Analysis of Android Spam Call Blocking Applications: Developing a Methodology For Dynamic Analysis

Atanas Pashov
EEMCS, Delft University of Technology, The Netherlands

1. Introduction

- Many Android applications that detect and/or prevent spam calls exist, as malicious actors have developed ways to abuse the telephony industry
- Little information as to how they work
- Already existing research on:
  - building a database of caller IDs [1]
  - how Android OS is protecting against spam [2]
  - utilize different algorithms to detect spam [3],[4],[5]
- Little to no research on how existing production applications work, or how to approach analyzing them

2. Objectives

1. Develop a methodology for dynamically analysing an Android application in order to determine:
   - What Android API calls are performed?
   - How the application ensures that the spam calls are blocked? (blocklisting/allowlisting phone numbers, keeping track of a "score" that is updated based on reputation, etc.)
   - What data does it need in order to block spam calls?
   - Where is this data stored and how often is it updated?
   - Can we get a list of blocked caller IDs from the application?
2. Develop a script that extracts the differences between subsequent runs of a given application

3. Methodology

- ACVTool
- Generate several reports depending on different outputs
- Extract and analyse the differences between them
- Also analyse the common libraries that could contain potentially useful information
- Develop some techniques to extract more information from the common libraries (presented in the results)
- Mostly focused on Hiya to develop the methodology

4. Results

com.webascender.callerid

1. Initial Analysis

- Start by looking into the non-obfuscated libraries based on their names

`com.hiya.client.database.db`

- Contains references to a Room database

`HiyaRoomDb_Impl$a.smali` file

- Create table SQL statements

caller_ids	
_id	**INTEGER**
entity_type	TEXT
phone_number	TEXT
display_name	TEXT
display_location	TEXT
display_image_url	TEXT
attribution_image	TEXT
attribution_url	TEXT
attribution_name	TEXT
profile_tag	TEXT
display_line_type	TEXT
entity_expired_time_millis	INTEGER
source_type	TEXT
last_access_time_millis	INTEGER
profile_icon_type	TEXT
reputation_category_id	INTEGER
category_name	TEXT
display_category_name	TEXT
line_type_id	TEXT
display_detail	TEXT
display_description	TEXT
language_tag	TEXT
display_background_url	TEXT
display_background_assettype	TEXT

local_override_ids	
_id	**INTEGER**
phone_number	TEXT
reported_name	TEXT
user_comment	TEXT
category_name	TEXT
reputation_category_id	INTEGER
profile_tag	TEXT
time_created	INTEGER

block_numbers	
_id	**INTEGER**
phone_number	TEXT
normalized_number	TEXT
created_time_millis	INTEGER
is_partial	INTEGER
country_calling_code	INTEGER

`com.hiya.stingray`

- a lot of methods that deal with displaying the information
- "Showing post call notification: reputation=%s identity=%s notification=%s"
- `TelephonyManager` and `BroadcastReceiver` were used to intercept the calls

2. Analysing the Differences

- Ran the application 4 times:

Number	How it is handled
(650) 555-1212	produces a name and location of caller
605-367-1378	produces a warning
0611945863112	produces "suspected spam"
201-200-0014	falgged but also identified caller ID

- Extract the different libraries using the developed tool

Obfuscated classes containing `this.isFraudOrSpam`

`this.toCallerId` that returns a `RoomCallerId` (same as the DB table)

- Some differences in the internal libraries
  - for the "suspected spam" case: NumberParseException class from the i18n library is executed and a custom PhoneParserFailure class returns an exception.
    => maybe unexpected format, but still is flagged as suspected spam
    => there is some offline on-device phone number processing able to flag some caller IDs.

`com.hiya.stingray`

- Contains an enum that indicates an internal database (possibly user defined ADD_BLACKLIST, REMOVE_BLACKLIST) together with some caller ID analysis (BLOCKED_STARTS_WITH to categorize the phone numbers.
  - `PhoneSendEvent` saved to a Realm DB
  - obfuscated library in a class from the stingray manager package pattern matches country codes (maybe categorize caller IDs based on country)

3. Analyzing Code Referencing Obfuscated Classes

- Helps to map class names to obfuscated references (method calling isFraudOrSpam(), classes connected to flaggin numbers, enums and caller ID DAO classes)
- `ContactManager` class
- `getReputationLevel() =>` reputation is used to determine OK, UNCERTAIN, SPAM, FRAUD from a `ReputationLevel` enum
- SELECT * FROM `caller_ids` to Room DB method; the Room class uses a Map object
  - The method that adds to that map is called in 11 classes; all seem to be internal DB calls

4. Analyzing Code Referencing Certain Android Libraries

- `HttpURLConnection`
  - References to Google Protocol Buffers
- `m/z$a;->request()`
  - `HiyaExcessiveAuthRequestsException`
  - Probably Hiya HTTP authenticated requests

5. Analysing Code Referencing Certain Strings

- Searching for "JSON"
- Requests to `hash/hashCountries`, `auth/token`, `phone_numbers/feedback`, `phone_numbers/eventProfile`, `phone_numbers/events`: `eventProfileEvent` stores phone call info, although it was not executed in any report
- Cryptographic public keys
- Call logs from the DB are used

5. Conclusions

- Dynamic analysis should only be performed for applications that are hard to analyze statically or using different methods.
  - Or in combination with other different methods
- Obfuscation really slows an engineer down
- When little information could be extracted from other analysis approaches, or when simply more information is needed for a specific application
- Further research:
  - Would be useful to search for class/method names that were executed
  - Renaming package/class/methods in ACVTool

6. REFERENCES

[1] Sharbani Pandit, Roberto Perdisci, Mustaque Ahamad, and Payas Gupta. *Towards measuring the effectiveness of telephony blacklists*. 01 2018.

[2] A. Shabtai, Y. Fledel, U. Kanonov, Yuval Elovici, and Shlomi Dolev. Google android: A state-ofthe-art review of security mechanisms. Neural Networks, abs/0912.5, 12 2

[3] Arka Bhowmik and Debashis De. mtrust: Call behavioral trust predictive analytics using unsupervised learning in mobile cloud computing. Wireless Personal Communications, 117:1–19, 03 2021

[4] Chinmay R C, Mrinal Raj, Sarthak Mishra, and Shobha K. Record.ai - an ai based solution to classify calls based on conversation. In 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), pages 1096–1101, 2021

[5] Chang Sung, Chi Kim, and Joo Park. Development of humming call system for blocking spam on a smartphone. Multimedia Tools and Applications, 76, 08 2017.

7. Acknowledgements

Atanas Pashov (A.I.Pashov-1@student.tudelft.nl)
Supervisors: Apostolis Zarras (A.Zarras@tudelft.nl), Yury Zhauniarovich (Y.Zhauniarovich@tudelft.nl)