Predicting Android Application Security and Privacy Risk With Static Code Metrics

Akond Rahman*, Priysha Pradhan*, Asif Partho$^{\phi}$, and Laurie Williams*
North Carolina State University, Raleigh, NC, USA
Nested Apps, Dhaka, Bangladesh
*{aarahman, ppradha3, lawilli3}@ncsu.edu, $^{\phi}$asif@nestedapps.com

***Abstract*—Android applications pose security and privacy risks for end-users. These risks are often quantified by performing dynamic analysis and permission analysis of the Android applications after release. Prediction of security and privacy risks associated with Android applications at early stages of application development, e.g. when the developer (s) are writing the code of the application, might help Android application developers in releasing applications to end-users that have less security and privacy risk. *The goal of this paper is to aid Android application developers in assessing the security and privacy risk associated with Android applications by using static code metrics as predictors*. In our paper, we consider security and privacy risk of Android application as how susceptible the application is to leaking private information of end-users and to releasing vulnerabilities. We investigate how effectively static code metrics that are extracted from the source code of Android applications, can be used to predict security and privacy risk of Android applications. We collected 21 static code metrics of 1,407 Android applications, and use the collected static code metrics to predict security and privacy risk of the applications. As the oracle of security and privacy risk, we used Androrisk, a tool that quantifies the amount of security and privacy risk of an Android application using analysis of Android permissions and dynamic analysis. To accomplish our goal, we used statistical learners such as, radial-based support vector machine (r-SVM). For r-SVM, we observe a precision of 0.83. Findings from our paper suggest that with proper selection of static code metrics, r-SVM can be used effectively to predict security and privacy risk of Android applications.**

Keywords-Android application; code metrics; prediction; security and privacy risk

I. INTRODUCTION

Gartner has reported that by 2017, mobile users would provide personalized data to more than 100 apps and services every day [1]. Considering the fact that Android applications have the potential of posing security and privacy risks for end-users [2], such wide-scale transmission of end-user data to third-party services, can lead to major data breaches, as well as security threats. For example, according to a recent news report, 75% of the top free Android applications were susceptible to data breaches [3]. These data breaches include leak of user names, leak of passwords, and leak of MAC addresses [4]. Android developers may create Android applications that re-use third party libraries that contain vulnerabilities and infected binaries, or use permission settings that expose private health and monetary information of the end-user [1], [5].

Prior studies have showed the effectiveness of using static code metrics in predicting failure prone components [6]. In our work, we investigate whether metrics that are available from static analysis tools can be used to predict security and privacy risk of Android applications. Such investigation might help Android developers to identify risky Android applications at the development stage, and prior to release of the Android applications.

The goal of this paper is to aid Android application developers in assessing the security and privacy risk associated with Android applications by using static code metrics as predictors.

In our paper, security and privacy risk of Android application refers to how susceptible the application is to leaking private information of end-users and to releasing vulnerabilities. We focus our effort in answering the following research question to achieve this goal:

RQ: How effectively can statistical learners be used to predict different levels of security and privacy risk using static code metrics?

In our research study, we analyzed a collection of 1,407 Android applications with security and privacy risk scores that are computed using Androrisk [7], a tool that assigns a risk score to an Android application using analysis of Android permissions and dynamic analysis. The Androrisk risk score approximates the amount of security and privacy risk for the Android application based on Android activities and permissions used by the application and on dynamic analysis of the application [7]. We applied four statistical learners namely, classification and regression trees (CART), k Nearest Neighbor (kNN), Support Vector Machine (SVM), and Random Forest (RF) to predict the level of security and privacy risk.

We summarize the contribution of this paper as following:

- An evaluation of how static code metrics can be used to predict the security and privacy risk with the help of statistical learners.

II. RELATED WORK

Our paper is closely related to prior academic studies that have investigated detection and prediction of Android malware, and effects of source code properties of Android applications on performance and market success.

Existing work had focused on detecting and categorizing malicious Android applications using Android permission files [8], dynamic analysis [9], and applying statistical and machine learning techniques [10]. Yang et al. [9] introduced DroidMiner that tracks Framework API calls executed by

Android applications and used that information to detect and classify malicious Android applications. Gorla et al. [8] used differences in the description topics and permission levels of Android applications to detect Android malware applications in an unsupervised fashion. Peng et al. [10] stated the in-effectiveness of permission-based alerting to inform end-users about the privacy risks of Android applications and proposed multiple variants of Naïve Bayes models that categorizes market place Android applications as being highly or less risky.

Prior research studies have also investigated the negative impacts of low quality source code of Android applications. Corral and Fronza [11] investigated how market success is dependent on source code quality of Android applications. Mannan et al. [12], investigated code smells within Android applications, and identified differences between the smell distribution of Android applications, and Java desktop applications. These research studies provide empirical evidence that source code of Android applications is correlated with Android application performance and quality.

We use evidence from the above-mentioned papers as inspiration to conduct our research study. Our research study focuses on using source code to predict security and privacy risk of Android applications.

III. METHODOLOGY

In this section, we describe the major steps of our research study as summarized in Figure 1.

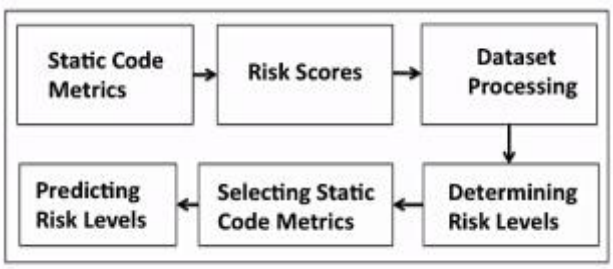

Figure 1: Major steps of the methodology.

A. Static Code Metrics

We based our selection of static code metrics on prior research in the domain of software engineering. From prior research on defect prediction of software system, we observe static code metrics that can be derived from object-oriented programming (OOP) languages, such as lines of code, number of classes, and number of functions, can be used to predict software defects [13]. Prior research has also provided evidence that duplicated pieces of code can contribute to defects [14]. Furthermore, practitioners perceive bad coding practices to hinder team productivity because bad coding practices, such as creating multiple instances of variables and objects, not closing network sockets, and use of duplicate string literals, might lead to software defects as well as to increased amount of rework [15], [16]. From evidence of prior research, we assume that extracting bad coding practices, duplications, and OOP attributes, in forms of metrics might help in predicting security and privacy risk of Android applications. We take all these empirical findings into account and hypothesize that the static code metrics that are used for defect prediction might also be used for predicting security and privacy risk of Android applications. In our research study, we use 21 static code metrics that are extracted using SonarQube [16]. SonarQube is a tool that applies static analysis on source code files, such as Java files, and produce metrics. As Android applications are built using a Java-based Android SDK, we used SonarQube to extract the static code metrics from Java source code files of the Android applications. We present the 21 static code metrics in Table I. The definitions of the code metrics are available online[1].

Table I: Static Code Metrics

Category	Metrics
Bad Coding Practice	Blocker practices, Critical practices, Major practices, Minor practices, Total bad coding practices
Duplication	Duplicated blocks, Duplicated files, Duplicated lines
Object-oriented	Class complexity, Comment lines, Complexity, Density of comment lines, Files, File complexity, Function complexity, Lines, Lines of code, Methods, Number of classes, Percentage of comments, Percentage of duplicated lines

B. Risk Scores

We selected Androrisk for our security and privacy risk oracle because it has been successfully evaluated for both open source and commercial Android applications, and is widely used for research in the security and software engineering domain [17], [18], [19]. Androrisk considers 21 risk categories of how an application can pose security and privacy risk for end-users. Using the mapping of activities to the 21 categories, Androrisk computes the security and privacy risk of the application using fuzzy logic [20]. For each Android application, Androrisk provides a security and privacy score between 0 and 100. We refer to security and privacy score of an Android application as ‘risk score’ for the rest of this paper.

C. Dataset Processing

To conduct our research study, we used an existing dataset provided by Krutz et al. [17]. This dataset contained Androrisk risk scores of 4,416 Android applications. As datasets are susceptible to noise, Tan et al. [21] stated the importance of data cleaning before applying data analysis techniques on the dataset of interest. From our preliminary exploration, we observe that the original dataset in its raw

[1] http://tiny.cc/android-metric

format included applications that have no risk scores. We only included applications for which the risk score is available. Upon removing the applications with no risk scores, we obtain a pre-processed dataset, which we refer to as the 'formatted dataset' for the rest of this paper.

D. Determining Risk Levels

The risk scores of the Android applications are numeric, and these risk scores might be more interpretable to practitioners presented with respect to levels of risk. For example, two applications with risk scores of 70 and 73 might be more interpretable for assessment if the two applications are marked as 'high risk'. To achieve better interpretation of risk for Android applications, we apply k-means clustering [21] on risk scores of Android applications. We specify two inputs for the k-means clustering technique: the count of clusters needed to create, and the data that will be used to create the clusters. The risk scores of the Android applications included in the formatted dataset are used as the data that will be used to create the clusters. We determined the count of clusters using a cluster evaluation technique called Dunn index [22]. We describe the process of determining the count of clusters using Dunn index as following:

Using the risk scores of Android applications included in the formatted dataset, first we created {2, 5, 8, …, 100} clusters at j={1, 2, 3, …, 33}th cluster creation step. Then, for each of the j cluster creation steps we computed the Dunn index. Next, we determined the jth cluster creation step that yielded the highest Dunn index; we refer to this jth cluster creation step as 'j_max'. The count of clusters created at j_maxth step is determined as the count of clusters that will be created using k-means clustering.

In our research study we used the 'clvalid' package available in R v. 3.1.2, to implement Dunn index, and 'Scikit Learn' [23] to apply k-means clustering. After applying k-means clustering, all the Android applications will be assigned to a unique cluster. Each of the created clusters is referred to as 'level of risk' or 'risk level' throughout the paper.

E. Selecting Code Metrics

All of the 21 identified static code metrics might not contribute to prediction of risk levels, and we need to determine which of the 21 static code metrics that can be used to predict risk levels. We used principal component analysis (PCA) [21] to identify the static code metrics that contribute to security and privacy risk of Android applications and do not confound each other. We selected the principal components that account for greater than 95% of the total variance.

F. Predicting Risk Levels

Using the selected set of static code metrics from our analysis of Section III-E, we use statistical learners to predict risk levels. We first briefly describe the statistical learners that we use in the study, and then we describe on how we apply the statistical learners to build prediction models.

Prediction performance can vary significantly from one statistical learner to another [24] [25], and researchers have recommended the use of multiple statistical learners for building prediction models [25]. We select four statistical learners that belong to four categories: Radial-Basis Support Vector Machine (r-SVM) [26], Random Forests (RF) [27], k Nearest Neighbor (kNN) [28], Classification and Regression Trees (CART) [29].

We evaluate the performance of the created prediction models by applying 10-fold cross validation technique. We repeated the 10-fold cross validation 10 times to assess the statistical learner's prediction stability. The performance of prediction models are tested by using nine of the 10 folds as training data, and the remaining fold as test data [30]. We created one prediction model for each of the four statistical learners with their default parameter provided in Scikit Learn [23]. For each of the four statistical learners we use the following two metrics to determine the prediction accuracy:

- Precision: Precision is the fraction of the predicted levels by the prediction model of risk levels that are relevant [31].
- Recall: Recall is the fraction of the relevant levels of risk levels that are predicted by the prediction model [31].

IV. Empirical Findings

We use this section to present our findings that answer our research question of interest.

A. Dataset Processing

The formatted dataset included 1,407 Android applications. We eliminated 3,009 of the 4,416 Android applications because they did not have a risk score in the original dataset.

B. Determining Risk Levels

In Section III-D, we have described how we used Dunn index to determine the count of clusters to which the 1,407 Android applications will be assigned. According to Figure 2, Dunn index is highest when five clusters are created, and therefore we determine the count of clusters to be five for our analysis. Taking the risk scores of the 1,407 Android applications and the count of clusters determined by Dunn index as input, we create five clusters using k-means clustering. We assign labels to these clusters: very low (VL), low (L), medium (M), high (H), and very high (VH). The centroids of the five clusters namely VL, L, M, H, and VH are respectively, 0.00, 16.43, 30.00, 44.29, and 50.96. These five clusters correspond to five levels of risk. The count of Android applications that belong to levels VL, L, M, H, and VH are respectively, 767, 590, 10, 12, and 28.

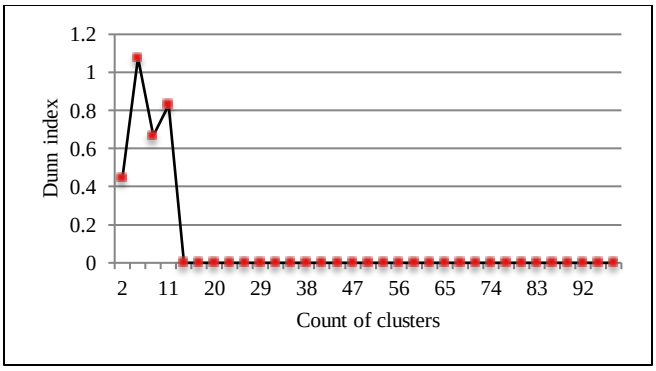

Figure 2: Use of Dunn index to find optimal count of clusters. The Dunn Index is highest for five clusters.

C. Selecting Appropriate Code Metrics

We have applied PCA, as described in Section III-E, to determine the static code metrics that can be used for security and privacy risk prediction. We observe one principal component accounts for 98.99% of the total variance. Therefore, we will use one principal component for predicting security and privacy risk. The top five static code metrics included in the principal component are: lines, lines of code, complexity, total bad coding practices, and major practices, with weights respectively of, 0.83, 0.51, 0.12, 0.11, and 0.08.

D. Predicting Risk Levels

We report our findings related to predicting security and privacy risk levels in Figures 3, and 4. In Figures 3, and 4 the y-axis presents the measures of prediction performance namely, precision, and recall, whereas, the x-axis corresponds to the five levels of security and privacy risk namely, very high (VH), high (H), medium (M), low (L), and very low (VL).

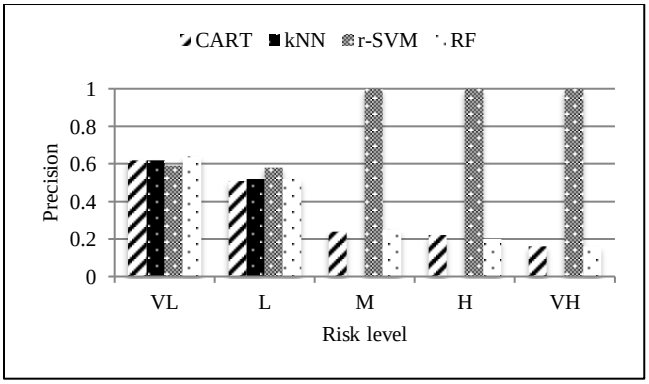

Figure 3: Precision scores of the four statistical learners. Overall r-SVM performs better for all five levels of security and privacy risk.

For example, according to Figures 3, and 4, the precision, and recall, for CART was 0.62, and 0.61, respectively considering level VL. Findings from Figures 3, and 4, indicate that one single learner is not sufficient enough that have high prediction performance with respect to precision, and recall for all the five levels of security and privacy risk.

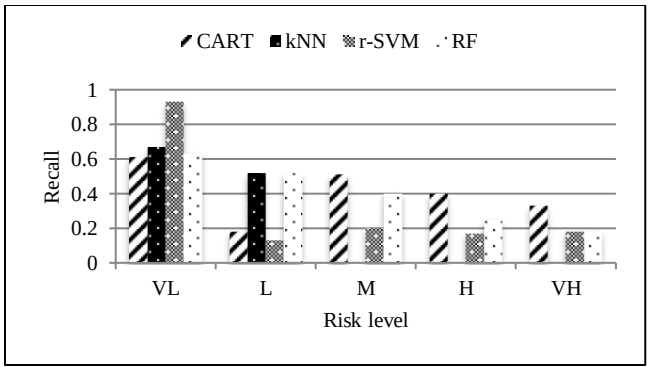

Figure 4: Recall scores of the four statistical learners.

With respect to precision, r-SVM outperforms the other three statistical learners, as the average precision across the five levels for r-SVM is 0.83, which is the highest amongst the four statistical learners. With respect to predicting security and privacy risk levels 'H' and 'VH', r-SVM also outperforms the other three statistical learners, as r-SVM has a precision of 1.0 for levels 'H' and 'VH'.

Answer to RQ: *r-SVM can be used to build a prediction model for predicting security and privacy risk that takes static code metrics as input. We observe an average precision of 0.83 considering five levels of security and privacy risk for r-SVM.*

V. LIMITATIONS

We use this section to describe the limitations of the study:

Dataset: In our study, we used a dataset that contained 4,416 applications, and we have used data pre-processing techniques to get a formatted dataset of 1,407 applications. We acknowledge that more data can help to generalize our findings.

Use of static code metrics: Our use and analysis of static code metrics for predicting multiple levels of risk is limited to the 21 static code metrics. We observe the importance of considering other static code metrics, as well as other types of metrics such as process metrics to predict multiple levels of security and privacy risk for Android applications, as future work.

VI. CONCLUSION

In this paper, we have evaluated how static code metrics such as number of lines, functional complexity, and McCabe's complexity can be used to predict security and privacy risk for Android applications. We have evaluated four statistical learners and have observed that r-SVM can effectively predict risk levels. We conclude that with proper use of statistical learners, static code metrics might be used effectively to predict multiple levels of risk for Android applications. Researchers can take our findings into account for future research in the domain of Android security and privacy.

REFERENCES

[1] Gartner, "Gartner Says by 2017, Mobile Users Will Provide Personalized Data Streams to More Than 100 Apps and Services Every Day," Gartner Says by 2017, Mobile Users Will Provide Personalized Data Streams to More Than 100 Apps and Services Every Day, 22-Jan-2014. .

[2] A. Atzeni, T. Su, M. Baltatu, R. D'Alessandro, and G. Pessiva, "How Dangerous is Your Android App?: An Evaluation Methodology," in Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, ICST, Brussels, Belgium, Belgium, 2014, pp. 130–139.

[3] N. CBS, "Beware downloading some apps or risk 'being spied on,'" Beware downloading some apps or risk "being spied on," 24-Feb-2016. .

[4] NowSecure, "2016 NowSecure Mobile Security Report," 2016 NowSecure Mobile Security Report, 2016. .

[5] J. Zang, K. Dummit, J. Graves, P. Lisker, and L. Sweeney, "Who Knows What About Me? A Survey of Behind the Scenes Personal Data Sharing to Third Parties by Mobile Apps.," Who Knows What About Me? A Survey of Behind the Scenes Personal Data Sharing to Third Parties by Mobile Apps., 30-Oct-2015. .

[6] N. Nagappan and T. Ball, "Static Analysis Tools As Early Indicators of Pre-release Defect Density," in Proceedings of the 27th International Conference on Software Engineering, New York, NY, USA, 2005, pp. 580–586.

[7] K. Dunham, S. Hartman, J. Morales, M. Quintans, and T. Strazzere, Android Malware and Analysis. Boca Raton,FL: Taylor and Francis, 2014.

[8] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking App Behavior Against App Descriptions," in Proceedings of the 36th International Conference on Software Engineering, New York, NY, USA, 2014, pp. 1025–1035.

[9] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Computer Security - ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I," M. Kuty\lowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 163–182.

[10] H. Peng et al., "Using Probabilistic Generative Models for Ranking Risks of Android Apps," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, New York, NY, USA, 2012, pp. 241–252.

[11] L. Corral and I. Fronza, "Better Code for Better Apps: A Study on Source Code Quality and Market Success of Android Applications," in Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on, 2015, pp. 22–32.

[12] U. A. Mannan, I. Ahmed, R. A. M. Almurshed, D. Dig, and C. Jensen, "Understanding Code Smells in Android Applications," in Proceedings of the International Conference on Mobile Software Engineering and Systems, New York, NY, USA, 2016, pp. 225–234.

[13] N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," in Proceedings of the 28th International Conference on Software Engineering, New York, NY, USA, 2006, pp. 452–461.

[14] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do Code Clones Matter?," in Proceedings of the 31st International Conference on Software Engineering, Washington, DC, USA, 2009, pp. 485–495.

[15] V. Ranganathan, "Developers Beware: Security Vulnerabilities Resulting from Poor Coding Practices," Developers Beware: Security Vulnerabilities Resulting from Poor Coding Practices, 14-Mar-2015. .

[16] G. A. Campbell and P. P. Papapetrou, SonarQube in Action, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.

[17] D. E. Krutz et al., "A Dataset of Open-source Android Applications," in Proceedings of the 12th Working Conference on Mining Software Repositories, Piscataway, NJ, USA, 2015, pp. 522–525.

[18] D. E. Krutz, N. Munaiah, A. Meneely, and S. A. Malachowsky, "Examining the Relationship between Security Metrics and User Ratings of Mobile Apps: A Case Study," in Proc.\ WAMA, 2016, pp. 8–14.

[19] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications," in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, New York, NY, USA, 2013, pp. 73–84.

[20] L. A. Zadeh, "Fuzzy logic = computing with words," IEEE Trans. Fuzzy Syst., vol. 4, no. 2, pp. 103–111, May 1996.

[21] P.-N. Tan, M. Steinbach, V. Kumar, and others, Introduction to data mining, Indian Subcontinent Edition., vol. 1. UP, India: Pearson Addison Wesley Boston, 2006.

[22] G. Brock, V. Pihur, S. Datta, and S. Datta, "clValid: An R Package for Cluster Validation," J. Stat. Softw., vol. 25, no. 1, pp. 1–22, 2008.

[23] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[24] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 485–496, Jul. 2008.

[25] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, Jan. 2007.

[26] B. Schölkopf et al., "Comparing support vector machines with Gaussian kernels to radial basis function classifiers," Signal Process. IEEE Trans. On, vol. 45, no. 11, pp. 2758–2765, 1997.

[27] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[28] P. Cunningham and S. J. Delany, "k-Nearest neighbour classifiers," Mult. Classif. Syst., pp. 1–17, 2007.

[29] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, Classification and regression trees. CRC press, 1984.

[30] R. Kohavi and others, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Ijcai, 1995, vol. 14, pp. 1137–1145.

[31] D. Powers, "Evaluation: From Precision, Recall and F Factor to ROC, Informedness, Markedness & Correaltion," Sch. Inform. Eng. Flinders Univ. S. Aust. Adel., 2007.