Project 4: Classify Android Goodware VS. Malware

This project is due on **April 13th, 2024** at **11:59pm CST**. We strongly recommend that you get started early. You will get 80% of your total points if you turn in up to 72 hours after the due date. Late work will not be accepted after 72 hours past the due date.

The project is split into two parts. Checkpoint 1 helps you to get familiar with tools you will be using for this project and how to extract features from an Android APK. The recommended deadline for checkpoint 1 is **March 31th, 2024**. We strongly recommend that you finish the first checkpoint before the recommended deadline. However, you do NOT need to make a separate submission for checkpoint 1. That is, you need to submit your answers for both checkpoints in a single folder before the project due date on **April 13th, 2024** at **11:59pm CST**. Detailed submission requirements are listed at the end of the document.

This is an individual project; you SHOULD work **individually**.

The code and other answers you submit must be entirely your own work, and you are bound by the Student Code. You MAY consult with other students about the conceptualization of the project and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone else. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions MUST be submitted electronically on Github repository, following the submission checklist given at the end of each checkpoint. Details on the filename and submission guideline is listed at the end of the document.

Release date: March 21st, 2024
Checkpoint 1 Recommended Due date: March 31th, 2024
Checkpoint 2 Due date: April 13th, 2024 at 11:59pm CST

Introduction

Android devices have been widely adopted and are capable of storing and accessing significant private and confidential information. Many malware developers started to target android devices and steal private information. In 2021, AV-Test (`https://www.av-test.org/en/statistics/malware/`) recorded almost 3.4 million new pieces of malware for Android alone and has already seen around 1 million malware in 2022. This has created an immediate need for security professionals to build detection frameworks and classify malware from all the applications in the market.

Please read the assignment carefully before you start implementing.

Objectives

- Know how to extract features from APKs with static analysis.
- Be able to build an ML classifier to distinguish malware from goodware.
- Create adversarial samples with the consideration of realizability of Android apps.

Checkpoints

This machine problem is split into 2 checkpoints. Checkpoint 1 will help you get familiar with what features are commonly extracted from Android apps and how to extract them. You will extract features for 2 apks with the apktool. In Checkpoint 2, you will be given an Android dataset (with feature vectors and ground-truth labels). You need to implement a LinearSVM classifier to predict whether a sample is benign or malicious, manipulate a few features (with and without realizability constraints) to create adversarial samples, and answer some questions based on the user graph.

Implementations

To help you with the implementation, you are provided with a code skeleton which prototypes the main classes and includes some useful methods. You are free to modify the provided code as long as they generate correct results.

4.1 Checkpoint 1 (20 points)

This machine problem is split into 2 checkpoints. Checkpoint 1 will help you get familiar with the Android feature extraction. You don't need to write any code for this part, all you need to do is to set up the appropriate environment and run the tool. You are also required to answer questions on Coursera titled 'MP4: Android Malware'.

4.1.1 Environment Setup

The tool we are using to extract features from Android apks is called apktool. This is a tool for reverse engineering 3rd party, closed, binary Android apps. You can install apktool using Docker by following section 4.1.1.1 (recommended), or alternatively install it on your local system by following instructions in section 4.1.1.2.

4.1.1.1 Set up Dockerfile Environment

1. We have provided a dockerfile, which you are not required to use, but we recommend doing so. If you are unfamiliar with docker, please read this overview: `https://docs.docker.com/get-started/`.
2. Docker can be installed using the instructions listed here: `https://docs.docker.com/get-docker/`.
3. To build the provided dockerfile, navigate to the location of the dockerfile in your command prompt and run the command `docker build -t apkdocker`. Note: you can replace apkdocker with any name you wish to give this image.
4. To run the built docker image, use `docker run -it apkdocker`. This will open a terminal in that image with the required setup.

4.1.1.2 APK Tool

The apktool will allow us to statically analyze the apk files that have been provided.

1. To install apktool follow the instructions provided here `https://ibotpeaches.github.io/Apktool/install/`. If you are using the dockerfile environment, use the instructions provided for the Linux environment.
2. Alternatively, you can download Homebrew using this command: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"` Note: There will be two required updates at the end of the comments, `(echo; echo 'eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"') >> /root/.profile` and `eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"`. Once you have Homebrew installed, use the command `brew install apktool`
3. Once the apktool has been installed, run the analysis on each of the apk files using the command `apktool d <apkfile>` where <apkfile> should be replaced with the name of the apk file (with the .apk extension) to disassemble.

What to submit

- AndroidManifest_49875A.xml
- AndroidManifest_5E06B7.xml

4.2 Checkpoint 2 (75 points)

For this checkpoint, you'll need to implement a LinearSVM classification model to differentiate malware and goodware. And also manually create some adversarial samples to make your trained classifier misclassify the sample after your perturbation. We have provided a .json file for both apks with the features that were extracted during Checkpoint 1 analysis that you will use as baseline samples.

4.2.1 Environment Setup

Due to GitHub storage limit, you need to unzip "data/apg-X.json.zip" first, the unzipped file should be "data/apg-X.json".

You can follow the steps to install these libraries (If you already set it up during MP1, you only need to do the last two steps:

1. Install Anaconda here: `https://www.anaconda.com/products/distribution`.
2. Create a virtual environment named as cs463 with Python 3.8.8: `conda create -n cs463 python==3.8.8`
3. Activate the virtual environment: `conda activate cs463`
4. `pip install -r requirements.txt` (the one in the root folder of MP4 with numpy and scikit-learn only.)

4.2.2 Train an SVM model (20 points)

SVM is very popular in Android malware detection. You need to implement the "fit()" function of class SVM in "mp4_model.py" and return the classifier. Since our "perform_feature_selection()" is pretty generic, you should also implement half of it to deal with the case that we actually don't do any feature selection. You should carefully read how the performance is calculated and F_1, precision, and recall on the testing set would be returned. Another function you need to implement is "stat()" in "main.py". You should calculate how many samples were used in training; how many samples were from testing; how many malware and goodware in the training set, and how many malware and goodware in the testing set. Think about if the goodware VS malware ratio matches with your common sense.

Tips

- Please refer to Sciki-learn documentation on LinearSVC here: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`.

What to submit

- "main.py".
- You can submit "mp4_model.py" together with the next subsection.

4.2.3 Feature Selection (15 points)

If you take a look at the shape of "svm_model.X_train" in "main.py", you will notice that the original dataset is very sparse and could easily go over to more than 200 thousand features. This is neither efficient nor effective. So you need to implement the "perform_feature_selection()" function in "mp4_model.py". To achieve slightly better performance, you can use Scikit-learn LinearSVC module to perform a feature selection to reduce the feature dimension to 5,000. Please feel free to try other dimensions. The final code would be graded based on 5,000.

What to submit

- `mp4_model.py`
- Implementations for this section.
- all_feature_names_5000.json, generated by your model.
- Please put your thought as code comments to compare the performance of with and without feature selection and why do you think one could achieve a little bit better results.

4.2.4 Create Adversarial Samples

You should have learned about adversarial samples from the lecture. But how could we generate them? There are many optimization-based methods but here we want to focus on the simplicity. Given the two apks provided in Checkpoint 1 in the "cp1_feature_extract/mp4_apks" folder, 49875AXXX is malware while 5E06B7XXX is goodware. Both of them are correctly predicted as their corresponding label given the classifier trained using 5000 features (already verified). Your goal is to change some feature values to artificially create a new sample and it would be misclassified as "benign/malicious". For example, 49875AXXX should be predicted as benign by the original classifier while 5E06B7XXX was predicted as malicious. To make it easier, I have provided the name of the 5000 features in "data/all_feature_names_5000
_example.json". This file corresponds to the feature vector. You can use the first few lines of "train_model()" to generate your feature names. Theoretically, they should be the same as the one I provided. To create an adversarial example, usually you should find some features and change their values to make them misclassified. But in the context of security, there are more constraints compared to common Machine Learning domains such as images.

1. To protect the integrity of these apks, usually we can only add features instead of removing features because the latter can easily break the dependencies.

2. Imagine that we need the adversarial sample to be effective in the real world, we need to make sure that it still keep the original malicious / benign functionality, this is usually called problem space attack compared to commonly used feature space attack (more details can be found here: `https://arxiv.org/abs/1911.02142`). In reality, it's usually difficult to add some features like the activity or API call or other features that need to add code. So similar to `https://www.usenix.org/conference/usenixsecurity21/presentation/severi`, we only allow certain types of features can be added, i.e., the features from the Android Manifest file which does not affect the code logic at all.

3. You should not add too many features which would make the sample suscipious.

To perform the problem space attack, you should follow these guidelines:

1. You can only add features. That means, you can choose some features and mark their values as 1 and add to the original feature vector of the sample, see if they can be misclassified by your SVM model (trained using 5000 features).

2. You can start with 49875AXXX.json and 5E06B7XXX.json as the base features for the two samples and use data/added-features-xxx to add features to make it adversarial.

3. You can at most add 30 features. We need to stay stealthy.

4. After adding features, you will need to merge and create one feature vector to run the classification (more details in the verify_adv.py)

5. You should only choose feature names start with "app_permission" or "api_permission".

6. You should complete script "verify_adv.py" on how to merge your added features with the original feature vector and use your trained SVM model to make a prediction (output the predicted label and the original label). You only need to find one solution for each sample.

Tips

- Think about which features are benign-prone or malicious-prone. You may use SVM to make an explanation on the feature weights.

What to submit

- `mp4_model.py` from previous checkpoint
- `data/all_feature_names_5000.json`
- `data/added-features-5E06B7.txt`
- `data/added-features-49875A.txt`
- `mp4_SVM_models/svmf5000.p` that will be saved from checkpoint 2 automatically

- `verify_adv.py` with code comments why these features are helpful.

  You would not need to delete any files or submit selected files to the autograder, the above is a minimum checklist the autograde requires to grade the checkpoint. You can submit the repository as it is after making sure these files are present

Submission Instructions

Checklist

Place the following files in your mp4 directory.

- AndroidManifest_49875A.xml
- AndroidManifest_5E06B7.xml
- `main.py`
- `mp4_model.py` with code comments explaining your thought.
- `mp4_SVM_models/svmf5000.p` that will be saved from checkpoint 2 automatically
- `data/all_feature_names_5000.json`
- `data/added-features-5E06B7.txt`
- `data/added-features-49875A.txt`
- `verify_adv.py` with code comments why these features are helpful.
- Coursera Questions in the MP4 Assignment

Auto-grader

This semester we use auto-grader to eliminate any misalignment between student's working and grading.

Please use Python version 3.7 for checkpoint 2

To use the auto-grader please do as follows:

1: Upload your files into your git repo as mentioned above

2: Send HTTP POST request with your netid

We provide 2 methods, pick one as your preference. We recommend command line because it's more reliable.

Method1: Command Line

```
curl -X POST -H "Content-Type:  application/json" -d @submission.json
http://128.174.136.25:8081/grade_cp1
```

`submission.json` is a json file in your local computer, with only one key-value pair "netid":"YOUR_NETID"

Replace "YOUR_NETID" with your own netid, such as "netid":"aj3".
It's fine to change "submission.json" into other names, just remember to ensure it is a json file and you are doing the same change to the command line.
Remember, it's HTTP request rather than HTTPS.

Method2: Web Tool

If you don't want to use the above command line (sometimes more nasty), you can also use web tool such as Postman to send HTTP POST request to http://128.174.136.25:8081/grade_cp1 with a json body carring a key-value pair, "netid":"YOUR_NETID"

Replace "YOUR_NETID" with your own netid, such as "netid":"aj3".
Remember, it's an HTTP request rather than HTTPS.

3: Get Feedback

The feedback will be a web response to your HTTP request. The format is as follows (if no error message):

```
{
  "error": "",
  "result": 20
}
```

Any error will be put into "error" field such as Python error trace, wrong folder structure, git error or wrong web request.
Your grading result with other metadata will be reflected on the result field Remember, your grade and grading report will be automatically recorded on our server.

If you see **"curl: (6) Could not resolve host: application"**, don't worry, just wait for the result.

4.2.5 Sample responses for checkpoints

1: Checkpoint 1 (cp1)
On making a HTTP request to http://128.174.136.25:8081/grade_cp1 you will receive the following response:

```
{
  "error": "",
  "result": 20
}
```

Any error will be put into "error" field such as Python error trace, wrong folder structure, git error or wrong web request.
Your grading result and grade for this checkpoint will be put into the "result" field. The result will contain the score, in this case will either be 0/20 or 20/20

2: Checkpoint 2 - Evaluate SVM model with feature selection (cp2_1)
On making a HTTP request with the necessary files mentioned in 4.2.3 to http://128.174.136.25:8081/grade_cp2_1, you will receive the following response:

```
{
  "error": "",
  "result": {
    "f1_with_feature_selection": 0.9010458567980693,
    "f1_without_feature_selection": 0.8917609046849757,
    "totalScore": 35
  }
}
```

Any error will be put into "error" field such as Python error trace, wrong folder structure, git error or wrong web request.
Your grading result will contain three fields, two of which are the f1 value with and without feature selection which should be synonymous to your local environment. The final field is totalScore which can range from 0-35 based on the model performance

3: Checkpoint 2 - create adversarial samples (cp2_2)
On making a HTTP request to http://128.174.136.25:8081/grade_cp2_2 with the necessary files mentioned in 4.2.4, you will receive the following response:

```
{
  "error": "",
  "result": {
    "output": "{'5E06B7': 1, '49875A': 0}",
    "totalScore": 40
  }
}
```

Any error will be put into "error" field such as Python error trace, wrong folder structure, git error or wrong web request.
Your grading result will contain two fields, the output of the python file that is submitted (verify_adv.py) and the totalScore of this stage which can be 40/40 or 0/40.

Note: There might be a warning about LinearSVC version mismatch, please ignore as long as the "result" field shows the output

4.2.6 Timeout

You have 5 min for checkpoint1. Auto-grader will throw an error if timeout value is reached. Please make sure your code is time-efficient.

4.2.7 Abuse

You should only submit your own NetID when querying the auto-grader. Please do not abuse the auto-grader in any form. Abuse behaviors include but are not limited to using other students' NetIDs to submit a query and get their grading results. We will closely monitor the NetIDs sent by each IP address. Abusing the auto-grader is considered a form of violation of the student code of conduct, and the corresponding evidence will be gathered and reported.

4.2.8 Important Note

1: Your highest grade and grading report will be automatically recorded in our server.

2: You **must** send request to our auto-grader **at least once** before the deadline to get a grade. We don't grade your code manually, the only way is to use our auto-grader.

3: Try auto-grader as early as possible to avoid the heavy traffic before the deadline. **DON'T DO EVERYTHING IN THE LAST MINUTE!**

4: If the auto-grader is down for a while or you are encountering weird auto-grader behaviour, don't be panic. Contact us on Campuswire, we will fix it and give some extension if necessary.

5: If you see **"curl: (6) Could not resolve host: application"**, don't worry, just wait for the result to show up.

6: We have anti-cheating mechanisms, NEVER CHEAT OR HELP OTHERS CHEAT!

7: When auto-grader tests your code on dataset1, we will mask the location info for those who don't share their location even though it is provided in the dataset1! Be careful when you are testing in your local computer. Last semester lots of students overlooked this, causing misalignment between

local and online testing result.

8: If you are using web tool such as Postman, sometimes you will fail and see "server hang up". Keep trying more times or use command line instead.

9: Please use Python version 3.7. Don't use any Python modules except for the ones that are specified in the release.

10: Read section 4.2.7 carefully. Never abuse the auto-grader.