Attack Android Malware Detection With NLP

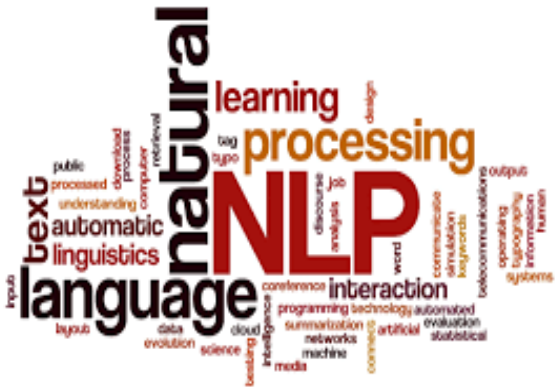
NIV KOTEK – 20823631

ORIA ZADOK - 315500157

Our Github repo for this project can be found here.
The original source repo for this project is located here.

Moderator: HAREL BERGER

What is Natural Language Processing (NLP)?



- Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

- NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.

Introduction

- The name of the paper : Malware Detection With NLP.
- The feature: API call
- The type of classifier: static

- **Stages of building the classifier**:
  - 1. Unpacks the APKs present.
  - Tags the unpacked APK based on the directory it was found in to benign and malicious
  - 2. Collect dex files from each APK and store them with malicious or benign tag.
    Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the Java programming language.
  - 3. Create xml of the dex files using Dexdump utility
  - 4. Create csv file of label and text of dex dumps
  - 5. Train document vectors on xml of classes.dex files

Dex file format:

1. File Header
2. String Table
3. Class List
4. Field Table
5. Method Table
6. Class Definition Table
7. Field List
8. Method List
9. Code Header
10. Local Variable List

	label	file_content
0	1	<api> <package name="android.annotation" > <class name="SuppressLint" extends="java.lang.Object" interface="true" abstract="true" static="false" final="false" visibility="public" > <implements name="jav
1	-1	<api> <package name="com.adwo.adsdk" > <class name="A" extends="java.lang.Object" interface="false" abstract="false" static="false" final="true" visibility="package" > <implements name="android.view.
2	1	<api> <package name="com.android.security" > <class name="DataStorage$OpenHelper" extends="android.database.sqlite.SQLiteOpenHelper" interface="false" abstract="false" static="false" final="false" vis
3	1	<api> <package name="ru.droid.install.other" > <class name="ControlReceiver" extends="android.content.BroadcastReceiver" interface="false" abstract="false" static="false" final="false" visibility="public" > <
4	-1	<api> <package name="com.changcheng.download.service" > <class name="DBOpenHelper" extends="android.database.sqlite.SQLiteOpenHelper" interface="false" abstract="false" static="false" final="false"

The Attack

The name of the attack:
In-app feature attack.
Type of weak spots:
The number of features.
The way to find the weakness:

- A. First we opened the classifier and checked which files it works on.
- B. We printed the names of the applications that the classifier took as "test" and those that he took as "training". (From this section we work on the applications that were taken for testing)
- C. We found that the features are xml files that describe the API of the applications.
- D. We looked for the weak spots of the features and changed the values of the features in the xml files.
- E. After section D was unsuccessful, we took several features from an application that was classified as "benign" and put them into an application that was classified as "malicious".
- F. After a considerable number of activations the classifier after each change we were able to narrow the search range. of
- G. We found that a number of features (which we took from the application which was classified as "benign") that we add (More than 5000 lines)to an application that is classified as "malicious" lowers the accuracy of the classifier.
- H. After that, we added empty xml tags in the amount of more than 5000 lines of code to the beginning of the features of the application which was classified as " malicious " and we found that even now it lowers the accuracy of the classifier.
- I. Next, we added more than 5000 lines of code empty xml tags to the end of the app's features that were classified as "bad" and found that the classifier found them to be bad.
- J. After that, we added in a comment block the empty xml tags in the amount of more than 5000 lines of code to the beginning of the features of the application classified as " malicious " and we found that even now it lowers the accuracy of the classifier.
- K. After that, we added the same empty xml tags (which have no meaning) to other apps that were classified as " malicious ", but this time the classifier found them to be malicious.
- L. After that, we added various empty (meaningless) xml tags to other apps that were classified as " malicious", and found that the accuracy of the classifier decreased.
- In conclusion: we discovered that the weak point of the classifier is the number of features. That is, if we add to the beginning of the features of the application an amount of more than 5000 lines of code of empty characters (which have no meaning) or any features from an application that was classified as " benign", we will lower the accuracy of the classifier.

The Attack

```
<api>
<package name="ru.droid.install.other"
>
<class name="ControlReceiver"
 extends="android.content.BroadcastReceiver"
 interface="false"
 abstract="false"
 static="false"
 final="false"
 visibility="public"
>
```

Examples of features that can be added to the application:

- A. <"stam name="android.nlp"">
- B. <"package name="android.support.v4.app"">

How you intend to exploit it:

- A. We will add a code that checks which bad apps the classifier takes for "testing".
- B. We will add a code that opens the " malicious " application with Apktool.
- C. We will add to the smali group a file with a name starting with the letter "A" (so that the features we add will be first in the xml).
- D. We will add to the file that starts with the letter "A" any features from the application that was classified as " benign " (at least 5000 lines of code)
- D. We will add a code that closes the " malicious " app after the changes with apktool
- E. We will run the classifier on the application after the changes.

important comments:

- A. We did a test with 18 benign apps and 18 malicious apps, of which about 4 benign and 4 malicious apps were taken for testing. We ran the classifier and the accuracy came out 1.0. After that we opened the 4 malicious applications of the test with apktool and added several features from a benign application randomly. After that, we closed the applications and ran the classifier and got 0.5 accuracy(The addition was done in the same way for all 4 apps)
- B. Any tags can be added as features or any features from a benign application
- C. Must have at least more than 5000 lines of code of adding features
- D . The addition of the features must be at the beginning of the xml.
- E. The testing and addition of features was done by apps randomly (that is, the weak point of the classifier is the number of features)

Code

Shell - Apktool

```
#! /usr/bin/bash
start_dir=$PWD
echo $start_dir
#the given app
echo ss
#ls -d TM*
apps=`ls -d TM*`
echo $apps
echo ee
insert=$1
echo $insert
for f in $apps
do
# open the given app
apktool d $f
# get the folder to go into
folder="${f:0:-4}"
cp -r $insert $folder/smali/
#go back to the start folder
cd $start_dir
# pack the folder to an apk
apktool b $folder
# go to the place apktool put the new app85.apk
cd $folder/dist
# move the app to the current folder
mv $f "$start_dir/new_$f"
# remove the folder apktoll created
cd $start_dir
rm -r $folder
done
mv new* ../testData/malicious
```

Split Train & Test

```
import os
from androidMalwareDetectionWithNLP.utilities import constants
import math
import subprocess

ls_mal =[file for root, directory, files in os.walk(constants.RAW_FILES_PATH)
for file in files if file.endswith('.apk') and 'malicious' in root]
len_ls_mal = len(ls_mal)
len_ls_mal_test=math.ceil(len_ls_mal*0.2)

ls_benign=[file for root, directory, files in os.walk(constants.RAW_FILES_PATH)
for file in files if file.endswith('.apk') and 'benign' in root]
len_ls_benign = len(ls_benign)
len_ls_benign_test =math.ceil(len_ls_benign*0.2)
def execute_unzip(len_ls_mal_test,len_ls_benign_test):
    for root, directory, files in os.walk(constants.RAW_FILES_PATH):
        for file in files:
            if file.endswith('.apk') and 'benign' in root:
                if len_ls_benign_test>0:
                    len_ls_benign_test = len_ls_benign_test - 1
                    if "TB" in file:
                        continue
                    else:
                        subprocess.run('mv {} {}'.format(root + "/" + file,
                                                         root + "/" + "TB" + file),
                                       shell=True,
                                       capture_output=True,
                                       text=True,
                                       check=True).stdout.split('\n')

            elif file.endswith('.apk') and 'malicious' in root:
                if len_ls_mal_test > 0:
                    len_ls_mal_test=len_ls_mal_test-1
                    if "TM" in file:
                        continue
                    else:
                        subprocess.run('mv {} {}'.format(root + "/" + file,
                                                         root + "/" + "TM" + file),
                                       shell=True,
                                       capture_output=True,
                                       text=True,
                                       check=True).stdout.split('\n')

            else:
                print('Unknown packaging. Cannot unpack the archive.')

execute_unzip(len_ls_mal_test,len_ls_benign_test)
```

Data frame with train and test

```
def generate_csv(path, tag):
    labels = []
    file_content = []
    names =[]
    for files in os.listdir(path):
        if "malicious_" in files:
            labels.append("-1")
            if "TM" in files :
                names.append("TM")
            else:
                names.append("ALL")

        else:
            if "TB" in files:
                names.append("TB")
            else:
                names.append("ALL")
            labels.append("1")
        filename = os.path.join(path, files)
        print("[INFO] Reading file :  " + filename)
        with open(filename, encoding='ascii', errors='ignore') as f:
            content = f.read().splitlines()
            content = " ".join(content)
            file_content.append(content)
            print("[INFO] File text added to list...")
    print("[INFO] Saving the texts in dataframe...")
    df = pd.DataFrame(
        {    'name': names,
            'label': labels,
            'file_content': file_content
```

Insert into a list

```
X_train=[]
X_test=[]
y_train=[]
y_test=[]

for i ,v in enumerate(data.name):
    if "TM" in v or "TB" in v:
        X_test.append(data.file_content[i])
        y_test.append(data.label[i])
    else:
        X_train.append(data.file_content[
        y_train.append(data.label[i])
```

Dataset

```
<api>
<package name="ru.droid.install.other"
>
<class name="ControlReceiver"
 extends="android.content.BroadcastReceiver"
 interface="false"
 abstract="false"
 static="false"
 final="false"
 visibility="public"
>
<field name="a"
 type="java.lang.String"
 transient="false"
 volatile="false"
 static="true"
 final="false"
 visibility="public"
>
</field>
<constructor name="ControlReceiver"
 type="ru.droid.install.other.ControlReceiver"
 static="true"
 final="false"
 visibility="package"
>
</constructor>
<constructor name="ControlReceiver"
 type="ru.droid.install.other.ControlReceiver"
 static="false"
```

- **Train** on 11,000 App
  - Benign - 10,000 App
  - Malicious - 1,000 App
- **Test** on 20% App
  - Benign - 2,000 App
  - Malicious - 2,00 App

API

5	ALL	1	<api> <package name="com.troii.weblauncher" > <class name="R$attr" extends="java.lang.Object" interface="fal
6	TB	1	<api> <package name="com.jtj.bgbg" > <class name="AboutActivity" extends="android.app.Activity" interface="f
7	TM	-1	<api> <package name="0Android.support.v4.accessibilityservice" > <class name="AccessibilityServiceInfoCompat$A
8	ALL	1	<api> <package name="it.mp.codicileggi.libri.casellariogiudiziale" > <class name="CreaDBAsyncTask" extends="a
9	TB	1	<api> <package name="com.worldmanager.beast" > <class name="BuildConfig" extends="java.lang.Object" interfac
10	ALL	1	<api> <package name="com.toronto_channels.iptv" > <class name="BuildConfig" extends="java.lang.Object" inter
11	ALL	1	<api> <package name="android.annotation" > <class name="SuppressLint" extends="java.lang.Object" interface="

Results

- **Dataset** on 11,000 App
  - Benign - 10,000 App
  - Malicious - 1000 App
- **Train** on 8,800 App
  - Benign - 8,000 App
  - Malicious - 800 App
- **Test** on 20% App
  - Benign - 2,000 App
  - Malicious - 2,00 App
- **Epochs** on 30

```
100%|██████████| 10995/10995 [00:00<00:00, 1411020.18it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1310645.50it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1412186.81it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1506086.63it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1409381.51it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1515092.07it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1404873.35it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1405215.81it/s]
100%|██████████| 10995/10995 [00:00<00:00, 1493164.08it/s]
```

Before

Accuracy 0.9731696225557072

	precision	recall	f1-score	support
-1	0.85	0.85	0.85	200
1	0.98	0.99	0.99	1999
accuracy			0.97	2199
macro avg	0.92	0.92	0.92	2199
weighted avg	0.97	0.97	0.97	2199

After

Accuracy 0.8958617553433379

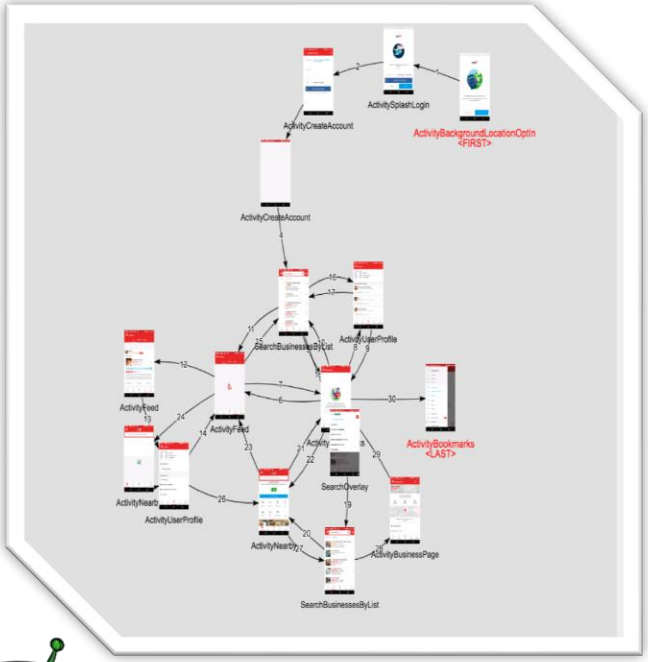
	precision	recall	f1-score	support
-1	0.00	0.00	0.00	200
1	0.91	0.99	0.95	1999
accuracy			0.90	2199
macro avg	0.45	0.49	0.47	2199
weighted avg	0.83	0.90	0.86	2199

Droidbot

Results

TMapp1472,0,0
TMapp575,0,0
TMapp851,0,0
TMapp800,0,0
TMapp216,0,0
TMapp217,0,0
TMapp1058,0,0
TMapp821,0,0
TMapp853,0,0
TMapp379,0,0
TMapp708,0,0
TMapp154,0,0
TMapp247,0,0
TMapp103,0,0
TMapp527,0,0

- DroidBot is a lightweight test input generator for Android. It can send random or scripted input events to an Android app, achieve higher test coverage more quickly, and generate a UI transition graph (UTG) after testing.
- We will use Droidbot to , to verify that the attack did not damage the functionality of the application.
- The test was done on 100 apps before and after the change
- can be seen from the results the attack did not damage the functionality of the application

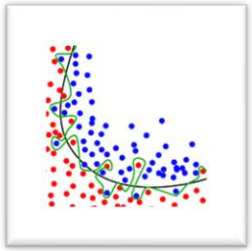


Summary

sklearn

NLP

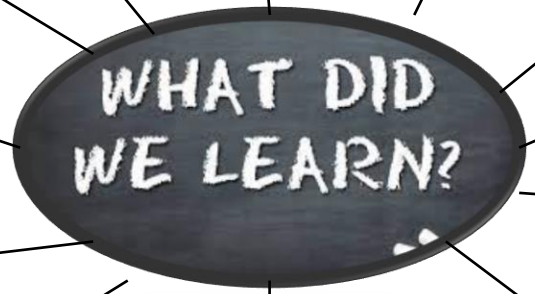
Dexdump



over fitting

Dataset

GENSIM



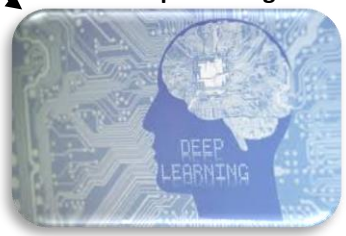
APKTOOL

Git and GitHub

Droidbot

BASH

Machine Learning



Deep Learning


QUESTIONS


ANY
QUESTIONS