Automated Semi-supervised Authorship Attribution of Android Binaries

Hugo Gonzalez, Natalia Stakhanova, Ali A. Ghorbani

Faculty of Computer Science, University of New Brunswick

ISCX Information Security Centre of Excellence

UNB EST. 1785 UNIVERSITY OF NEW BRUNSWICK — Computer Science

The Problem

- Malware on the Android platform has increased exponentially. Growing more than 600% in the last two years.
- New threats appear each day in the form of botnet, ransomware, adware or even worse: all together
- Typical methods to perform malware analysis focus on binaries it self, creating signatures or behavioral profiles.

Proposed solution

- It is our believe that only a reduced set of skilled authors produce primary strands of the malware, meanwhile the available samples are repackaged, reused, recompiled or evolved versions from the main strand.
- Our proposed approach looks for similarities in Android binary code from authors perspective.
- It involves 3 step process:
  - Creation and evaluation of benchmark profiles
  - Creation and evaluation of incremental profiles
  - Emergent behavior layer, where a large number of apps are analyzed.

Contributions

- Android Authors Dataset
- Method to classify and cluster on top of Random Forest algorithm
- Approach to create label profiles
- Large scale evaluation

Future work

- Evaluate complementary features
- Improve random forest performance using distributed task
- Use small number of files to create possible profiles.
- Web service to analyze apps.

STEP 1

Benchmark profiles

Authors Dataset

33 authors.
1,428 apk files,
from 8 markets

Feature Extraction

Table 2: Values extracted from apps

Value
total number or classes, total number of classes containing interfaces, total number of classes containing information about source file, total number of classes containing annotations, total number of classes containing data.
total number of direct methods, total number of virtual methods, total number of abstract methods, total number of methods containing try and catch, total number of methods containing debug info.
total number of static fields, total number of instanced fields.
total size of instructions expressed in bytes.

+

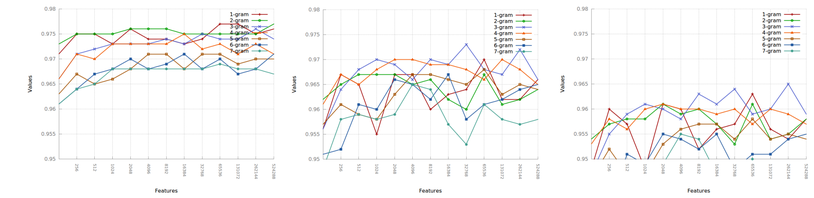
Table 1: Array related bytecode mnemonics

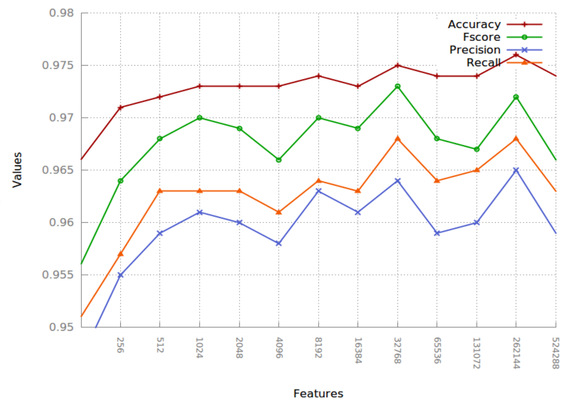
mnemonic bytecode	description
ARRAY LENGTH	Store in the given destination register the length of the indicated array.
NEW ARRAY	Construct a new array of the indicated type and size.
FILLED NEW ARRAY	Construct an array of the given type and size, filling it with the supplied contents.
FILLED NEW ARRAY RANGE	Similar as previous.
FILL ARRAY DATA	Fill the given array with the indicated data
AGET, AGET BOOLEAN, AGET BYTE, AGET CHAR, AGET SHORT AGET WIDE, AGET OBJECT	Perform array operation at the identified index of the given array, loading into the value register.
APUT, APUT BOOLEAN, APUT BYTE, APUT CHAR, APUT SHORT, APUT WIDE, APUT OBJECT	Perform array operation at the identified index of the given array, storing into the value register.

Supervised learning

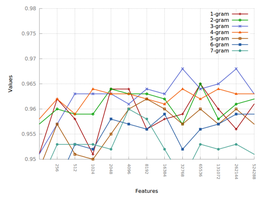
Random Forest

** Array is a data structure used in Android programming that is used in almost all apps. Main objective of array structures is to efficiently store similar data. Different from other structures, compilation process produce minimal effects in resulting binary code.





3-gram results



We choose 3gram with 32 768 features based on time and performance.
** We perform a 10 fold cross validation

STEP 2

Incremental profiles

Authors Dataset

33 authors.
1,428 apk files,
from eight markets

Feature Extraction

Table 2: Values extracted from apps

Value
total number or classes, total number of classes containing interfaces, total number of classes containing information about source file, total number of classes containing annotations, total number of classes containing data.
total number of direct methods, total number of virtual methods, total number of abstract methods, total number of methods containing try and catch, total number of methods containing debug info.
total number of static fields, total number of instanced fields.
total size of instructions expressed in bytes.

+

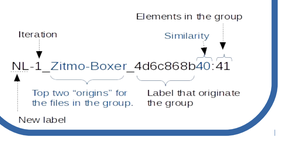
Table 1: Array related bytecode mnemonics

mnemonic bytecode	description
ARRAY LENGTH	Store in the given destination register the length of the indicated array.
NEW ARRAY	Construct a new array of the indicated type and size.
FILLED NEW ARRAY	Construct an array of the given type and size, filling it with the supplied contents.
FILLED NEW ARRAY RANGE	Similar as previous.
FILL ARRAY DATA	Fill the given array with the indicated data
AGET, AGET BOOLEAN, AGET BYTE, AGET CHAR, AGET SHORT AGET WIDE, AGET OBJECT	Perform array operation at the identified index of the given array, loading into the value register.
APUT, APUT BOOLEAN, APUT BYTE, APUT CHAR, APUT SHORT, APUT WIDE, APUT OBJECT	Perform array operation at the identified index of the given array, storing into the value register.

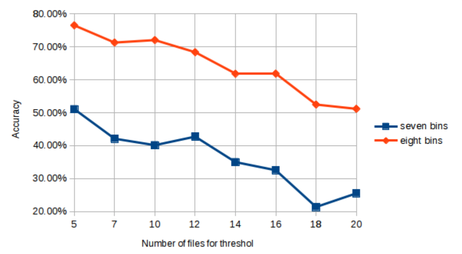
Supervised learning

Random Forest
3gram, 32 768 features

Leave out 20% of authors
Create new labels for unseen authors




```
Minibatch_Classify_and_cluster :

input -> list_of_apps.
output -> results_map-list, nofinding_map-list, groups_map-list

binvalues = 80, 70, 60, 50, 40, 30, 20 ,10
for binval in binvalues:
   clear map-list[binval]

clear results_map-list
clear nofinding_map-list

for app in list_of_app:
   predictions = model.predict_probability(app)
   value, class = get_max(predictions)
   if value > .90 then:
      results_map-list [ class]  add (app)
   else :
            notfind = True
            for binval in binvalues:
                  if value > binval / 100 :
                        notfind = False
                        map-list[binval][class] add (app)
                        break
   if notfind == True:
      nofinding_map-list add (app)

clear groups_map-list

for each binval in binvalues:
   for each class in map-list[binval]:
      if map-list[binval][class] contains more than 20 elements:
         label = CreateNewLabel (map-list[binval][class])
         groups_map-list [label] = map-list[binval][class]
      else :
         for each app in map-list [class]:
            nofinding_map-list add (app)

return results_map-list, nofinding_map-list, groups_map-list
```



78% of apps previously unseen where correctly attributed to a new label profile created during the process. To reduce the model size and time, we choose 8 bins and 10 files instead of 5 files.
*We perform a 10 fold cross validation for these experiments.

STEP 3

Emergent behavior layer

Datasets

Table 3: Datasets used

Name	Apps	Description
Authors dataset	1,428	Collected from eight different markets, this dataset contains information about 33 known authors.
Fdroid dataset	1,395	Open source apps without advertisement libraries.
Drebin partial dataset	3,181	From the original dataset, we keep only families with more then 20 apps, to explore relations between families and authors.
Adware dataset	211	Three families related between them. They trojanize legit apps.
Ransomware dataset	136	Ransomware software usually contain specialized behavior. It includes four families.
GooglePlay-2015 dataset	4,574	Apps collected from Google Play in middle 2015 from top popular and top new.
Markets dataset	23,656	Apps were collected from eight third part markets, removing suspicious apps.

Feature Extraction

Values extracted from apps (classes, metods …)

+

Array related opcodes

Semi -Supervised learning

Random Forest

3gram, 32 768 features, 10 files per groups and 8 bins.

Total Unseen apps	33,153		
Attributed apps	4,830	14.57%	
Correctly attributed apps *	3,371	**69.79%**	10.17%
Unattributed apps	28,323	85.43%	
Related without label	11,839	35.71%	
Non related apps	16,484	49.72%	
New created label profile	1,147		
With more than 200 apps	3	Plankton	
With more than 100 but less than 200	12	Googleplay	

* Apps are considered correctly attributed when they belong to the same malware family or from the same market..
Further investigation is needed.