Android App Similarity Visualization Proposal

Gabby Xiong
Univ. of British Columbia, Canada
gbxpeiyu@ece.ubc.ca

Michael Cao
Univ. of British Columbia, Canada
michael.cao@alumni.ubc.ca

ACM Reference Format:
Gabby Xiong and Michael Cao. 2020. Android App Similarity Visualization Proposal. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Mobile smart phones have become increasingly popular in the past decade. As a matter of fact, the number of smart phone users are expected to nearly double from 2016 to 2021 [17]. While smart phones bring many benefits to a user's life, their adoption has also greatly stimulated the growth of mobile malware. In particular, the Android market is a vulnerable target for malicious users due to its open sourced nature and large user market.

A number of approaches have recently emerged to support Android malware detection [1, 3, 8, 14]. Most of the proposed approaches rely on extracting application features and training a machine learning classifier to distinguish between the benign and malicious applications. Benign applications are easily collected and can be considered as an unlimited source. On the other hand, malware applications are often limited and difficult to find in practice. When training a classifier, practitioners often collect as many malware applications as possible and randomly sample from the unlimited benign pool. Training on such data is likely to create a highly separable classifier model.

A highly separable model clearly distinguishes between the benign and malware classes—often providing high performance results when detecting applications from the same distribution as training. The model often will learn highly distinguishing features that are associated with the benign or malware class. While a highly separable model provides good detection performance, it trades off reliability. Malware can easily exhibit the highly distinguishing benign features by adopting similar functionalities from the benign applications to disrupt the model's detection rate. In fact, recent studies have shown that simply injectinga small amount of benign features into malware can substantially reduce the malware detection rate of a highly separable model from 94% to 67% [4].

We hypothesize that the classifier trained using benign applications similar to the malware will reduce the number of highly distinguishing benign features and hence produce a less separable model. While less separable models are more resilient to the benign feature injection attack, we suspect the models' overall ability to distinguish between the benign and malware classes to decrease. Varying the similarity between benign and malware training applications could possibly tune the trade-off between performance and reliability of the model. To confirm this fact, we would like to build a visualization tool to effectively explore this hypothesis. In particular, we aim to visualize the characteristics of different training data sets to study the relationship between data selection, the reliability of a model, and the detection performance of a model.

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

2.1 Android Application Visualizations

Existing visualization tools for Android applications focus on Android malware. These works are divided into two categories: (1) visualizations that help security analysts locate malicious behaviors [9, 16, 21], and (2) visualizations of Android malware families [9].

Yan et al. [21] presented a visualization framework to assist malware analysts in statically locate the malicious code. The proposed tool first prunes an app's function call graph into sub graphs that contain suspicious behaviors, and visualize the graphs using a force-directed scheme. A set of interactions are provided for malware analysts to verify and further dissect the code block related to suspicious behaviors. Ganesh et al. also presented a visualization toolbox [16] designed to efficiently display program artifacts in manual malware analysis. They encoded the hierarchical class structure using containment marks, and allowed the user to abstract call sequences to a higher level by changing the visualization scope. To help researchers locate malicious behaviors while running the app, De Lorenzo et al. [6] presented a ML-based framework trained to classify blocks of execution traces related to malicious behaviors. After deployment, it monitors and visualizes the maliciousness of execution traces through a temporal bar chart.

Clustering Android malware into different families based on their malicious behaviors can also help malware analysts prevent attacks from malware variants. Gonzales et al. [9] applied various dimension reduction techniques to project high dimensional traffic data of malware applications to 2D, and visualized the results using scatter plots. By comparing the results from different projection functions, they identified differentiating traffic patterns among malware families.

To the best of our knowledge, only Rory et al. [5] have presented a tool to study similarities between benign and malware applications. They measured the distance among applications in the feature space and used a circular dendrogram to visualize the hierarchical relationships among applications. Malware analysts can easily infer the distance between applications by locating them on the dendrogram, and understand the classification results from classifiers trained on the same applications.

2.2 Visualizing Machine Learning Data

Over the years, an abundance of work has been proposed to incorporate visualization into the machine learning process. Most have focused on providing visualizations to help users explore correlations between features, or provide an interaction interface to keep users in the loop and improve the reliability of the model. Only a few tools [10, 12, 15, 20] have been proposed to help users investigate training data set characteristics, or examine relationships between the training data and resulting model properties [11].

Patel et al. presented a visualization framework [15] to understand properties of the training data. Their tool helps users identify noteworthy examples, by exploiting the classification results based on multiple trained model instances. The Profiler [12] system proposed by Kandel et al. automatically flags problematic data using data mining techniques. The authors present results of their flagging process by using coordinated summary visualizations that support interactions and links to assess detected anomalies. Wongsuphasawat et al. [20] presented a tool to help analysts in discovery by offering recommendations of potentially interesting visualizations on particular features. The tool also generates coordinated views given a user's partial specifications. In addition, Google launched the web interface FACETS [10] for users to visualize their high-dimensional machine learning data. It offers two visualizations: "Facets Overview" and "Facets Dive". "Facets Overview" produces summary statistics over each feature, and compares the distributions over training and testing data. "Facet Dive" provides an interactive interface for users to sort and examine individual applications in the data.

Hohman et al. [11] pointed out that understanding the data should be of equal importance as understanding the model. They describe that it is crucial to evaluate the quality of training data and monitor the performance of the model. They presented a visualization interface, "CHAMELEON", which integrated multiple coordinated views to help researchers compare the trained model over different data sets.

3 DATA AND TASK ABSTRACTION

In this section, we describe our task abstractions in the context of terminologies introduced in Visualization Design and Analysis.

3.1 Data

We collected malware applications based on snapshots provided by VirusTotal Academic [19]. Six snapshots were provided between the years 2016 to 2019, consisting of 10K applications. And we collected our benign applications from AndroZoo [2], a large repository which continuously scrapes applications from a variety of markets between a similar time range. All applications will be converted into feature vectors using Drebin [3], a popular Android machine learning malware detection tool. Drebin uses lightweight static analysis to extract eight types of features based on an application's bytecode and manifest file. The eight categories include: (1) requested permissions, (2) used permissions, (3) application components, (4) hardware components, (5) intent filters), (6) permissions guarded API calls, (7) suspicious API calls, and (8) network addresses. All feature values are binary and are represented as either 0 (i.e., feature absent from the application) or 1 (i.e., feature present

Table 1: What-Why-How-Analysis-Table

What: Data	Tabular, binary-valued attributes and sparse
What: Derived	Dimension reduced sample values (applying relative distance preserving DR). Pair-wise sample similarity (e.g. Euclidean distance, Cosine similarity, Jaccard Coefficients)
Why: Tasks	Explore training data sets, visualize characteristics of user selected subset of data sets. Evaluate the performance of the model (e.g. detection rate, robustness under attack). Track how model change when using different data sets
How: Encode	Scatter plots, Bar charts, Line charts, Table
How: Facet	Multiform, overview-detail
How: Reduce	Filtering on the samples, filtering on the features
How: Embed	Pop-up window with details (e.g. sample detail)
How: Manipulate	Zoom, pan, select, sort
Scale	Approx. 60,000 items (10,000 malware samples and 50,000+ benign samples)

in the application). From our previous experiences of using Drebin, a training data set of 10K applications leads to 108K features. We plan to apply feature selection to reduce the number of features.

3.2 Tasks

At a high level, we would like to provide a tool that helps researchers in analyzing the relationship between the characteristics of the training data and the properties of the resulting model. In detail, the tool can assist the researcher in achieving the following tasks:

- Visualize the overall training data distribution in 2D and visually inspect possible clusters.
- Evaluate the similarity among all applications in training data based on selected feature of interest and locate similar or distinguishable applications. This could help the researcher to tune the similarity among benign and malware applications in training dataset.
- Visualize feature distributions over a selected set of benign and malware applications. This could assist the researcher in identifying key features contribute to similarities or dissimilarities.
- Construct a training data set from the total application pool to train the model for further analysis.

After the researcher selects a set of training applications, our system will train an SVM model and test it on a set of malware and benign testing applications. Based on the results, the researcher can investigate the reasons behind the model's performance and reliability by:

- Visualizing the overall training and testing data distribution in 2D, visually inspect the distribution of the testing applications compare to that of training applications.
- Identifying the misclassified applications, and locate similar training applications to reason about model prediction results using their similarities.
- Identifying the effort (i.e. number of benign features) required for malicious applications to evade model detection.
- Analysing the weight assigned by the trained model to each feature to explain prediction on samples.

4 SOLUTION

In this section, we describe our plans to implement the visualization system that can assist researchers perform the previously defined tasks.

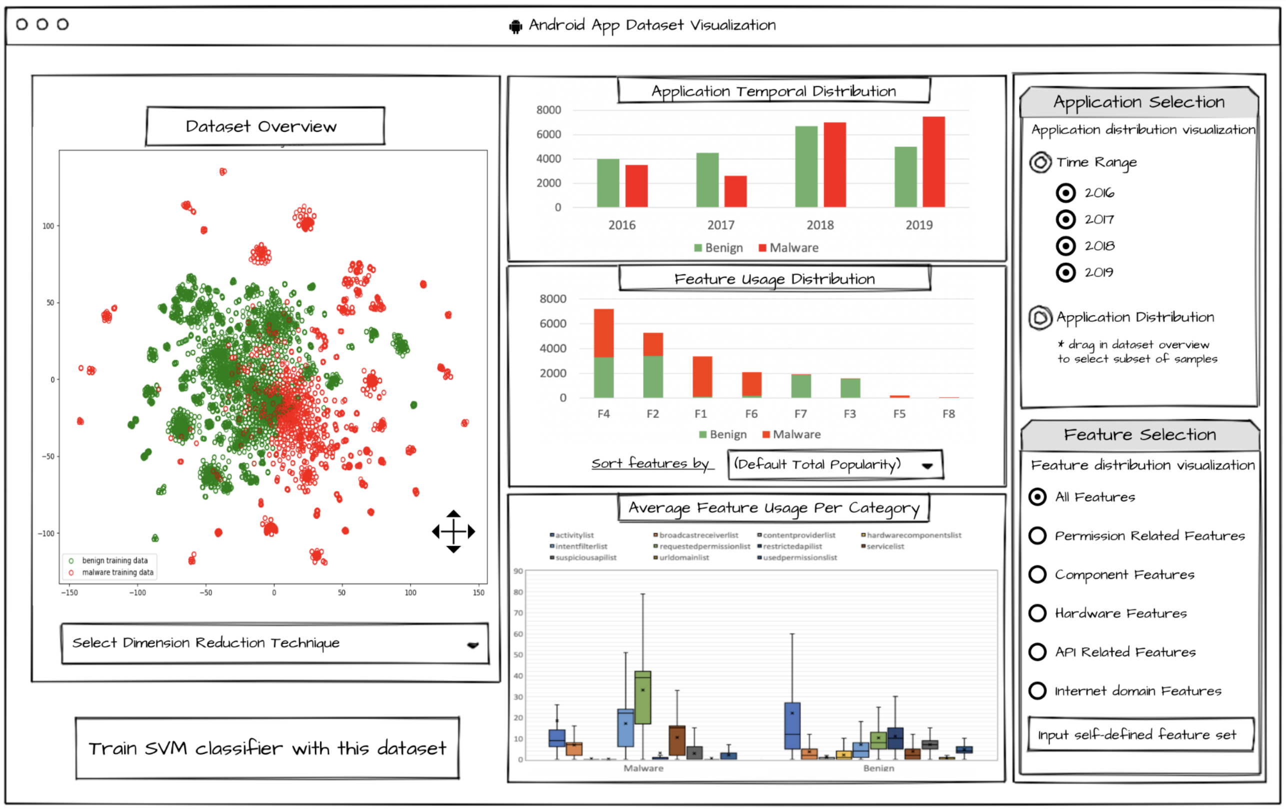


Figure 1: Data Set Visualization Page

Task: Visualize the training data set distribution in 2D.

In order to create an overview of the applications in 2D, we plan to perform popular dimension reduction techniques (e.g. t-SNE [13], ISOMAP [18]) on the application features provided by the researcher. To visualize the result from dimension reduction, we considered using a simple scatter plot (shown in Figure 1, "Dataset Overview"). However, we noticed that scatter plots have issues with scalability—overlapped applications could obscure the density information. To reduce overlap, we used "jittering" — by adding a small fraction of noise to the each sample's location such that they are less likely to be on same spot. We also considered an alternative visualization where we divide the 2D space into square blocks, and use pie charts to display the benign and malware distribution over each block. The size of the each circle encodes the number of applications within each block. This approach preserves the density information, however breaking the continuous space into discrete blocks can potentially distort continuous clusters. From our initial testing with 30,000 samples, we found jittering can already reduce overlap and produced little disturbance on the sample distribution, so we plan to continuing using scatter plot.

Task: Identify similar and dissimilar applications.

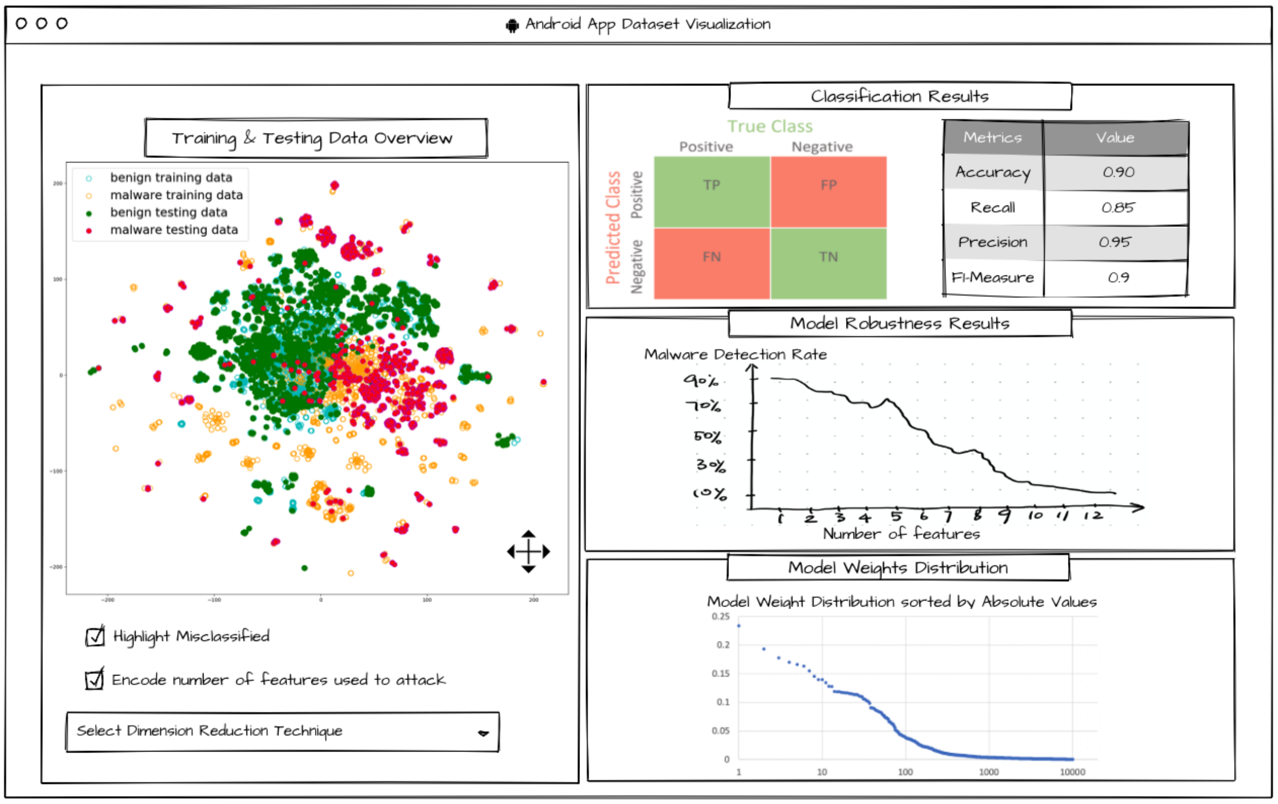
As dimension reduction techniques project samples from high dimensional space to 2D while preserving the overall representation of the data set, the distances among samples in 2D scatter plot reflect the similarity among samples. The researcher can identify similar samples by locating clusters in the 2D plot.

Task: View summary statistics of the feature usage over a set of applications.

Visualizing the feature distribution over a set of applications is useful for exploring the characteristics of a data set. As DREBIN features only contain binary values, we can aggregate the feature usage information over a particular set of features, and visualize them using a single bar chart; the x-axis corresponds to features, and y-axis corresponds to the feature usage among benign and malware applications (shown in Figure 1, "Feature Usage Distribution"). To better uncover the feature usage trend, we also offer the flexibility for users to choose how to sort the features on the x-axis. Other than the trend in overall feature usage, there could also be interesting patterns in how different categories of features are used among benign and malware applications (e.g. malicious applications may tend to request more permissions to access different resources on device, or use more suspicious API to perform malicious tasks) Hence, we will also designate a view on the average usage of different feature categories over benign and malware applications (shown in Figure 1, "Average Feature Usage Per Category"). We plan to use either a bar chart or line trend chart with error bars, to show the average feature usage distribution for each category among benign and malware applications separately.

Task: Perform analysis on subset of data.

To see how characteristics vary over different data sets, we offer the user options to filter applications by time (e.g. select the temporal range) or to select the applications directly by drawing over



Metrics	Value
Accuracy	0.90
Recall	0.85
Precision	0.95
F1-Measure	0.9

Figure 2: Model Performance Visualization Page

the application distribution view (shown in Figure 1, "Application Selection"). Once the user filters or draws a selection of applications, the feature distribution views will be updated accordingly.

Task: Perform analysis using different features.

If the researcher wants to ignore some features, they can use the feature selection panel to restrict the features to categories of interest (shown in Figure 1, "Feature Selection"). Once the feature category selection is updated, the visualization views will also be updated to reflect the changes in similarity among applications.

Task: Evaluate model performance.

After the researcher analyzes and selects a desired training data set (the selected applications will be highlighted in the "Data set Overview" in Figure 1), the system will train an support vector machine (SVM) model and evaluate the model performance on a pre-defined testing data set. The performance result shown to the user will include: (1) a confusion matrix (i.e. True Positive, True Negative, False Positive, and False Negative), (2) accuracy, precision, recall, and f1-measure on malware detection (shown in Figure 2, "Classification Results"), (3) the malware detection rate degradation under the injection of benign features (shown in Figure 2, "Classification Results").

Task: Understand model performance.

As the first step to help the user interpret the prediction results from the model, we will visualize the application distributions over both training and testing data. Similar to the procedure used to visualize training data distributions, the same dimension reduction technique will be applied to generate the application distribution in 2D using both training and testing applications. To differentiate training and testing data, we will use different colors to encode training benign, training malware, testing benign, and testing malware (shown in 2, "Training & Testing Data Overview"). Based on the performance results of the model, we can identify and highlight misclassified applications on the data overview. Another interesting piece of information to include in the view is the evasiveness of different malware applications (i.e. the number of features needed to evade detections). We plan to categorize the malware applications into buckets based on the number of features injected to evade detection, and use color gradients to encode the buckets (i.e., darker red represents the applications which require more injected features to evade detection, and lighter red requires less features to evade detection). This is shown in Figure 2 under "Training & Testing Data Overview". By including such information on the application distribution view, we hope to use the samples' surrounding training applications to explain the model's predictions. To understand model robustness, we will visualize the model weight distribution (shown in Figure 2, "Model Weights Distribution"). It is suggested by Demontis et al. [7] that a model is more robust against attacks if its feature weights are more evenly distributed. Intuitively, this is because there are less highly distinguishing benign features. We plan to sort the model's feature weights from largest to smallest, and visualize the trend using a scatter plot to see if all the feature

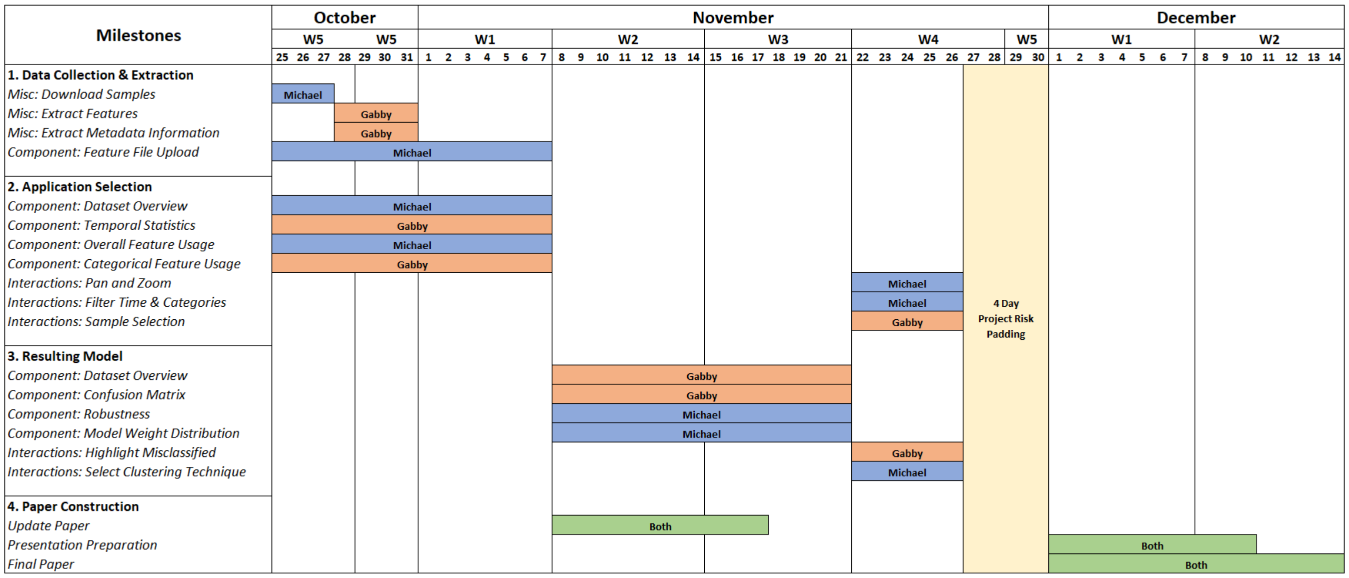


Figure 3: Work Breakdown

weight values are bounded. If the feature weights show a slow decreasing trend over the set of features, this would indicate a more robust model.

The design decisions we described above are also summarized in Table 1 by following the What-Why-How analysis approach introduced in Visualization Design and Analysis.

5 IMPLEMENTATION

We started to develop the visualization system as a web application. Using Django and Python as BackEnd to perform computational demanding tasks, and the React framework for FrontEnd with d3 as visualization library.

6 MILESTONES

We partition our milestones based on the three screens required to aid researchers in selecting applications, including: (1) Data Collection & Extraction, (2) Application Selection, and (3) Resulting Model. In addition to the three milestones, we include a fourth to describe tasks related to constructing the project paper and presentation. Figure 3 shows a gantt chart describing time estimations of tasks required to implement the four milestones. Work done by Michael and Gabby are associated with blue and orange bars, respectively. We plan to implement our milestones in two week software sprints, where the result of every sprint produces a working visualization program.

In the first sprint (October W4/W5 and November W1) we plan to collect all necessary data and build a minimalistic app that displays the application feature data on the screen. The first week (October W4/W5) will be used to collect data used for the study. We estimate 3 days for Michael to download all benign and malicious applications, as we plan to collect 60,000 applications overall and noticed our servers were previously capable of downloading 20,000 applications per day from available online sources. After the downloading of applications is complete, Gabby will write scripts to collect features and related metadata information necessary for visualization purposes. We estimate the collection of this information to take four days, as substantial computational power is needed to extract this information from the samples. In parallel, we plan to start setting up the baseline web components used to retrieve and visualize the application feature data. We further estimate that setting up the web components will take a week, and integrating with d3 to take another week. The first sprint will result in a web application that can take application feature data as input and display the data on the web application.

In the second sprint (November W2/W3) we plan to develop components used to visualize the model, produced from selecting a set of samples. These components include a visualization of training and testing data, a confusion matrix, a model weight distribution, and a measurement of the robustness to the benign feature injection attack. We estimate that it will take two weeks to implement the new components and integrate it with the existing product from sprint one. Note that we have not implemented the interactions necessary to select applications, and plan to mock application selection until the next sprint. In parallel, we plan to update the project paper with details related to visualizing the set of malware applications, benign applications, and the resulting model. The second sprint will result in a web application which displays visualizations of the feature data and model results.

In the third sprint (November W4/W5) we plan to implement user interactions with the system. This includes panning, zooming, time and category feature filtering, sample selection, highlighting misclassifications, and selecting cluster techniques for visualization purposes. We estimate a total of five days to implement all user interactions. The third sprint results in a fully functional visualization system, which allows users to input application feature data, filter and select data for the study, and view the resulting model in terms of weights, performance, and reliability. We additionally add four days of padding to account for risks in implementation. Finally, the last two weeks are used to prepare for the final paper and presentation.

7 DISCUSSION

Fine to leave empty in proposal.

8 FUTURE WORK

Fine to leave empty in proposal.

9 CONCLUSIONS

Fine to leave empty in proposal.

REFERENCES

[1] Kevin Allix, Tegawendé F. Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 2016. Empirical Assessment of Machine Learning-Based Malware Detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

[2] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proc. of Working Conference on Mining Software Repositories (MSR)*. 14–15.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. of Network and Distributed System Security Symposium (NDSS)*. 23–26.

[4] Michael Cao, Sahar Badihi, Khaled Ahmed, Peiyu Xiong, and Julia Rubin. 2020. On Benign Features in Malware Detection. In *Proc. of International Conference on Automated Software Engineering (ASE)*. ACM, 1234–1239.

[5] Rory Coulter, Lei Pan, Jun Zhang, and Yang Xiang. 2019. A visualization-based analysis on classifying Android malware. In *International Conference on Machine Learning for Cyber Security*. Springer, 304–319.

[6] Andrea De Lorenzo, Fabio Martinelli, Eric Medvet, Francesco Mercaldo, and Antonella Santone. 2020. Visualizing the outcome of dynamic analysis of Android malware with VizMal. *Journal of Information Security and Applications* 50 (2020), 102423.

[7] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, Machine Learning Can Be More Secure! a Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)* 16, 4 (2017), 711–724.

[8] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 3, Article 11 (2018).

[9] Alejandro González, Álvaro Herrero, and Emilio Corchado. 2016. Neural visualization of android malware families. In *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16*. Springer, 574–583.

[10] Google. [n.d.]. Facets - Know you data. https://pair-code.github.io/facets/.

[11] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[12] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. 547–554.

[13] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[14] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of Network and Distributed System Security Symposium (NDSS)*. 1–12.

[15] Kayur Patel, Steven M Drucker, James Fogarty, Ashish Kapoor, and Desney S Tan. 2011. Using multiple models to understand data. In *Twenty-Second International Joint Conference on Artificial Intelligence*. Citeseer.

[16] Ganesh Ram Santhanam, Benjamin Holland, Suresh Kothari, and Jon Mathews. 2017. Interactive visualization toolbox to detect sophisticated android malware. In *2017 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 1–8.

[17] Statista. [n.d.]. Number of smartphone users worldwide 2014-2020. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[18] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[19] VirusTotal. 2020. VirusTotal. https://www.virustotal.com/home. last accessed August 2020.

[20] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2648–2659.

[21] Yan Zhang, Gui Peng, Lu Yang, Yazhe Wang, Minghui Tian, Jianxing Hu, Liming Wang, and Chen Song. 2017. Visual Analysis of Android Malware Behavior Profile Based on PMCGdroid: A Pruned Lightweight APP Call Graph. In *International Conference on Security and Privacy in Communication Systems*. Springer, 449–468.