Checking App Behavior Against App Descriptions

A. Gorla, I. Tavecchia, F. Gross, A. Zeller
Saarland University, May 2014

slide author names omitted for FERPA compliance

Questions

How many of you read the full description of a mobile app before downloading it?

Even if we read it, how do we know if the application does what it claims to do?

Current Problem

- Checking whether a program does what it claims to do is very difficult
- Is the app malware?
- Existing technique: Using predefined patterns of malicious behavior
  - New attacks?
  - Beneficial or malicious?

Beneficial or Malicious?

- An app that tracks your current position seems malicious
  - Not if it is a navigation app

- An app that takes all of your contacts and sends them to some server seems malicious
  - Not messaging apps, Snapchat, etc.

Research Questions

- By looking at the implementation and description of an application, can we effectively identify anomalies in Android applications?
  - i.e., mismatches between description and behavior

- Can this technique be used to identify malicious Android applications?

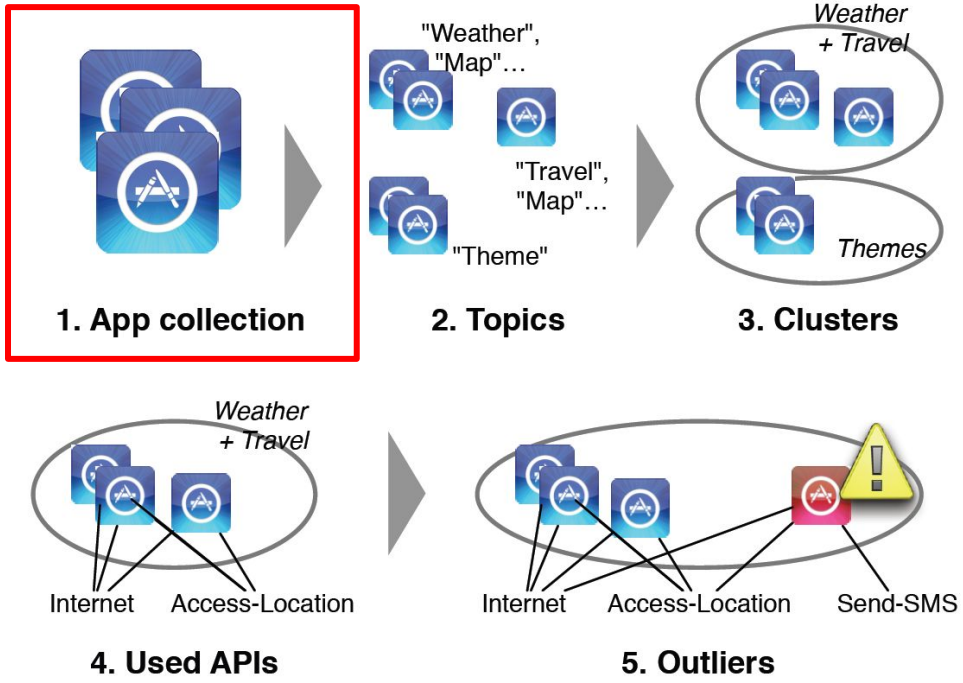
CHABADA

CHecking **A**pp **B**ehavior **A**gainst **D**escriptions of **A**pps

CHABADA != Ciabatta

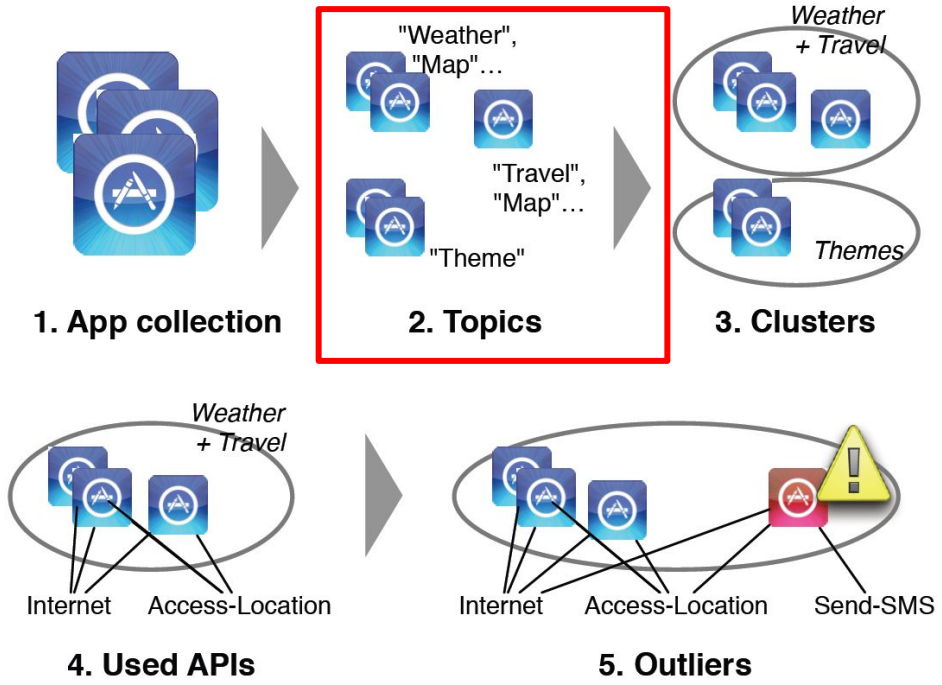
CHABADA - Step 1

CHABADA starts with a collection of 22,500+ “good” Android applications downloaded from the Google Play Store.



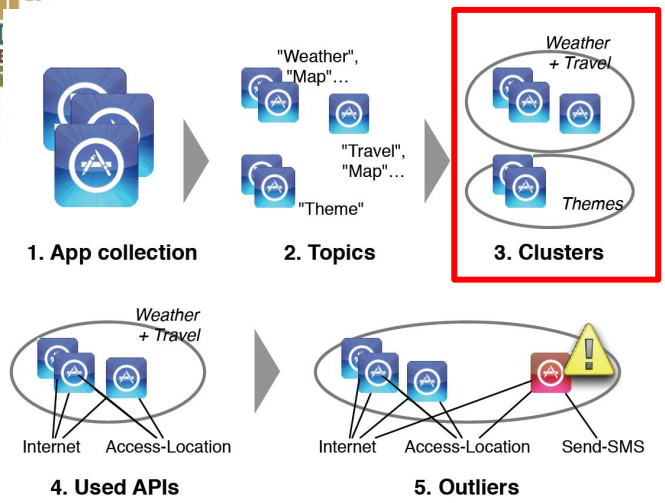
CHABADA - Step 2

Using Latent Dirichlet Allocation (LDA) on the app descriptions, CHABADA identifies the main topics (“theme”, “map”, “weather”, “download”, etc.) for each application.



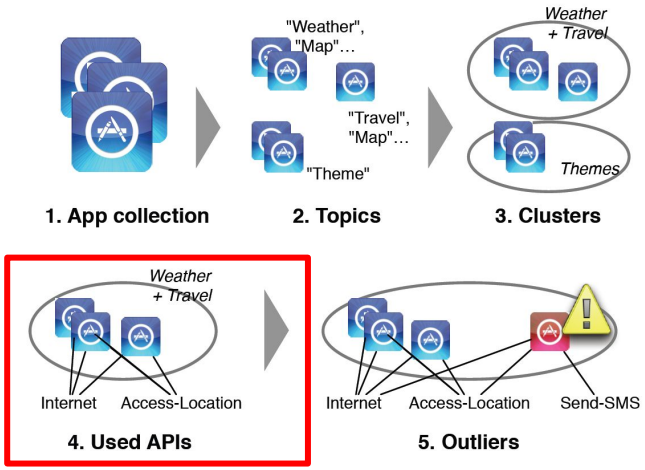
CHABADA - Step 3

CHABADA then clusters applications by related topic (“navigation” and “travel”)



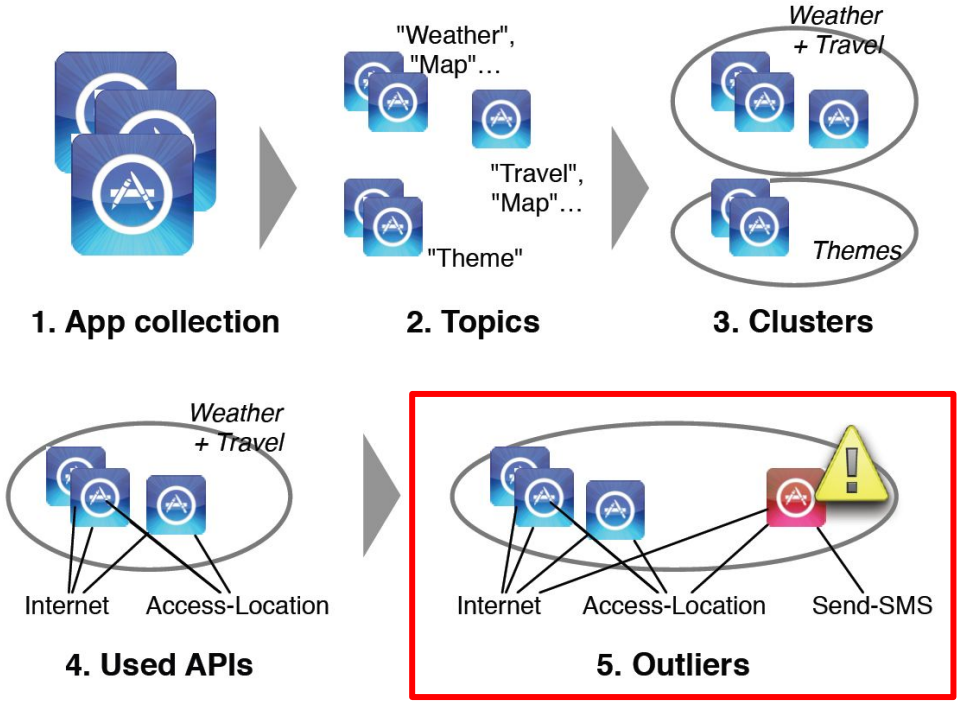
CHABADA - Step 4

In each cluster, CHABADA identifies the APIs each app statistically accesses.

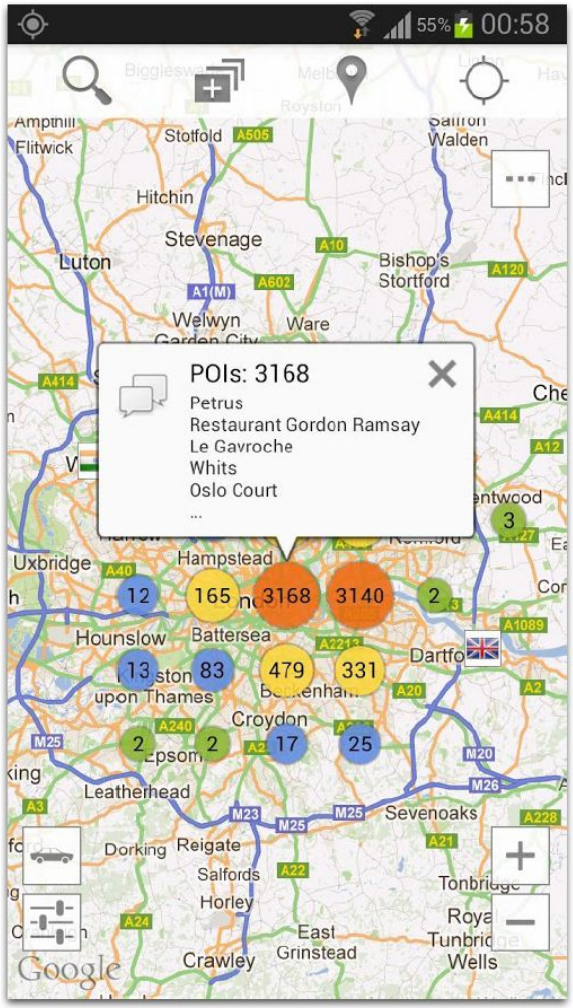


CHABADA - Step 5

Using unsupervised One-Class SVM anomaly classification, CHABADA identifies outliers with respect to API usage.



Example - London Restaurants App



Description

Looking for a restaurant, a bar, a pub or just to have fun in London? Search no more! This application has all the information you need:

* You can search for every type of food you want: french, british, chinese, indian etc.
* You can use it if you are in a car, on a bicycle or walking
* You can view all objectives on the map
* You can search objectives
* You can view objectives near you
* You can view directions (visual route, distance and duration)
* You can use it with Street View
* You can use it with Navigation

Keywords: london, restaurants, bars, pubs, food, breakfast, lunch, dinner, meal, eat, supper, street view, navigation

Example - London Restaurants App

- Easily put in the "Navigation and Travel" cluster
- API usage, however…

- "GET-ACCOUNTS" → getAccountsByType(), getDeviceID(), getLine1Number()
  - There goes your device id and phone number...

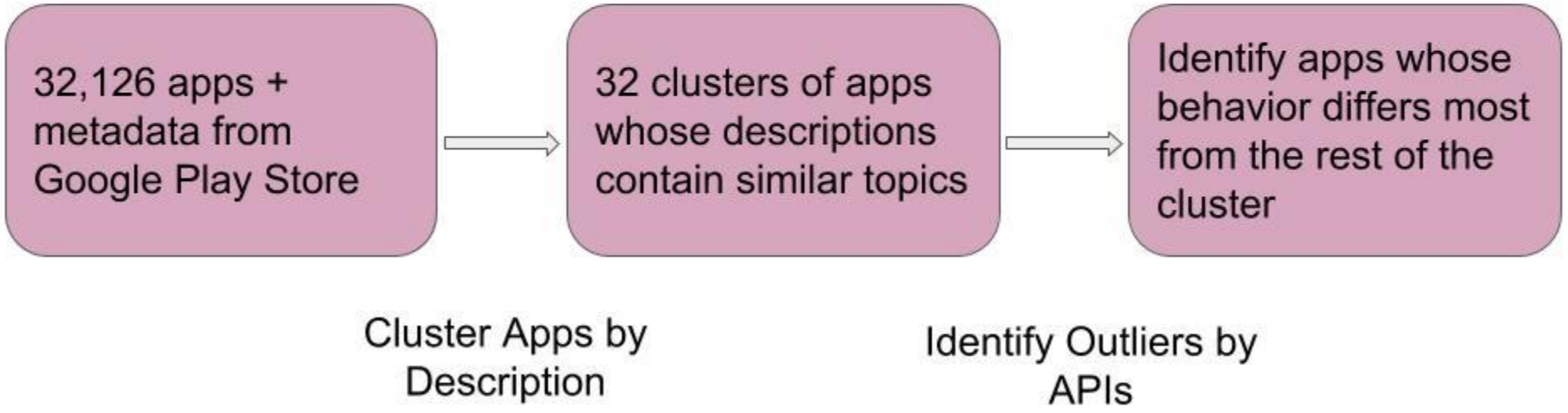
Key Point

- Is it malware?
  - Possibly?
- Is it unexpected behavior?
  - Certainly
- If the app description was explicit, it would have been in the "advertisements" cluster instead.
  - Not an outlier there

CHABADA identifies outliers based on their description and API usage. Red flag that tells you to look a little closer.

Idea

"Applications that are similar in terms of their description should also behave similarly."



Clustering Apps by Description

1. Preprocessing Descriptions with NLP
- only English
- Remove "stop words"
- Stemming
- Remove non-text (HTML links, e-mail addresses,…)

look restaur bar pub just fun london search applic inform need can search everi type food want french british chines indian etc can us car bicycl walk can view object map can search object can view object near can view direct visual rout distanc durat can us street view can us navig keyword london restaur bar pub food breakfast lunch dinner meal eat supper street view navig

- < 10 words in description after preprocessing? Eliminate!

Clustering Apps by Description

2. Identifying Topics with LDA (Latent Dirichlet Allocation)

- Topic - cluster of words that frequently occur together
  - recipe, cook, food, …
  - temperature, forecast, rain, …
- 30 topics, 1 app can belong to a max of 4 topics, 5% probability

Id	Assigned Name	Most Representative Words (stemmed)
0	"personalize"	galaxi, nexu, device, screen, effect, instal, customis
1	"game and cheat sheets"	game, video, page, cheat, link, tip, trick
2	"money"	slot, machine, money, poker, currenc, market, trade, stock, casino coin, finance
3	"tv"	tv, channel, countri, live, watch, germani, nation, bbc, newspap
4	"music"	music, song, radio, play, player, listen
5	"holidays" and religion	christmas, halloween, santa, year, holiday, islam, god
6	"navigation and travel"	map, inform, track, gps, navig, travel
7	"language"	language, word, english, learn, german, translat
8	"share"	email, ad, support, facebook, share, twitter, rate, suggest

Clustering Apps by Description

London Restaurants Example:

look restaur bar pub just fun london search applic inform need
can search everi type food want french british chines indian etc
can us car bicycl walk can view object map can search object
can view object near can view direct visual rout distanc durat
can us street view can us navig keyword london restaur bar pub
food breakfast lunch dinner meal eat supper street view navig

“navigation and travel” : map, inform, track, gps, navig, travel

“food and recipes” : recip, cake, chicken, cook, food

“travel” : citi, guid, map, travel, flag, countri, attract

Clustering Apps by Description

3. Clustering Apps with K-means

- Topic modeling for app : vector of affinity values for each topic

[Idea : similarity between different app descriptions!]

Input:

- set of elements in metric space
- K number of desired clusters

Output:

- centroid for each cluster
- association of each element in dataset with nearest centroid
- results in a cluster

Clustering Apps by Description

Input:

{app1, app2, app3, app4}

{topic1, topic2, topic3, topic4}

K = 2

Output:

Application	**topic1**	**topic2**	**topic3**	**topic4**
app1	0.60	0.40	-	-
app2	-	-	0.70	0.70
app3	0.50	0.30	-	0.20
app4	-	-	0.40	0.60

Clustering Apps by Description

4. Finding Best Number of Clusters

- Multiple trials
- Range of K values
- 2 to num(topics)x4

“Best” number of clusters?

Clustering Apps by Description

Elements Silhouette

- Measure of how closely the element is matched to other elements within its cluster, and how loosely it is matched to other elements of neighbouring cluster
- -> 1 : close to appropriate cluster
- -> -1 : wrong cluster

Clustering Apps by Description

RESULT:

- 32 clusters

Id	Assigned Name	Size	Most Important Topics
1	"sharing"	1,453	**share** (53%), settings and utils, navigation and travel
2	"puzzle and card games"	953	**puzzle and card games** (78%), share, game
3	"memory puzzles"	1,069	**puzzle and card games** (40%), game (12%), share
4	"music"	714	**music** (58%), share, settings and utils
5	"music videos"	773	**popular media** (44%), **holidays and religion** (20%), share
6	"religious wallpapers"	367	**holidays and religion** (56%), design and art, wallpapers
7	"language"	602	**language** (67%), share, settings and utils
8	"cheat sheets"	785	**game and cheat sheets** (76%), share, popular media
9	"utils"	1,300	**settings and utils** (62%), share, connection
10	"sports game"	1,306	**game** (63%), battle games, puzzle and card games
11	"battle games"	953	**battle games** (60%), **game** (11%), design and art
12	"navigation and travel"	1,273	**navigation and travel** (64%), share, travel

Identifying Outliers by APIs

1. Extracting API Usage
- static API usage <-> behavior
- Android bytecode : information flow analysis
- API usage : explicitly declared

How?

- *apktool*
- *smali* disassembler
- *number of call sites* for each API

Identifying Outliers by APIs

2. Sensitive APIs

- All APIs would result in overfitting
- Sensitive as per Android permission setting
- API is sensitive iff
  - declared in the binary
  - permission requested in manifest file

Identifying Outliers by APIs

android.net.ConnectivityManager.getActiveNetworkInfo()
android.webkit.WebView()
java.net.HttpURLConnection.connect()
android.app.NotificationManager.notify()
java.net.URL.openConnection()
android.telephony.TelephonyManager.getDeviceId()
org.apache.http.impl.client.DefaultHttpClient()
org.apache.http.impl.client.DefaultHttpClient.execute()
android.location.LocationManager.getBestProvider()
android.telephony.TelephonyManager.getLine1Number()
android.net.wifi.WifiManager.isWifiEnabled()
android.accounts.AccountManager.getAccountsByType()
android.net.wifi.WifiManager.getConnectionInfo()
android.location.LocationManager.getLastKnownLocation()
android.location.LocationManager.isProviderEnabled()
android.location.LocationManager.requestLocationUpdates()
android.net.NetworkInfo.isConnectedOrConnecting()
android.net.ConnectivityManager.getAllNetworkInfo()

Identifying Outliers by APIs

3. One-Class Support Vector Machine

- Learn features of one class of elements
- Detect anomaly/novelty within this class

In this case,

Features : sensitive APIs

Training set: subset of applications in a cluster

Result: cluster specific models that can identify outliers

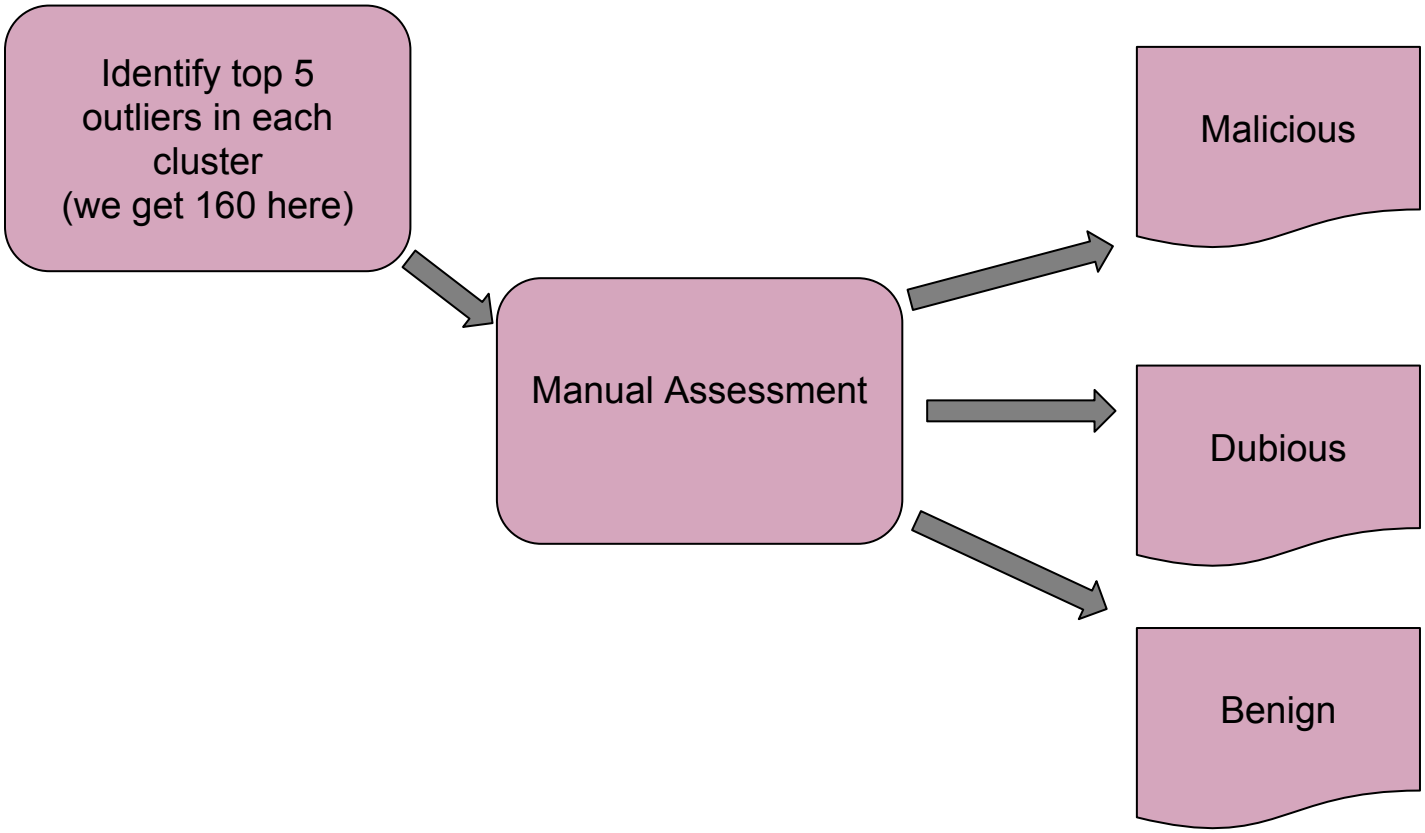
How? Actual distance of element from hyperplane built by OC-SVM

Evaluation

RQ1: Can our technique effectively identify anomalies (i.e mismatches between description and behaviour) in Android applications?

RQ2: Can our technique be used to identify malicious Android applications?

RQ1: Effectiveness



Results

Behavior	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	Total
Malicious	1	2	1	0	0	0	0	2	0	0	0	0	0	3	0	4	2	0	0	1	3	5	1	3	1	1	2	3	1	4	1	1	**42** (26%)
Dubious	1	2	1	0	0	0	0	0	0	2	2	1	1	0	0	0	1	2	1	1	0	0	2	0	1	0	1	0	1	0	0	0	**20** (13%)
Benign	3	1	3	5	5	5	5	3	5	3	3	4	4	2	5	1	2	3	4	3	2	0	2	2	3	4	2	2	3	1	4	4	**98** (61%)

Top outliers, as produced by CHABADA, contain 26% malware; additional 13% apps show dubious behavior.

RQ2: Malware detection

- Uses a known dataset of malicious apps for Android (1200, but filtering English only leaves us with 172)
- OC SVM used as a classifier. Trained on 90% of 'benign'-only set (i.e excluding the ones identified as malicious) and then used on set composed of known malicious apps and 10% of benign apps.
- Repeated 10 times on clusters that had different number of malicious apps

What we are trying to achieve - simulate a situation where malware attack is novel and CHABADA must correctly identify malware without knowing previous malware patterns.

Results

	Predicted as malicious	Predicted as benign
Malicious apps	96.5 (56%)	75.5 (44%)
Benign apps	353.9 (16%)	1,884.4 (84%)

In our sample, even without knowing existing malware patterns, CHABADA detects the majority of malware as such.

Classifying without clustering yields more false negatives.

Clustering by description topics is superior to clustering by given categories.

Limitations & threats to validity

- External validity
- Free apps only
- App and malware bias
- Researcher bias
- Native code and obfuscation
- Static Analysis
- Static API declarations
- Sensitive APIs

Conclusion

- CHABADA approach effectively identifies applications whose behavior would be unexpected given their description
- Identified examples of misleading advertizing
- Formulated a novel effective detector for yet unknown malware

Consequences

- Vendors must be much more explicit about what their apps do to earn their income.
- App store Application suppliers such as Google should introduce better standards to avoid deceiving or incomplete advertising

Discussion Question 1

Given what you've seen in this presentation, how many of you are going to look a bit further into the applications you download?

a. Descriptions are important but might not always describe the implemented behavior.

Discussion Question 2

CHABADA only identified 56% of malicious apps as malware, is it still worth using?

Discussion Question 3

The authors only tested CHABADA using apps from the Google Play Store, would this approach extend to Apple and Windows apps

Discussion Question 4

There is a manual distinction being made between dubious and malicious. Is this reliable enough?

Discussion Question 5

For identifying API outliers, the OC-SVM model is used. Is there a case when this model would not work?

References

Gorla, A., Tavecchia, I., Gross, F., & Zeller, A. (2014, May). Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 1025-1035). ACM.