National Institute of Technology, Tiruchirappalli

Department of Computer Applications

Information Security
Project Report

Submitted to:

Dr. Janet Barnabas

Submitted by:

Akriti Upadhyay

205120007

MCA – 2nd Year

RiskAnalyzer: Machine Learning-based Risk Analysis on Android

Problem Definition:

How can one verify the reliability of an Android Application, ensuring that it is not a malware that will harm their Mobile device?

Introduction:

Android-enabled smartphones remain a sensitive target for malware that aim at exploiting its diffusion to reach a high number of potential victims. Since users have access to a high number of apps through public markets and external web sites, they need reliable tools to rate the trustworthiness of apps they are going to install. App rating is empirically calculated according to different risk analysis techniques. Currently, most of them calculate a risk index value (hereafter, RIV) through probabilistic methods applied to the set of permissions required by the app. I argue that such approaches suffer from intrinsic limitations in terms of both methodology and setup. To prove this, I apply some optimizations to existing techniques at the state of the art, and I evaluate them through an extensive empirical assessment on a dataset made by 112.425 apps and 6.707 malware samples. Then, I propose a novel approach based on machine learning techniques that I implemented in an open-source tool, i.e., RiskAnalyzer (Risk Index for Android). Finally, I evaluate the performance of RiskAnalyzer on the same dataset, thereby proving that the proposed methodology outperforms probabilistic approaches.

Proposed Solution:

Risk analysis on Android is aimed at providing metrics to users for evaluating the trustworthiness of the apps they are going to install. Most of current proposals calculate a risk value according to the permissions required by the app through probabilistic functions that often provide unreliable risk values. To overcome such limitations, I have proposed RiskAnalyzer, a tool for risk analysis of Android apps based on machine learning techniques. RiskAnalyzer outperforms probabilistic methods in terms of precision and reliability.

RiskAnalyzer (Risk Index for Android) is a tool for quantitative risk analysis of Android applications written in Java (used to check the permissions of the apps) and Python (used to compute a risk value based on apps' permissions). The tool uses classification techniques through scikit-learn, a machine learning library for Python, in order to generate a numeric risk value between 0 and 100 for a given app. In particular, the following classifiers of scikit-learn are used in RiskAnalyzer:

- Support Vector Machines (SVM)
- Multinomial Naive Bayes (MNB)
- Gradient Boosting (GB)
- Logistic Regression (LR)

Unlike other tools, RiskAnalyzer does not take into consideration only the permissions declared into the app manifest, but carries out reverse engineering on the apps to retrieve the bytecode and then infers (through static analysis) which permissions are actually used and which not, extracting in this way 4 sets of permissions for every analyzed app:

- Declared permissions - extracted from the app manifest
- Exploited permissions - declared and actually used in the bytecode
- Ghost permissions - not declared but with usages in the bytecode
- Useless permissions - declared but never used in the bytecode

From the above sets of permissions (and considering only the official list of Android permissions), feature vectors (made by 0s and 1s) are built and given to the classifiers, which then compute a risk value.

I argue that the intrinsic limitations of probabilistic methods applied to Android Permissions (hereafter, APs) can be overcome by machine learning techniques able to build up more reliable RIVs.

Machine learning techniques are used for classifying elements, i.e., given a set of classes, they evaluate each element and assign a class to it. Therefore, they are particularly suitable for binary classification of malware. However, some techniques also provide a probability value related to the prediction. I leverage machine learning techniques to classify apps into two classes, i.e., malware and non-malware, and I use the classification probability to build up a RIV. For my purpose, I adopt the scikit-learn library, that implements a set of machine learning techniques and provides a probability function for some of them.

Machine learning techniques require feature vectors to compare and classify elements. In my context, elements are apps, and features are APs. I define feature vectors as follows: given AP Set the set of APs, for each app A, I define four feature vectors FV_S^A , with $S \in \{DAP_A , EAP_A , GAP_A , UAP_A\}$. Each FV is a binary vector of cardinality |APSet|, where $FV_S^A[i] = 1$ if $p_i \in S$, and $FV_S^A [i] = 0$ otherwise. I adopt a supervised learning approach. Supervised learning requires classifiers to be trained on a training set before being applied to classify new elements. I train a set of supervised classifiers on a subset of the dataset and then I use them to classify the remaining APKs.

Code:

- app.py

```
#!/usr/bin/env python3

import hashlweb

import os

import subprocess

import time



from flask import Flask

from flask import jsonify

from flask import make_response

from flask import render_template

from flask import request

from sqlalchemy import cast

from sqlalchemy.sql import text

from Irkzeug.exceptions import BadRequest, UnprocessableEntity

from Irkzeug.utils import secure_filename



from RiskAnalyzer import RiskAnalyzer

from model import db, Apk



ALLOID_EXTENSIONS = {"apk", "zip"}



def create_app():



   app = Flask(__name__)
```

```
app.config["UPLOAD_DIR"] = os.path.join(
    os.path.dirname(os.path.realpath(__file__)), "upload"
)
app.config["MAX_CONTENT_LENGTH"] = 100 * 1024 * 1024


app.config["DB_DIRECTORY"] = os.path.join(
    os.path.dirname(os.path.realpath(__file__)), "database"
)
app.config["DB_7Z_PATH"] = os.path.join(
    os.path.dirname(os.path.realpath(__file__)), "database", "permission_db.7z"
)
app.config["DB_PATH"] = os.path.join(
    os.path.dirname(os.path.realpath(__file__)), "database", "permission_db.db"
)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + app.config["DB_PATH"]
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False


# Establish the database connection.
db.init_app(app)


# Create the upload directory (if not already existing).
if not os.path.exists(app.config["UPLOAD_DIR"]):
    os.makedirs(app.config["UPLOAD_DIR"])


# Check if the database file is already extracted from the archive,
```

```
  # otherwise extract it.
  if not os.path.isfile(app.config["DB_PATH"]):
      instruction = '7z x "{0}" -o"{1}"'.format(
          app.config["DB_7Z_PATH"], app.config["DB_DIRECTORY"]
      )
      subprocess.run(instruction, shell=True)
  return app


application = create_app()


def check_if_valid_file_name(file_name):
  return (
      "." in file_name and file_name.rsplit(".", 1)[1].loIr() in ALLOID_EXTENSIONS
  )


@application.after_request
def add_cache_header(response):
  response.headers[
      "Cache-Control"
  ] = "public, max-age=0, no-cache, no-store, must-revalidate"
  response.headers["Pragma"] = "no-cache"
  response.headers["Expires"] = "0"
  return response


@application.errorhandler(400)
```

```python
@application.errorhandler(422)

@application.errorhandler(500)

def application_error(error):

  return make_response(jsonify(str(error)), error.code)



@application.route("/", methods=["GET"], strict_slashes=False)

def home():

  return render_template("index.html")



@application.route("/results", methods=["GET"], strict_slashes=False)

def results():

  return render_template("results.html")



@application.route("/upload", methods=["POST"], strict_slashes=False)

def upload_apk():

  # The POST request must contain a valid file.

  if "file" not in request.files:

    raise BadRequest("No file uploaded")

  file = request.files["file"]

  if not file.filename.strip():

    raise BadRequest("No file uploaded")



  if file and check_if_valid_file_name(file.filename):

    filename = secure_filename(file.filename)

    file_path = os.path.join(
```

```
        application.config["UPLOAD_DIR"],
        "{0}_{1}".format(time.strftime("%H-%M-%S_%d-%m-%Y"), filename),
    )
    file.save(file_path)
    rid = RiskAnalyzer()
    permissions = rid.get_permission_json(file_path)


    try:
        response = {
            "name": filename,
            "md5": md5sum(file_path),
            "risk": round(
                rid.calculate_risk(rid.get_feature_vector_from_json(permissions)),
                3,
            ),
            "permissions": [
                val
                for val in list(
                    map(
                        lambda x: {"cat": "Declared", "name": x},
                        permissions["declared"],
                    )
                )
                + list(
                    map(
```

```
                    lambda x: {"cat": "Required and Used", "name": x},
                    permissions["requiredAndUsed"],
                )
            )
            + list(
                map(
                    lambda x: {"cat": "Required but Not Used", "name": x},
                    permissions["requiredButNotUsed"],
                )
            )
            + list(
                map(
                    lambda x: {"cat": "Not Required but Used", "name": x},
                    permissions["notRequiredButUsed"],
                )
            )
        ],
    }
    return make_response(jsonify(response))
  except Exception:
    raise BadRequest("The uploaded file is not valid")
else:
  raise UnprocessableEntity("The uploaded file is not valid")


@application.route("/apks", methods=["GET"], strict_slashes=False)
```

```
def get_apks():

  query = Apk.query

  if request.args.get("sort") and request.args.get("sort_dir"):

      query = query.order_by(

          text(

              "{0} {1}".format(request.args.get("sort"), request.args.get("sort_dir"))

          )

      )



  if request.args.get("namefil"):

      fil = request.args.get("namefil").replace("%", "\\%").replace("_", "\\_")

      query = query.filter(Apk.name.ilike("%{0}%".format(fil), "\\"))



  if request.args.get("md5fil"):

      fil = request.args.get("md5fil").replace("%", "\\%").replace("_", "\\_")

      query = query.filter(Apk.md5.ilike("%{0}%".format(fil), "\\"))



  if request.args.get("riskfil"):

      fil = request.args.get("riskfil").replace("%", "\\%").replace("_", "\\_")

      query = query.filter(cast(Apk.risk, db.String).ilike("%{0}%".format(fil), "\\"))



  pag = query.paginate()

  item_list = []



  for item in pag.items:
```

```
        item_list.append({"name": item.name, "md5": item.md5, "risk": item.risk})


    response = {"current_page": pag.page, "last_page": pag.pages, "data": item_list}
    return make_response(jsonify(response))


@application.route("/details", methods=["GET", "POST"], strict_slashes=False)
def get_apk_details():
    if request.method == "GET":
        try:
            # An exception will be thrown if the query string doesn't contain an md5.
            md5 = request.args["md5"]
            apk = Apk.query.get(md5)
            response = {
                "name": apk.name,
                "md5": apk.md5,
                "risk": apk.risk,
                "type": apk.type,
                "source": apk.source,
                "permissions": [
                    val
                    for val in list(
                        map(
                            lambda x: {"cat": "Declared", "name": x.name},
                            apk.declared_permissions,
                        )
```

```
            )
            + list(
                map(
                    lambda x: {"cat": "Required and Used", "name": x.name},
                    apk.required_and_used_permissions,
                )
            )
            + list(
                map(
                    lambda x: {"cat": "Required but Not Used", "name": x.name},
                    apk.required_but_not_used_permissions,
                )
            )
            + list(
                map(
                    lambda x: {"cat": "Not Required but Used", "name": x.name},
                    apk.not_required_but_used_permissions,
                )
            )
        ],
    }
    return make_response(jsonify(response))
except Exception:
    raise BadRequest("Unable to get details for the specified application")
```

```
    if request.method == "POST":
        response = {
            "name": request.form["name"],
            "md5": request.form["md5"],
            "risk": request.form["risk"],
            "permissions": request.form["permissions"],
        }
        return render_template("details.html", apk=response)


def md5sum(file_path, block_size=65536):
    md5_hash = hashlweb.md5()
    with open(file_path, "rb") as filename:
        for chunk in iter(lambda: filename.read(block_size), b""):
            md5_hash.update(chunk)
    return md5_hash.hexdigest()


if __name__ == "__main__":
    application.run(host="0.0.0.0", port=5000)
```

- model.py

```
#!/usr/bin/env python3
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()
declared_permissions = db.Table(
    "declared_permissions",
    db.metadata,
```

```
    db.Column("apk_id", db.String(32), db.ForeignKey("apks.md5"), primary_key=True),
    db.Column(
        "permission_id", db.Integer, db.ForeignKey("permissions.id"), primary_key=True
    ),
)

required_and_used_permissions = db.Table(
    "required_and_used_permissions",
    db.metadata,
    db.Column("apk_id", db.String(32), db.ForeignKey("apks.md5"), primary_key=True),
    db.Column(
        "permission_id", db.Integer, db.ForeignKey("permissions.id"), primary_key=True
    ),
)

required_but_not_used_permissions = db.Table(
    "required_but_not_used_permissions",
    db.metadata,
    db.Column("apk_id", db.String(32), db.ForeignKey("apks.md5"), primary_key=True),
    db.Column(
        "permission_id", db.Integer, db.ForeignKey("permissions.id"), primary_key=True
    ),
)

not_required_but_used_permissions = db.Table(
```

```
  "not_required_but_used_permissions",
  db.metadata,
  db.Column("apk_id", db.String(32), db.ForeignKey("apks.md5"), primary_key=True),
  db.Column(
    "permission_id", db.Integer, db.ForeignKey("permissions.id"), primary_key=True
  ),
)


class Apk(db.Model):
  __tablename__ = "apks"
  md5 = db.Column(db.String(32), primary_key=True)
  type = db.Column(db.String(10), nullable=False)
  source = db.Column(db.String(24), nullable=False)
  name = db.Column(db.String(255), nullable=False)
  risk = db.Column(db.Float, nullable=False)
  declared_permissions = db.relationship(
    "Permission",
    secondary=declared_permissions,
    backref=db.backref("app_declaring", lazy="dynamic"),
  )


  required_and_used_permissions = db.relationship(
    "Permission",
    secondary=required_and_used_permissions,
    backref=db.backref("app_requiring_and_using", lazy="dynamic"),
```

```
  )


  required_but_not_used_permissions = db.relationship(
      "Permission",
      secondary=required_but_not_used_permissions,
      backref=db.backref("app_requiring_but_not_using", lazy="dynamic"),
  )


  not_required_but_used_permissions = db.relationship(
      "Permission",
      secondary=not_required_but_used_permissions,
      backref=db.backref("app_not_requiring_but_using", lazy="dynamic"),
  )


  def __repr__(self):
      return '<Apk (md5="{0}", name="{1}", risk="{2}")>'.format(
          self.md5, self.name, self.risk
      )


class Permission(db.Model):
  __tablename__ = "permissions"


  id = db.Column(db.Integer, primary_key=True)
  name = db.Column(db.String(255), unique=True, nullable=False)
```

```
def __repr__(self):

    return '<Permission (name="{0}")>'.format(self.name)
```

Results:

RiskAnalyzer has been developed in Python and implements the selected four classifiers. For each app A, RiskAnalyzer calculates the RIV on all four APs sets (i.e., DAP_A , EAP_A , GAP_A, and UAP_A), by combining the corresponding feature vectors in a unique one, i.e., $FV^A_{all} = FV^A_{DAP} || FV^A_{EAP} || FV^A_{GAP} || FV^A_{UAP}$. The RIV is calculated as the average score value of all four classifiers.

To train each classifier in RiskAnalyzer, I applied the 10-fold cross validation on one of the three sets used to evaluate the classifiers. I also used the same set to empirically assess whether applying all four APs sets may improve the accuracy. To this aim, my tests returned the following average accuracy values: 92.93% for DAP, 88.36% for EAP, 79.12% for GAP, 91.09% for UAP, and 94.87% for all sets. Therefore, I chose to consider all sets.

Advantages:

RiskAnalyzer does not take into consideration only the permissions declared into the app manifest unlike other tools, but carries out reverse engineering on the apps to retrieve the bytecode and then infers (through static analysis) which permissions are actually used and which not, extracting in this way 4 sets of permissions for every analyzed app.

Hence, the intrinsic limitations of probabilistic methods applied to Android Permissions (hereafter, APs) can be overcome by machine learning techniques able to build up more reliable RIVs.

Disadvantages:

Performance of RiskAnalyzer: The performance of RiskAnalyzer has been evaluated on a general-purpose Laptop equipped with an Intel i5 GHz processor, and 8GB RAM. Performance of classifiers is evaluated in terms of average time and standard deviation, during the training and the testing phase. Using all sets decreases the average performance up to 240% during the training phase.

However, it is worth noticing that this phase is executed once at the beginning. Instead, the testing phase is very quick and lasts in a few millisecs both with one and all sets, thereby suggesting to adopt all four sets to obtain a higher accuracy.

Novelty of the Solution:

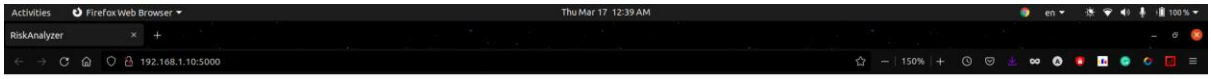
In this solution, I empirically assessed the reliability of probabilistic risk index approaches for Android apps, and I proposed a novel methodology based on machine learning aimed at overcoming the shortcomings of the probabilistic solutions. I implemented the methodology in a tool, RiskAnalyzer, that I empirically evaluated.

Application:

RiskAnalyzer provides a metric to users for evaluating the trustworthiness of the apps they are going to install. Android is still the most widespread mobile operating system in the world, as more than 300 million Android-enabled smartphones have been sold only in the third trimester of 2016. Android-enabled smartphones remain a sensitive target for malware that aim at exploiting its diffusion to reach a high number of potential victims. Since users have access to a high number of apps through public markets and external web sites, they need reliable tools to rate the trustworthiness of apps they are going to install.

Screenshots of Results:

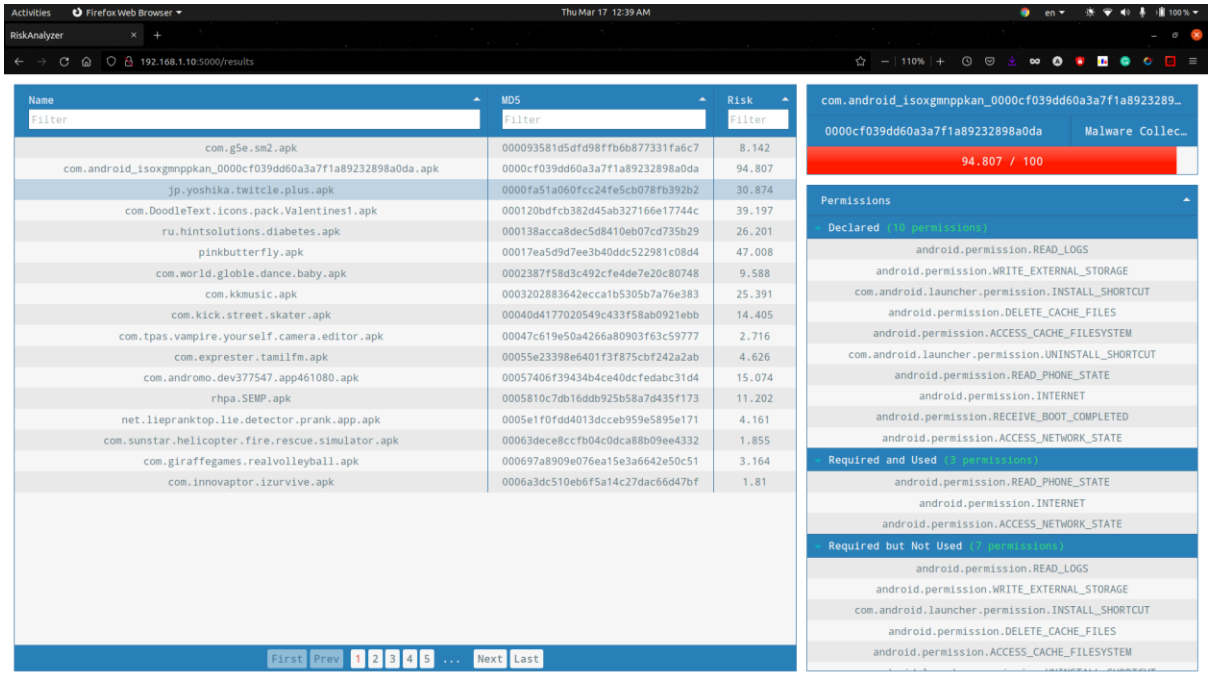
Results through a web interface and calculating the risk of new applications (by uploading the .apk file):



RiskAnalyzer

RiskAnalyzer is a tool for quantitative risk analysis of Android applications based on machine learning techniques. The tool uses classification techniques in order to generate a numeric risk value between 0 and 100 for a given app.

Analyze application | Experimental results



Name	MD5	Risk
com.g5e.sm2.apk	000093581d5dfd98ffb6b877331fa6c7	8.142
com.android_isoxgmnppkan_0000cf039dd60a3a7f1a89232898a0da.apk	0000cf039dd60a3a7f1a89232898a0da	94.807
jp.yoshika.twitcle.plus.apk	0000fa51a060fcc24fe5cb078fb392b2	30.874
com.DoodleText.icons.pack.Valentines1.apk	000120bdfcb382d45ab327166e17744c	39.197
ru.hintsolutions.diabetes.apk	000138acca8dec5d8410eb07cd735b29	26.201
pinkbutterfly.apk	00017ea5d9d7ee3b40ddc522981c08d4	47.008
com.world.globle.dance.baby.apk	0002387f58d3c492cfe4de7e20c80748	9.588
com.kkmusic.apk	0003202883642ecca1b5305b7a76e383	25.391
com.kick.street.skater.apk	00040d4177020549c433f58ab0921ebb	14.405
com.tpas.vampire.yourself.camera.editor.apk	00047c619e50a4266a80903f63c59777	2.716
com.exprester.tamilfm.apk	00055e23398e6401f3f875cbf242a2ab	4.626
com.andromo.dev377547.app461080.apk	00057406f39434b4ce40dcfedabc31d4	15.074
rhpa.SEMP.apk	0005810c7db16ddb925b58a7d435f173	11.202
net.liepranktop.lie.detector.prank.app.apk	0005e1f0fdd4013dcceb959e5895e171	4.161
com.sunstar.helicopter.fire.rescue.simulator.apk	00063dece8ccfb04c0dca88b09ee4332	1.855
com.giraffegames.realvolleyball.apk	000697a8909e076ea15e3a6642e50c51	3.164
com.innovaptor.izurvive.apk	0006a3dc510eb6f5a14c27dac66d47bf	1.81

First Prev 1 2 3 4 5 ... Next Last

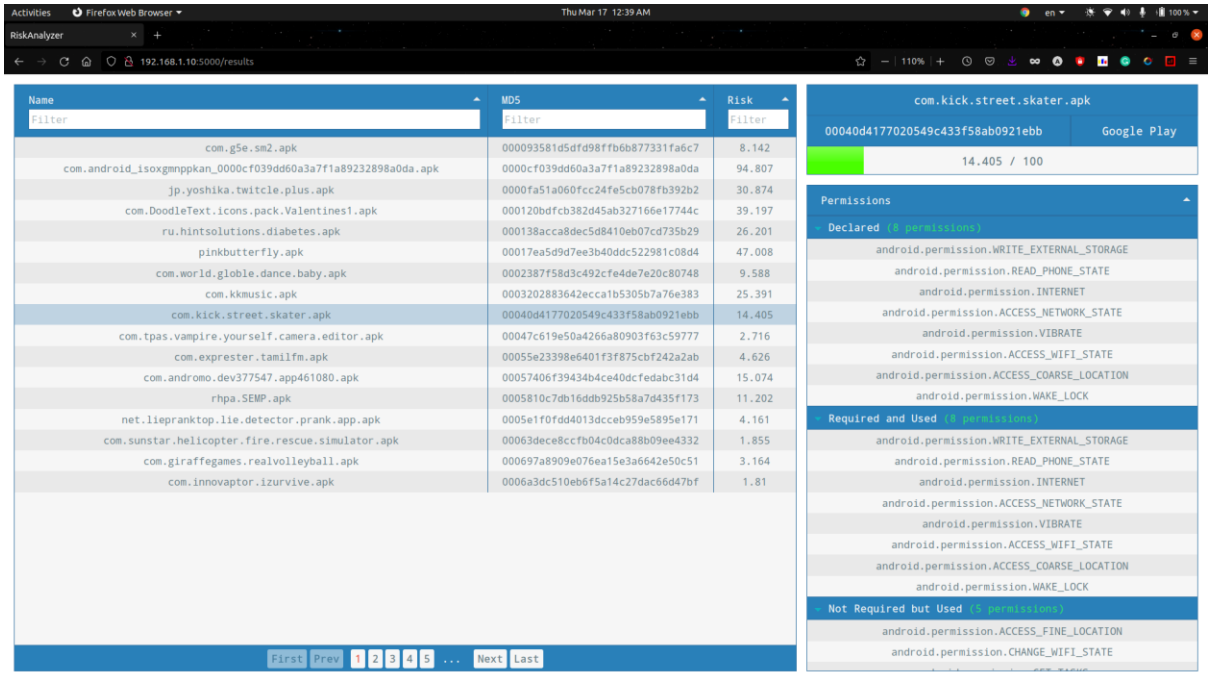
com.android_isoxgmnppkan_0000cf039dd60a3a7f1a8923289…

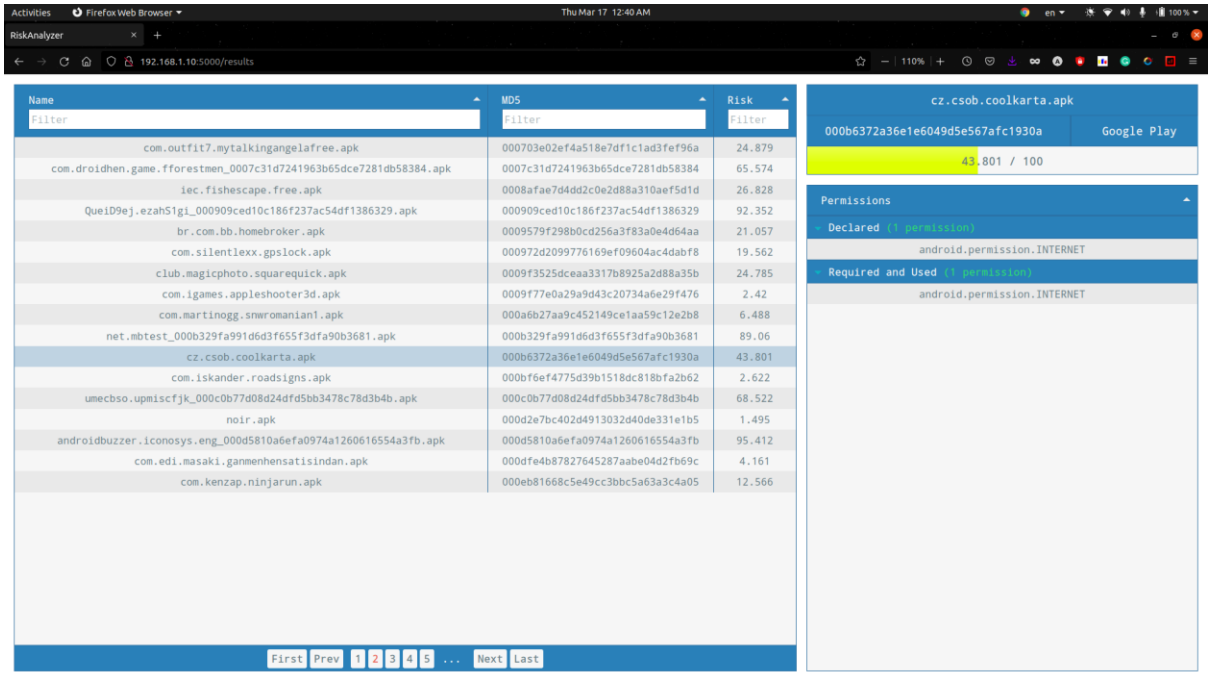
0000cf039dd60a3a7f1a89232898a0da | Malware Collec…

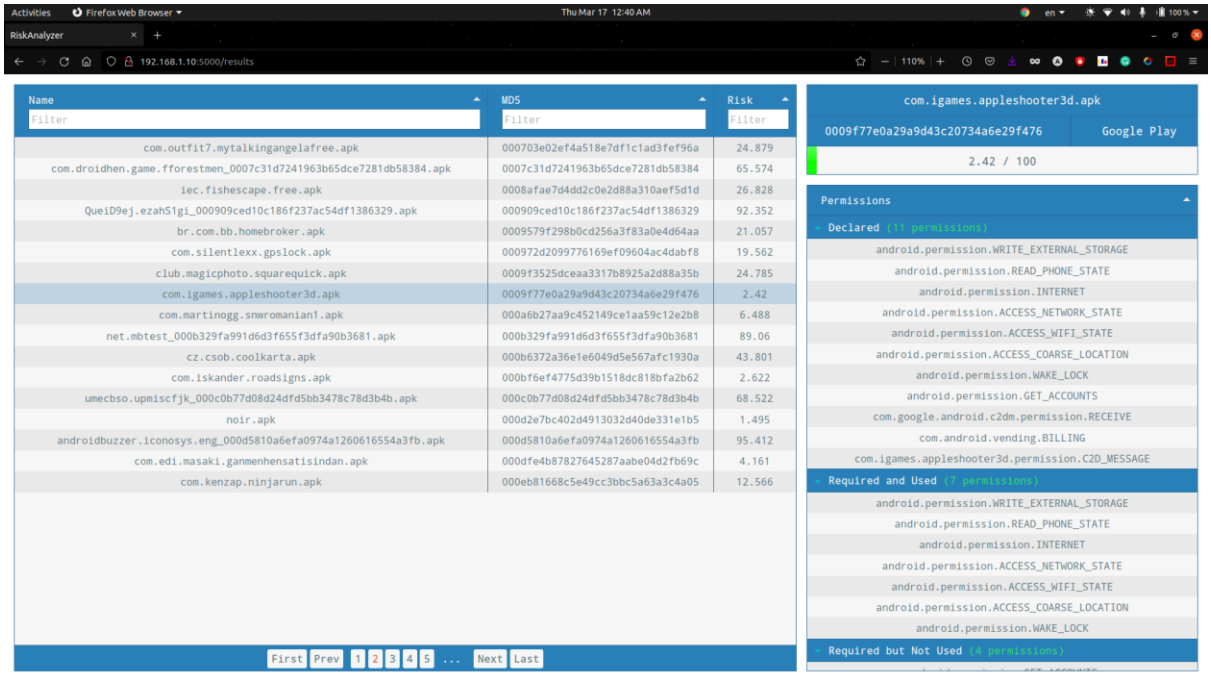
94.807 / 100

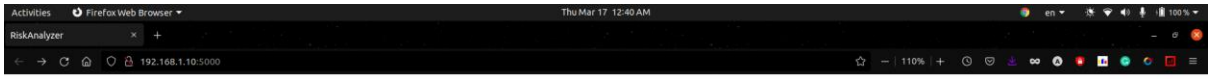
Permissions

- Declared (10 permissions)
  - android.permission.READ_LOGS
  - android.permission.WRITE_EXTERNAL_STORAGE
  - com.android.launcher.permission.INSTALL_SHORTCUT
  - android.permission.DELETE_CACHE_FILES
  - android.permission.ACCESS_CACHE_FILESYSTEM
  - com.android.launcher.permission.UNINSTALL_SHORTCUT
  - android.permission.READ_PHONE_STATE
  - android.permission.INTERNET
  - android.permission.RECEIVE_BOOT_COMPLETED
  - android.permission.ACCESS_NETWORK_STATE
- Required and Used (3 permissions)
  - android.permission.READ_PHONE_STATE
  - android.permission.INTERNET
  - android.permission.ACCESS_NETWORK_STATE
- Required but Not Used (7 permissions)
  - android.permission.READ_LOGS
  - android.permission.WRITE_EXTERNAL_STORAGE
  - com.android.launcher.permission.INSTALL_SHORTCUT
  - android.permission.DELETE_CACHE_FILES
  - android.permission.ACCESS_CACHE_FILESYSTEM


Activities
Firefox Web Browser
Thu Mar 17 12:39 AM
en
RiskAnalyzer
192.168.1.10:5000/results
110%
Name
MD5
Risk
Filter
com.g5e.sm2.apk 000093581d5dfd98ffb6b877331fa6c7 8.142
com.android_isoxgmnppkan_0000cf039dd60a3a7f1a89232898a0da.apk 0000cf039dd60a3a7f1a89232898a0da 94.807
jp.yoshika.twitcle.plus.apk 0000fa51a060fcc24fe5cb078fb392b2 30.874
com.DoodleText.icons.pack.Valentines1.apk 000120bdfcb382d45ab327166e17744c 39.197
ru.hintsolutions.diabetes.apk 000138acca8dec5d8410eb07cd735b29 26.201
pinkbutterfly.apk 00017ea5d9d7ee3b40ddc522981c08d4 47.008
com.world.globle.dance.baby.apk 0002387f58d3c492cfe4de7e20c80748 9.588
com.kkmusic.apk 0003202883642ecca1b5305b7a76e383 25.391
com.kick.street.skater.apk 00040d4177020549c433f58ab0921ebb 14.405
com.tpas.vampire.yourself.camera.editor.apk 00047c619e50a4266a80903f63c59777 2.716
com.exprester.tamilfm.apk 00055e23398e6401f3f875cbf242a2ab 4.626
com.andromo.dev377547.app461080.apk 00057406f39434b4ce40dcfedabc31d4 15.074
rhpa.SEMP.apk 0005810c7db16ddb925b58a7d435f173 11.202
net.liepranktop.lie.detector.prank.app.apk 0005e1f0fdd4013dcceb959e5895e171 4.161
com.sunstar.helicopter.fire.rescue.simulator.apk 00063dece8ccfb04c0dca88b09ee4332 1.855
com.giraffegames.realvolleyball.apk 000697a8909e076ea15e3a6642e50c51 3.164
com.innovaptor.izurvive.apk 0006a3dc510eb6f5a14c27dac66d47bf 1.81
First Prev 1 2 3 4 5 ... Next Last
com.kick.street.skater.apk
00040d4177020549c433f58ab0921ebb
Google Play
14.405 / 100
Permissions
Declared (8 permissions)
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.VIBRATE
android.permission.ACCESS_WIFI_STATE
android.permission.ACCESS_COARSE_LOCATION
android.permission.WAKE_LOCK
Required and Used (8 permissions)
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.VIBRATE
android.permission.ACCESS_WIFI_STATE
android.permission.ACCESS_COARSE_LOCATION
android.permission.WAKE_LOCK
Not Required but Used (5 permissions)
android.permission.ACCESS_FINE_LOCATION
android.permission.CHANGE_WIFI_STATE


Activities
Firefox Web Browser
Thu Mar 17 12:40 AM
en
RiskAnalyzer
192.168.1.10:5000/results
110%
Name
MD5
Risk
Filter
com.outfit7.mytalkingangelafree.apk 000703e02ef4a518e7df1c1ad3fef96a 24.879
com.droidhen.game.fforestmen_0007c31d7241963b65dce7281db58384.apk 0007c31d7241963b65dce7281db58384 65.574
iec.fishescape.free.apk 0008afae7d4dd2c0e2d88a310aef5d1d 26.828
QueiD9ej.ezahS1gi_000909ced10c186f237ac54df1386329.apk 000909ced10c186f237ac54df1386329 92.352
br.com.bb.homebroker.apk 0009579f298b0cd256a3f83a0e4d64aa 21.057
com.silentlexx.gpslock.apk 000972d2099776169ef09604ac4dabf8 19.562
club.magicphoto.squarequick.apk 0009f3525dceaa3317b8925a2d88a35b 24.785
com.igames.appleshooter3d.apk 0009f77e0a29a9d43c20734a6e29f476 2.42
com.martinogg.snwromanian1.apk 000a6b27aa9c452149ce1aa59c12e2b8 6.488
net.mbtest_000b329fa991d6d3f655f3dfa90b3681.apk 000b329fa991d6d3f655f3dfa90b3681 89.06
cz.csob.coolkarta.apk 000b6372a36e1e6049d5e567afc1930a 43.801
com.iskander.roadsigns.apk 000bf6ef4775d39b1518dc818bfa2b62 2.622
umecbso.upmiscfjk_000c0b77d08d24dfd5bb3478c78d3b4b.apk 000c0b77d08d24dfd5bb3478c78d3b4b 68.522
noir.apk 000d2e7bc402d4913032d40de331e1b5 1.495
androidbuzzer.iconosys.eng_000d5810a6efa0974a1260616554a3fb.apk 000d5810a6efa0974a1260616554a3fb 95.412
com.edi.masaki.ganmenhensatisindan.apk 000dfe4b87827645287aabe04d2fb69c 4.161
com.kenzap.ninjarun.apk 000eb81668c5e49cc3bbc5a63a3c4a05 12.566
First Prev 1 2 3 4 5 ... Next Last
cz.csob.coolkarta.apk
000b6372a36e1e6049d5e567afc1930a
Google Play
43.801 / 100
Permissions
Declared (1 permission)
android.permission.INTERNET
Required and Used (1 permission)
android.permission.INTERNET





RiskAnalyzer

RiskAnalyzer is a tool for quantitative risk analysis of Android applications based on machine learning techniques. The tool uses classification techniques in order to generate a numeric risk value between 0 and 100 for a given app.

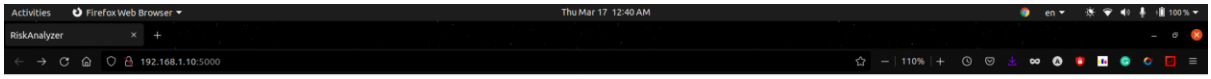
Drop an application here or click to choose one
(max. 100 MB)

RiskAnalyzer

RiskAnalyzer is a tool for quantitative risk analysis of Android applications based on machine learning techniques. The tool uses classification techniques in order to generate a numeric risk value between 0 and 100 for a given app.

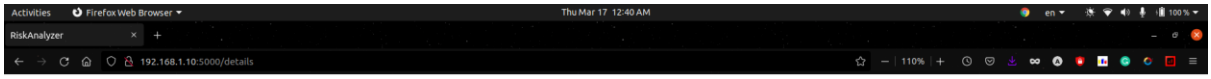
spotify.apk (44.2 MB)

Submit



RiskAnalyzer

RiskAnalyzer is a tool for quantitative risk analysis of Android applications based on machine learning techniques. The tool uses classification techniques in order to generate a numeric risk value between 0 and 100 for a given app.



spotify.apk

461d1739661214bb326a5b008e992690

13.787 / 100

Permissions

Declared (29 permissions)

com.samsung.android.app.spage.permission.READ_CARD_DATA
android.permission.GET_ACCOUNTS
com.samsung.android.app.spage.permission.WRITE_CARD_DATA
android.permission.FOREGROUND_SERVICE
android.permission.BLUETOOTH_ADMIN
com.spotify.music.permission.C2D_MESSAGE
android.permission.NFC
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.WRITE_EXTERNAL_STORAGE
com.spotify.music.permission.SECURED_BROADCAST
android.permission.ACCESS_NETWORK_STATE
com.samsung.WATCH_APP_TYPE.Companion
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.MANAGE_ACCOUNTS
com.google.android.c2dm.permission.RECEIVE
com.android.launcher.permission.INSTALL_SHORTCUT
com.samsung.android.rubin.context.permission.READ_CONTEXT_MANAGER
com.spotify.music.permission.INTERNAL_BROADCAST
android.permission.CHANGE_WIFI_MULTICAST_STATE
android.permission.BROADCAST_STICKY

spotify.apk

461d1739661214bb326a5b008e992690

13.787 / 100

Permissions

android.permission.READ_EXTERNAL_STORAGE
android.permission.BLUETOOTH
android.permission.READ_PHONE_STATE
com.sony.snei.np.android.account.provider.permission.DUID_READ_PROVIDER

Required and Used (9 permissions)

android.permission.BROADCAST_STICKY
android.permission.GET_ACCOUNTS
android.permission.INTERNET
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.MODIFY_AUDIO_SETTINGS
android.permission.BLUETOOTH
android.permission.MANAGE_ACCOUNTS
android.permission.READ_PHONE_STATE

Required but Not Used (20 permissions)

com.samsung.android.app.spage.permission.READ_CARD_DATA
com.samsung.android.app.spage.permission.WRITE_CARD_DATA
android.permission.FOREGROUND_SERVICE
android.permission.BLUETOOTH_ADMIN
com.spotify.music.permission.C2D_MESSAGE
android.permission.NFC
android.permission.RECEIVE_BOOT_COMPLETED

spotify.apk

461d1739661214bb326a5b008e992690

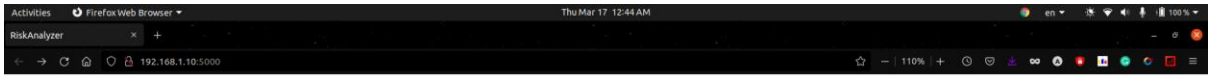
13.787 / 100

Permissions

- android.permission.NFC
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.WRITE_EXTERNAL_STORAGE
- com.spotify.music.permission.SECURED_BROADCAST
- android.permission.ACCESS_NETWORK_STATE
- com.samsung.WATCH_APP_TYPE.Companion
- com.google.android.c2dm.permission.RECEIVE
- com.android.launcher.permission.INSTALL_SHORTCUT
- com.samsung.android.rubin.context.permission.READ_CONTEXT_MANAGER
- com.spotify.music.permission.INTERNAL_BROADCAST
- android.permission.CHANGE_WIFI_MULTICAST_STATE
- android.permission.USE_CREDENTIALS
- com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
- android.permission.READ_EXTERNAL_STORAGE
- com.sony.snei.np.android.account.provider.permission.DUID_READ_PROVIDER

Not Required but Used (6 permissions)

- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.AUTHENTICATE_ACCOUNTS
- android.permission.VIBRATE
- android.permission.READ_PROFILE
- android.permission.WRITE_SETTINGS



RiskAnalyzer

RiskAnalyzer is a tool for quantitative risk analysis of Android applications based on machine learning techniques. The tool uses classification techniques in order to generate a numeric risk value between 0 and 100 for a given app.

InsecureBankv2.apk (3.3 MB)

Submit

Comparison of Existing Solutions:

The scientific literature related to risk analysis of Android apps is rather limited and mostly focused on APs, so I also take into account works regarding malware classification because I expect to see some relationships between malware and high-risk apps. Currently available proposals are probabilistic, i.e., the RIV indicates the probability that an app can be a malware, according to statistical analysis carried out on datasets containing both apps (that are expected to be mostly benign) and Ill-known malware samples. In an existing solution, authors propose a method for detecting risk signals according to the frequency of security-sensitive APs. The RIV is calculated according to Bayesian probabilistic models that compare the APs required by each app with those requested by other apps in the same category (that must be known a priori). Furthermore, authors define three properties that should be granted by any probabilistic function calculating a RIV for apps, namely, i) monotonicity (i.e., removing an AP should lower the RIV), ii) coherence (i.e., malware should have higher RIVs than apps), and iii) ease of understanding (i.e., the RIV of an app should be clearly understandable to the user, and it should allow straightforward comparison among values).

Also, another existing solution proposes a methodology for calculating a RIV for apps according to their category. More specifically, for each category, the kind and number of required APs are empirically inferred, thereby identifying permission patterns belonging to apps in each category. Then, the RIV is calculated by measuring a distance between the set of APs required by the app and the permission patterns of its category. Notwithstanding the encouraging empirical results obtained on a dataset made by 7.737 apps and 1.260 malware samples, the main limitation of the approach is in the need to know in advance the category

of the app. Such information can be often unreliable as categories are manually chosen by developers.

Maetroid evaluates app risk according to both APs and metadata information related to the developer's reputation and the source app market. The risk is calculated according to declared APs only, and by assigning static weights to each AP. Maetroid does not provide a quantitative RIV, but assigns each app in one (out of three) risk category. A framework for app risk analysis is made by three layers carrying out static, dynamic and behavioural analysis, respectively. The framework combines the results from each layer and builds up the RIV. Unluckily, the framework is purely theoretical and lacks any empirical evaluation, thereby making it difficult to assess the viability of the approach. DroidRisk is a quantitative method for calculating a RIV. DroidRisk is trained on a set of 27.274 apps and 1.260 malware samples, whereby it calculates the distribution of declared APs (i.e., those contained in the Android Manifest file). Then, DroidRisk applies a probabilistic function that calculates a RIV according to the kind and the potential impact of APs required by the app.

I argue that probabilistic methods suffer from some limitations.

1. They are unable to recognize as dangerous the malware that requires a limited set of APs; conversely, they averagely provide high RIVs for apps requiring many APs.
2. Current proposals deal with declared APs only, without deepening, for instance, which APs are actually exploited by the app. Due to the monotonicity of probabilistic risk indexes, relying only on declared permissions can impact the reliability, as apps are often overprivileged by their developers and can therefore obtain too high RIVs.
3. Probabilistic methods statically define the impact of APs, that is, all APs belonging to the same category (e.g., Normal, Dangerous, Signature, SignatureOrSystem) equally impact the estimation of the RIV. This choice does not allow to provide different impacts to APs, e.g., according to their distribution on the set of malware.

I argue that more reliable RIVs can be obtained through a machine learning approach based on – four sets of permissions for each app A, namely

1. Declared Permissions (DAP_A), i.e., declared in the Android Manifest file;
2. Exploited permissions (EAP_A), i.e., APs that are actually exploited in the app code;
3. Ghost permissions (GAP_A), i.e., APs that the app tries to exploit in the code, but they are not declared in the Android Manifest file;
4. Useless permissions (UAP_A), i.e., declared APs that are not exploited in the app code.

Conclusion and Future Development:

In this project I empirically assessed the reliability of probabilistic risk index approaches for Android apps, and proposed a methodology based on machine learning aimed at overcoming the shortcomings of the probabilistic solutions.

Future development of this project includes extending the feature set beyond APs, by taking into account suspicious API calls and URLs, both recognizable in the bytecode through the static analysis technique I adopted to build the permission sets.

X-X-X-X