

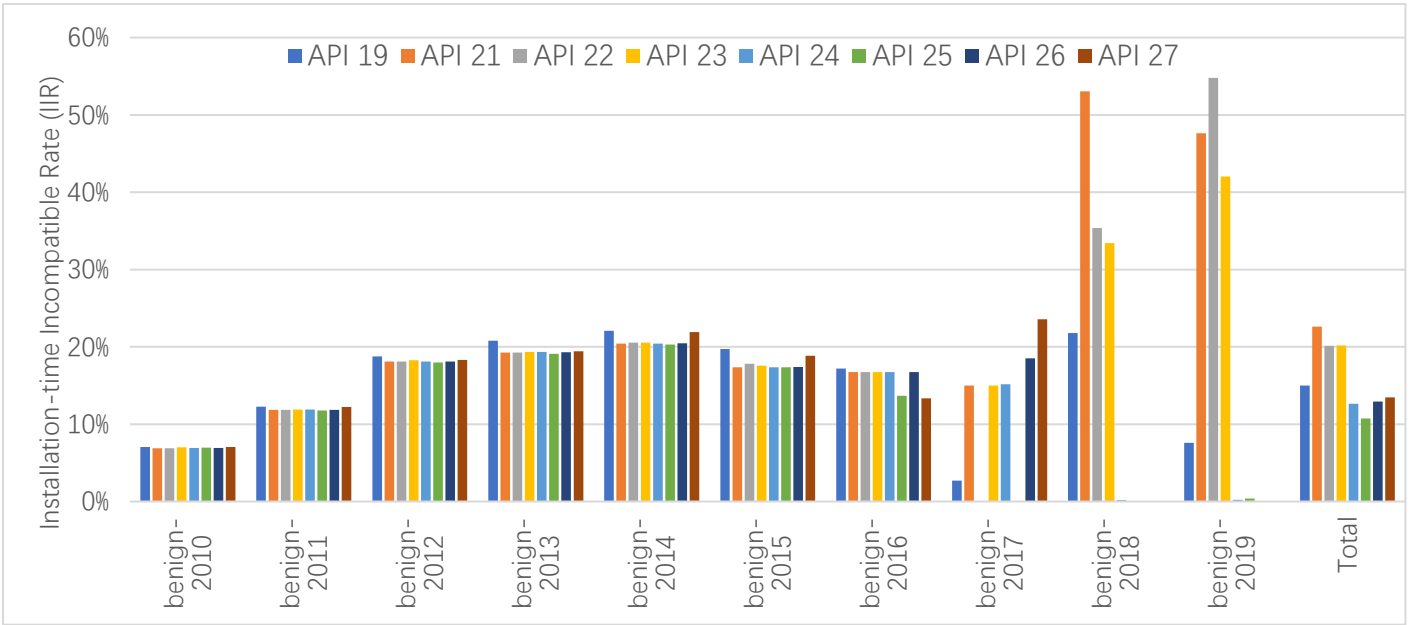


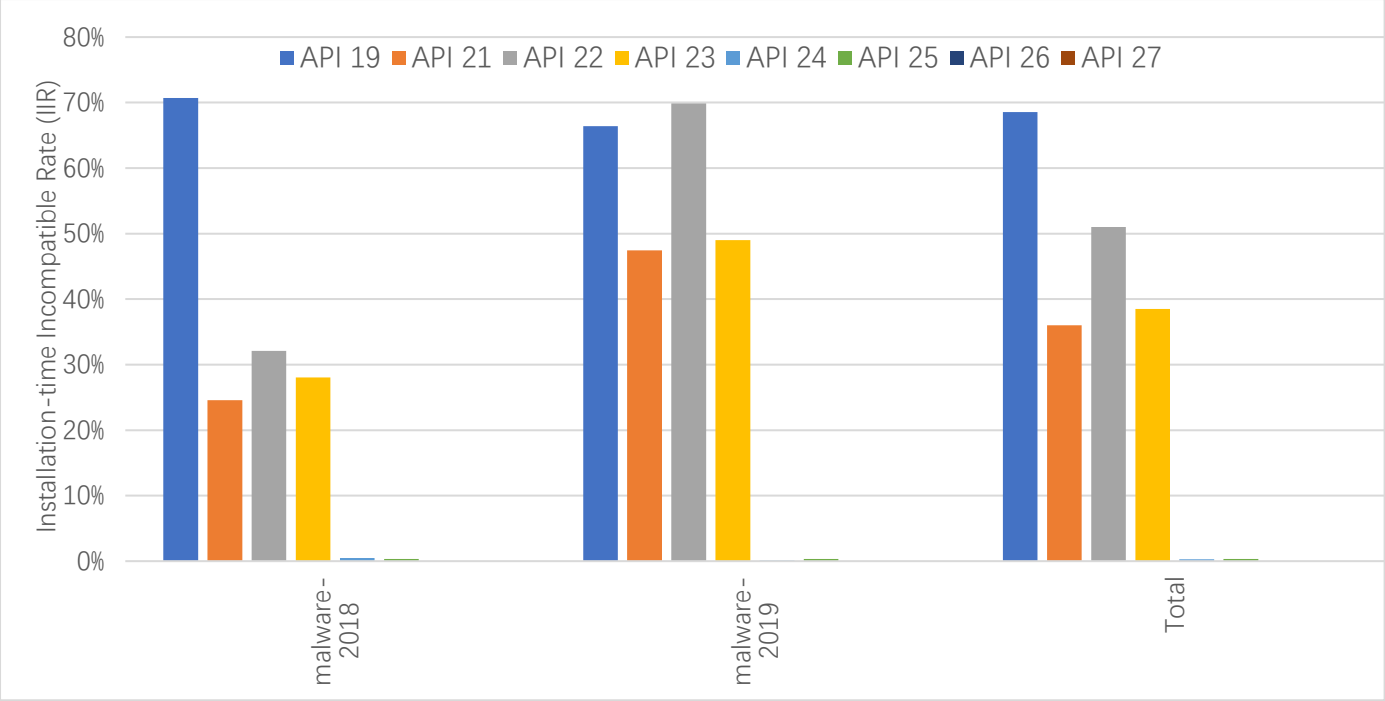
- How do I obtain these data?
  - Python BarStatRuntime.py RuntimeResult
  - Print to console then manually collect



K. Data/InstallationIIR(completed)-Benign.xslx
Data/InstallationIIR(completed)-Malware.xslx

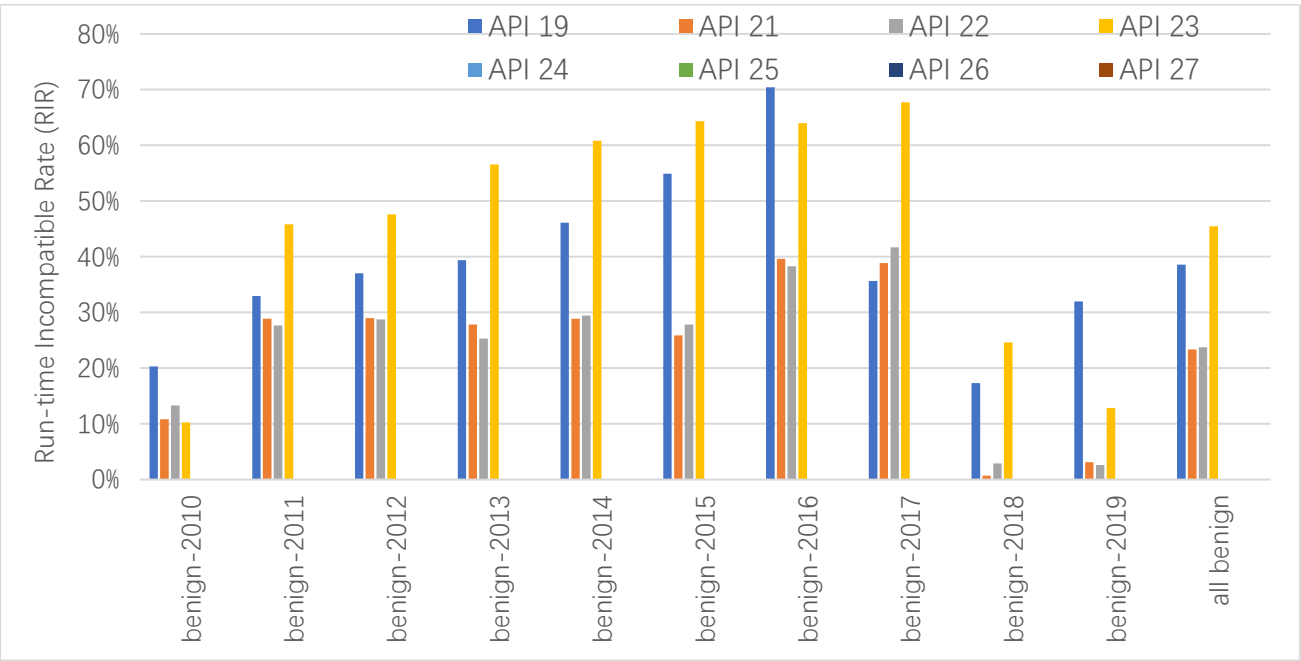
- Python Stats.py InstallResult
- Print to console then manually collect

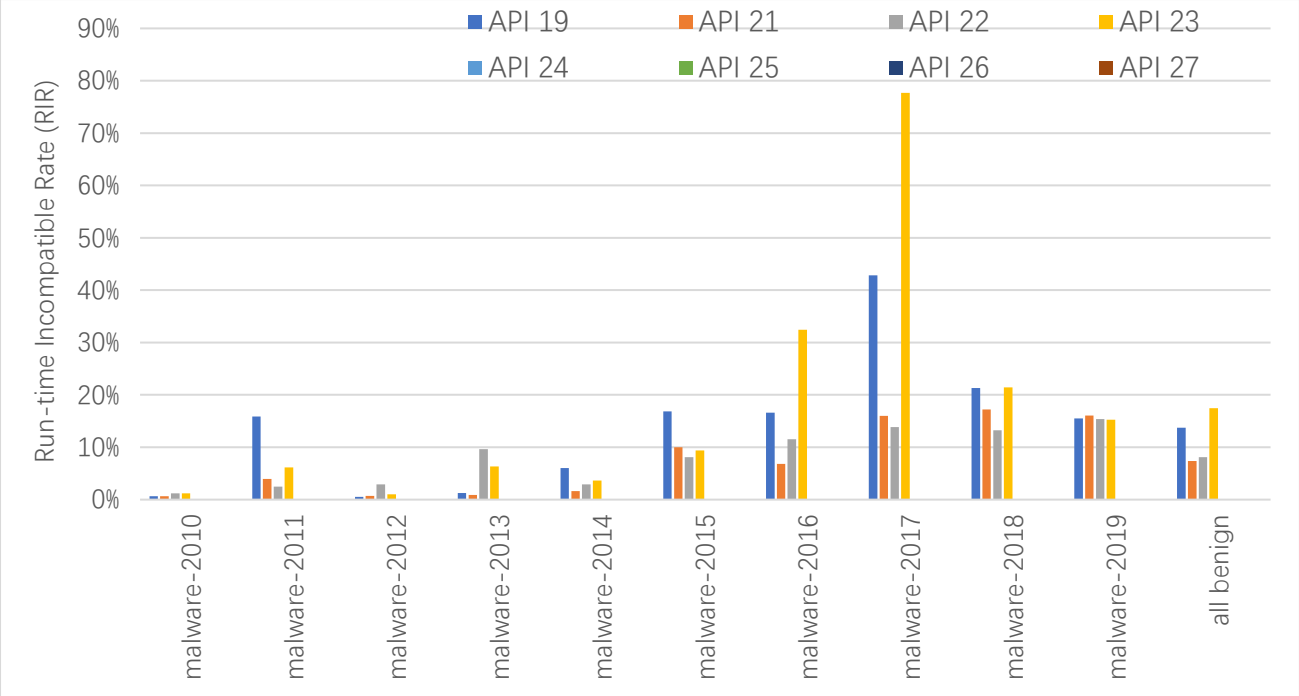




L. Data/RuntimeIIR(completed)-Benign.xlsx
Data/RuntimeIIR(completed)-Malware.xlsx

- Python Stats.py RuntimeResult
- Print to console then manually collect





M. SSPS/*.m

- Set of matlab script that generate the figures according the *.xlsx excel files in the current directory.

SSPS/*.xlsx

- Excel files for SSPS distribution.
- How do I obtain these data? **NEED A and C**
  - Python SSPSwrapper.py
  - Note that A and C dir are hardcoded.
- Generated:
  - SSPS/benignInstallTableABI.xlsx
  - SSPS/benignInstallTableLibrary.xlsx
  - SSPS/benignRuntimeTableVerify.xlsx
  - SSPS/benignRuntimeTableNative.xlsx
  - SSPS/malwareInstallTableABI.xlsx
  - SSPS/malwareInstallTableLibrary.xlsx
  - SSPS/malwareRuntimeTableVerify.xlsx
  - SSPS/malwareRuntimeTableNative.xlsx

The idea of SSPSwrapper:

1. Prepare Base Table Two options

   - FOR year [2010, 2019]:
     FOR api [19, 27] but not 20:
     FOR minsdk [0,28]:
     - { (minsdk, api, apkyear, apiyear):[#ofSuccess, #ofFail] }

   - FOR year [2018, 2019]:
     FOR api [19, 27] but not 20:
     FOR minsdk [0,28]:
     - { (minsdk, api, apkyear, apiyear):[#ofSuccess, #ofFail] }

2. Update the table value [#ofSuccess, #ofFail] per each text from A read.

3. Transform table to { (minsdk, api, apkyear, apiyear):[#total, #ofFail] }

4. Compute the failure rate from transformed table.

5. Write to excel.