Introduction) Smartphones have become a staple in today's society with Android being the most popular smartphone operating system. However because of the open source nature of Android, it is more suscceptible to hosting malware applications on its app store. In fact, the Symantec's Internet Security Threat Report states that 1 in 5 Android application is malware. The way we will tackle this problem by detecting if an app is malware or not by analyzing not only just the API calls of Android applications but by analzying the relationships between the API calls of Android apps and represent them as a heterogeneous information network. We decided with this approach since malware can contain non suspicious API calls and group/use these non malicious API calls to execute malicious actions. The malware Android apps will be given to us while we will be collecting the benign Android app samples from apkpure.com. apkpure.com is a website that provides downloads of Android applications. We will be using apktool to decompile these Android apps downloaded from apkpure.com to smali code so we can analyze the code and search for API calls. We also want our sample of benign apps to contain an even amount of apps across all categories, as apps from different types of categories would use different type of API calls and thus have different relationships between API calls. Our sample containing a lot more of one type of app would be a shortcoming with the data. Another shortcoming would be if our sample contained any duplicate applications. To avoid the imbalance of categories being represented in our data, we will download the same amount of Android apps from each category. We will also randomize our app downloads within a category (ex: randomly download 5 apps in the productivity category). To avoid downloading duplicate applications, we will be keeping track of the names of the apps we downloaded in a list and checking that list every time we randomly select an app to download. By doing this we will prevent bias towards certain categories/apps. Our sample of benign apps should be the same size as our sample of malicious apps, not less to avoid underfitting and not more to avoid overfitting.

Our data will be originating from apkpure.com, a website that provides downloads of Android applications. Since apkpure.com only provides downloads of apps that are already free on appstores, there should be no legal issues with us getting our data from apk.pure. The schema I will be using organized in a way such that I have a directory called "Apps". "Apps" contains one subdirectory for every app category listed in apkpure (a directory for art, another directory for business, etc). The category subdirectories will contain one folder for every app that we downloaded and is part of that category (a folder lifestyle app will be in the lifestyle category) and inside the folder of that app will be its files, including its smali files which are the most relevant to our research. (Basically the schema is "Apps" -> categories -> specific app folder -> smali code). The game category in particular contains many different variations, such as trivia or simulation. Since I believe that these variations differ enough from each other (like trivia vs sports game) I will be treating each variation as their own category. This reason is also why I want to separate games by category in my schema. A productivity app and an entertainment app vastly differ from each other in terms of functionality and API calls. By separating them, I will be able to keep track how much of each category the apps in our data is comprised of.

Graph Definitions) The Hindroid paper describes four types of graphs that they used. All of the graphs involve API calls and one of the graphs also involves apps in it which is the first graph the paper talked about. The first one, labeled A is a graph where there is an edge for each app in the data set to the API calls the app contains. The second graph, labeled B is a graph where there is an edge between one API call to another only if they were in the same code block (a block of code in a smali file that starts with ".method" and ends with ".endmethod". The third one is labeled P and it is a graph where for every API call, there is an edge between it and another API call if they share the same package name. The last one is labeled I and it is a graph where for every API call, there is an edge between it and another API call if they both share the same invoke method. We will be using the Python package, NetworkX to make these graphs and Regex to extract API calls, code blocks and invoke calls.

EDA on Graphs) I have decided to only collect apps that belong in the productivity and entertainment category for my dataset. I chose those two categories because I feel like those most apps people have on their smartphones belong in those two categories (games are a really popular category too however games is a very broad and contains multiple sub categories). I've initially gathered about 211 apps for both categories each, making that 423 (I accidentally downloaded an extra entertainment app) apps in total and have extracted out 2153825 smali files from all the apps. I found that the most commonly used library for productivity apps in the dataset is is "Ljava/lang/StringBuilder;" while the most common library for entertainment is "Lcom/google/android/gms/internal/zzai;->". The most commonly used API call is "Ljava/lang/StringBuilder;->append;" for both entertainment and productivity apps. Initially, because of this, I would probably want to remove the top 100 or so most common API calls from the dataset if they are shared by both categories. However, I later learned that it was a better idea to remove the the more obscure APIs. The total number of API calls in this dataset is 1690119 while the average number of API calls made throughout the whole dataset is 169011.9.

Baseline Classification Model) The features I handmade for every app follows the out line, [library used the most, number of API calls (degree of app), most common API call] for every app. I chose those features because they were mentioned on the writeup and I assumed that the feautures would be very different for entertainment and productivity type apps. I trained 3 classifiers to see if any would do better than the other. The ones I chose were the RandomForestClassifier, The GradientBoostingClassifier, and the DecisionTreeClassifier. Overall they all had very similar accuracy scores of .79 for training and 0.67 for testing. In the end, I decided to only go for the DecisionTreeClassifier as it's training accuracy is 0.793 which is a very small and insignificant improvement compared to the others which were both 0.791.

The Hindroid Approach) The "Hin" in Hindroid refers to the very basis of their approach, the heterogenous information network. Basically, the heterogenous information network consists of different types of entities (here we have Apps and APIs) as well as different types of relationships (which were described in Graph Definitions). HIN gives structure and a way for these different entities and relationship to relate to each other which is important because then that way, we'll get information in how these variables are related. For example, by mulitplying the graph A with its transpose, A^T, we would be able to get the similarities between two apps from originally from the same graph.

Results) While I did not run my code on malware samples (yet), I tested Hindroid matrices with the same types of Apps as the ones from the dataset I did EDA on, except with 20 apps (for now). While the AA^T matrix had favorable results (testing accuracy score of .86) I cannot say the same for the rest of the matrices. Both my ABA^T and APBP^TA^T matrices have an accuracy score of .57 and my APA^T matrix has a pitiful accuracy score of .28. Thus, I've added new items onto my to-do list so I can go fix whatever mistakes I've made when constructing the P and B matrices. Or maybe it's not my contruction that's wrong and it's just that my sample size is too small. Regardless, I'll work to find out.

In [ ]: