An investigation study for risk calculation of security vulnerabilities on android applications

Radhwan M. Abdullah[1], Abedallah Zaid Abualkishik[2], Najla Matti Isaacc[1], Ali A. Alwan[3], Yonis Gulzar[4]
[1]Division of Basic Sciences, College of Agriculture and Forestry, University of Mosul, Mosul, Iraq
[2]College of Computer Information Technology, American University in The Emirates, Dubai, United Arab Emirates
[3]School of Theoretical and Applied Science, Ramapo College of New Jersey, New Jersey, United States
[4]Department of Management Information Systems, College of Business Administration, King Faisal University, Al Ahsa, Saudi Arabia

Article Info

Article history:

Received Jul 28, 2021
Revised Dec 24, 2021
Accepted Jan 13, 2022

Keywords:

Access control
Android
Predictive analysis
Risk assessment
Security metrics

ABSTRACT

Applications within mobile devices, although useful and entertaining, come with security risks to private information stored within the device such as name, address, and date of birth. Standards, frameworks, models, and metrics have been proposed and implemented to combat these security vulnerabilities, but they remain to persist today. In this review, we discuss the risk calculation of android applications which is used to determine the overall security of an application. Besides, we also present and discuss the permission-based access control models that can be used to evaluate application access to user data. The study also focuses on examining the predictive analysis of security risks using machine learning. We conduct a comprehensive review of the leading studies accomplished on investigating the vulnerabilities of the applications for the Android mobile platform. The review examines various well-known vulnerabilities prediction models and highlights the sources of the vulnerabilities, prediction technique, applications and the performance of these models. Some models and frameworks prove to be promising but there is still much more research needed to be done regarding security for Android applications.

Corresponding Author:

Yonis Gulzar
Department of Management Information Systems, College of Bussiness Administration
King Faisal University, Al-Ahsa, Saudi Arabia
Email: ygulzar@kfu.edu.sa

1. INTRODUCTION

Mobile devices are very popular today and play a critical role in the everyday lives of most humans [1]. The majority of these devices are so useful because they contain android applications [2]-[5]. In most cases, these applications require sensitive personal information such as name, address, and date of birth. Security vulnerabilities in these applications can be catastrophic to the users. To combat these vulnerabilities, we must perform a thorough analysis and make or utilize a model to foresee these vulnerabilities. In this literature review, we will explore a few security measures that have been used to do just that. The first measure or model that we decided to conduct more research on was risk calculations, which should be used to determine how secure an application is. The second measure we decided to explore was access control, for this metric we decided to dive deep into things such as permission-based access and healthcare applications where user privacy is even more critical. The last measure we chose to review is predictive analysis, we knew that there has been a huge increase in the popularity of machine learning and artificial intelligence. We came to know that this could be very beneficial to finding security vulnerabilities and allowing us to enjoy our apps without fear of cyberattacks.

There have been many studies focused on security metrics for Android Mobile applications as a result of the recent increase of research interest in the area of Android smartphone security. Gonzalez *et al*. [6] investigated string offset order, a new easy-to-extract function. They were able to recognize the symptoms of reverse-designed Android applications using their tool, which did not require any further research. They used datasets from the Android Malware Genome Project, Droid Analytics, and Drebin to test the String Order metric's capabilities. In addition, over 5,000 Android applications were retrieved from the Google Play Store, and over 80,000 samples from the Virus Total service were analysed on a wide scale. In a comprehensive review of some repackaging exploration techniques. Offline and web technologies are the two basic forms of technology. They each have their feature. A technology that is used offline cannot be replaced by a tool that is used online and vice versa. Offline technologies are for the smartphone store owner's direct use, while online technologies are for Android users' direct use. We investigate a variety of online and online-related technology. These technologies describe a subset of technologies that use various features and metrics to discover application similarities [7]-[10].

The research conducted by Ordoñez-Ordoñez *et al*. [11] examines the reliability of mobile banking applications from both the inside and out. The authors used blog mining as a research method to comb through blog posts about mobile banking app protection. Security threats, assurance protocols/best practices, and potential security patterns are outlined to assist banks and customers in reducing the security risks involved with a range of financial applications. A study presented by Zheng *et al*. [12] looked at numerous security vulnerabilities in Android portable applications, especially for sensitive applications, and concentrated on analysing popular security attacks on cell phone applications. Preliminary studies to determine the feasibility and complexity of applying security attacks to the vital mobile mission application, finding current solutions and research holes, and recommending research directions Using numerous hacking methods and tactics, we successfully conducted three repackaging attacks to obtain access to victim files. We evaluate the extent of risk and recommend technological mitigation by evaluating these scenarios. The research carried out by Khokhlov and Reznik [13] describes information protection and quality assessment approach based on a probabilistic assessment of disruptive behaviours that exploit Android operating system features and bugs in the same way that installed apps can. It depicts the Android OS engineering functionality associated with protection and major security tools. The paper demonstrates the measurements chosen for information security assessment and the approach developed to combine them, resulting in an overall security evaluation score that addresses sensor-based information dependability. The count of instances of the total security assessment ranking is added and dissected.

The remainder of this paper is organized as follows. Section 2 elaborates the risk value calculation of applications in the marketplace. The discussion encompasses elucidating the vulnerabilities of the major changes to the Android permission system and describing the levels of threat based on the given score. In section 3, the access control security in Android applications has been presented and examined. A description of the mobile app development affected by access control vulnerabilities has been given. Furthermore, various access control metrics that are used to assess the risk of access control functionality in mobile applications are elaborated. Several frameworks that could be utilised to moderate data access for mobile application development are also explained. The predictive analysis to combat security vulnerabilities is demonstrated in section 4. The discussion includes describing the security mechanisms that have evolved with millions of applications in the past, as well as innovative technologies that could offer more sophisticated security. The conclusion is depicted in the final section, section 5.

2. RISK VALUE CALCULATION OF APPLICATIONS IN THE MARKETPLACE

Smartphone applications are an essential part of our daily life. From social media to banking to access medical reports, we are using mobile applications for ease of access [14]-[17]. More than 80% of smartphones run on the Android operating system [18]. Such dominance in the market also makes Android an exclusive target for mobile threats. Security mechanisms are needed for such applications considering the sensitivity of the data. When we want to install any app from the Android Playstore, it does ask permission to access personal data, location, camera etc. Some malicious applications take advantage of these permissions, as most of the users tend to accept the clause without thinking about the impacts due to the complex language in the clause or the lack of time or understanding. Once these malicious applications gain access to the personal data, they can access our location, photos, camera, internet, or anything they can use for the attack [19]-[24]. Due to the massive collection of applications on the play store, it is difficult to recognize which applications are genuine and which are not. Since Android allows third-party developers applications to be released in the play-store security cannot be guaranteed from those applications as well. Also, some unofficial markets do not have any centralized control so untrusted developers can distribute malicious applications.

Playstore itself cannot track all the applications yet for its maliciousness and common users do not have the understanding to identify which applications can misuse their data. There should be a mechanism in the AppStore, that can indicate the severity of the risk posed by the applications if we accept the permission clause [25]-[30]. There needs to be some scoring system that can define how safe is the app to install without compromising personal data. Figure 1 shows the timeline of the Android evolution with vulnerabilities and version updates. On many occasions, attackers were able to gain root privileges to take over the control of the device.

Android applications put their users at risk in a variety of ways, such as by using code that may jeopardize user privacy or device integrity [31], [32]. The majority of existing security countermeasures for identifying unsafe applications have flaws, most of which are linked to devices comprehension and acceptance. As a result, consumers will benefit from a clear but successful method of determining if an application is secure. Android applications pose many risks to their users, e.g., by including code that may threaten user privacy or system integrity [31], [32]. Most of the current security countermeasures for detecting dangerous apps show some weaknesses, mainly related to users' understanding and acceptance. As a result, consumers will benefit from an easy but reliable method for determining if an application is safe to install or not.

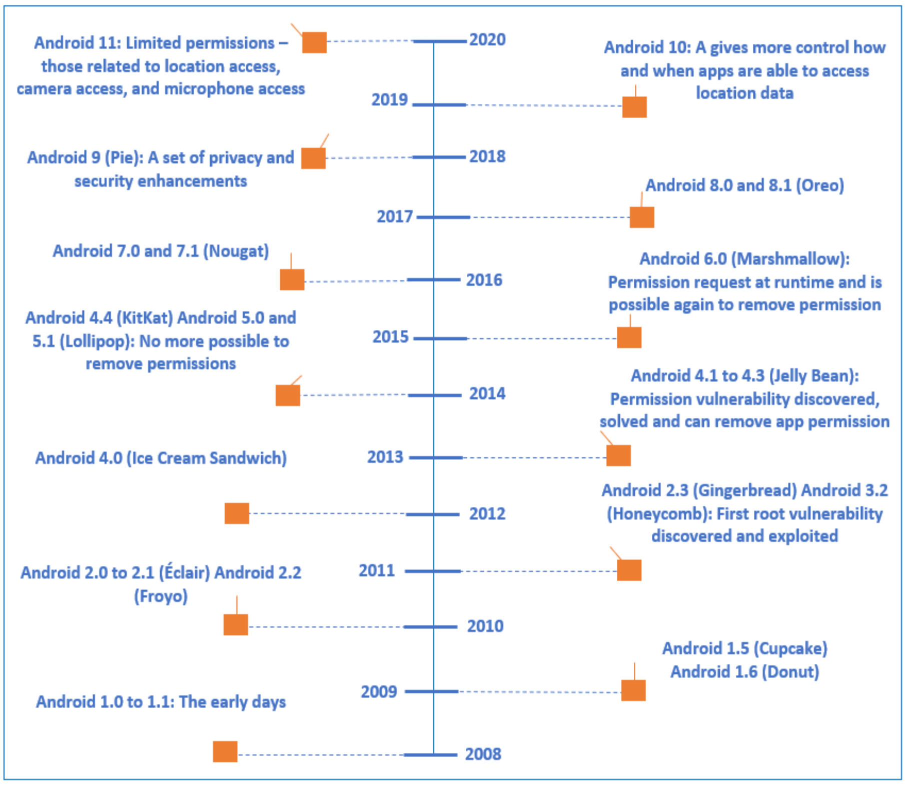
Android 11: Limited permissions – those related to location access, camera access, and microphone access
2020
Android 10: A gives more control how and when apps are able to access location data
2019
Android 9 (Pie): A set of privacy and security enhancements
2018
Android 8.0 and 8.1 (Oreo)
2017
Android 7.0 and 7.1 (Nougat)
2016
Android 6.0 (Marshmallow): Permission request at runtime and is possible again to remove permission
Android 4.4 (KitKat) Android 5.0 and 5.1 (Lollipop): No more possible to remove permissions
2015
2014
Android 4.1 to 4.3 (Jelly Bean): Permission vulnerability discovered, solved and can remove app permission
Android 4.0 (Ice Cream Sandwich)
2013
2012
Android 2.3 (Gingerbread) Android 3.2 (Honeycomb): First root vulnerability discovered and exploited
Android 2.0 to 2.1 (Éclair) Android 2.2 (Froyo)
2011
2010
Android 1.5 (Cupcake) Android 1.6 (Donut)
2009
Android 1.0 to 1.1: The early days
2008

Figure 1. Vulnerabilities and timeline of major changes to the android permission system [33]

Multi-criteria software evaluator of confidence for AndrOID (MAETROID) is one of the proposed frameworks for evaluating the trustworthiness of Android applications as depicted in Figure 2 [32]. It evaluates the authenticity of the app before installation using analytical hierarchy process (AHP) [34]. This evaluation is labelled to the app to rank the risk metric. The goal of this framework is to assign labels to the applications from Trusted, Medium or high risk by computing and comparing alternatives and criteria. It calculates the threat score using a weighted and normalized total of single threat scores obtained by manually reviewing all of Android's permissions. The MAETROID platform has been turned into an Android application [32]. When a new app is getting installed, MAETROID intercepts the information and does the analysis and shows the threat score to the user. Table 1 outlines the details of the threat levels and their meaning. The threat score is used to determine privacy threat, financial threat, system threat, and global threat.

Table 1. Threat levels [19]

Given Score	Meaning
0	No Threat
0.2	Negligible
0.4	Limited
0.6	Significant
0.8	Relevant
1	Maximum

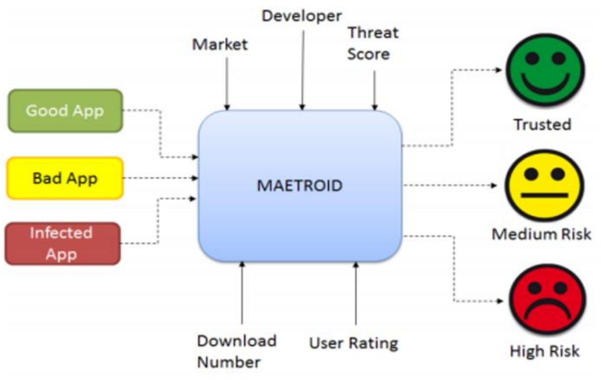

Figure 2. MAETROID classification process [19]

Referenced articles mention different solutions proposed to increase security measures while installing any Android app. Several proposals suggest that a scoring system will help ensure common users are aware of these impacts before they can install the app and it will help them to decide to accept the risk or not. Some suggested calculating security risk scores based on permissions demanded by Android applications using crawl models and statistical measurements [35].

For an android device, the risk value calculation process is used to measure risk values. Optimal risk value is evaluated based on a set of selected malware and normal applications. Application is processed to extract data and then the risk is estimated using the risk parameters and then the final risk value is calculated. Extraction of data gets the permissions, API calls, resources, intentions. High rated malicious applications are used for sampling the risk associated. This way risk value is calculated, and the potential of the malicious application can be determined. When a user wants to install the application, risk value calculation (RVC) extracts features from the APK file and they are matched with the database stored with high-risk features and then it can accurately estimate the risk the applications possess as shown in Figure 3 [18].

Figure 4 shows the RVC findings as well as the associated risk steps [18]. The percentage of malicious applications is used to pick and identify each sorted list. The alarming rate and identification rate are the terms used to describe these two acts. Since there is no absolute alert rate threshold value for our assessment process, and it is also independent of the risk value collection, we must make a rational distinction between them at that stage. Similar to RVC, one more scoring system free update mean (FUM) scoring is available to use [2]. Figure 5 illustrates the list of vulnerabilities discovered and patched over time as reported in [2].

The FUM Security matrices are used to rate system vendors and network operators based on their ability to provide upgrades and their vulnerability to critical vulnerabilities. a difficult-to-game composite FUM ranking (§5.7). By reducing knowledge asymmetry, the FUM [1] score allows private and public sector consumers, as well as individuals, to make more educated buying decisions. These metrics free, upgrade and mean (F, U, M) together calculate a platform's protection in terms of known vulnerabilities and upgrades by calculating the proportion of operating devices free of critical vulnerabilities, the proportion of devices receiving the most recent Android version updates, and the mean amount of vulnerabilities yet to be patched. The FUM score is a ten-point scale that can be measured [1]. We give F more weight because it is the most important metric. We map M into the range (0–1) since it is an unbounded positive real number. As a result, we have the FUM score:

$$FUM_{score} = 4F + 3U + 3\frac{2}{1+e^{M}} \quad (1)$$

The FUM security metric evaluates, and rates system vendors and network operators based on their ability to provide upgrades and their vulnerability to crucial vulnerabilities. Buyers and regulators will use the measure to see which system vendors and network providers have upgrades and which do not. Considering MAETROID is already implemented with the Android marketplace, and it is easier for the user to understand the threat score before installing the app, it is the preferred framework to use for preventing malicious applications.

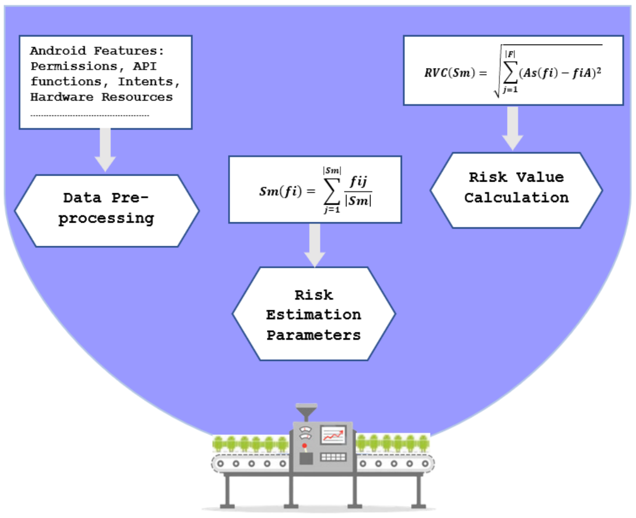
Android Features: Permissions, API functions, Intents, Hardware Resources
Data Pre-processing
$Sm(fi) = \sum_{j=1}^{|Sm|} \frac{fij}{|Sm|}$
Risk Estimation Parameters
$RVC(Sm) = \sqrt{\sum_{j=1}^{|F|} (As(fi) - fiA)^2}$
Risk Value Calculation

Figure 3. RVC risk estimate model [18]

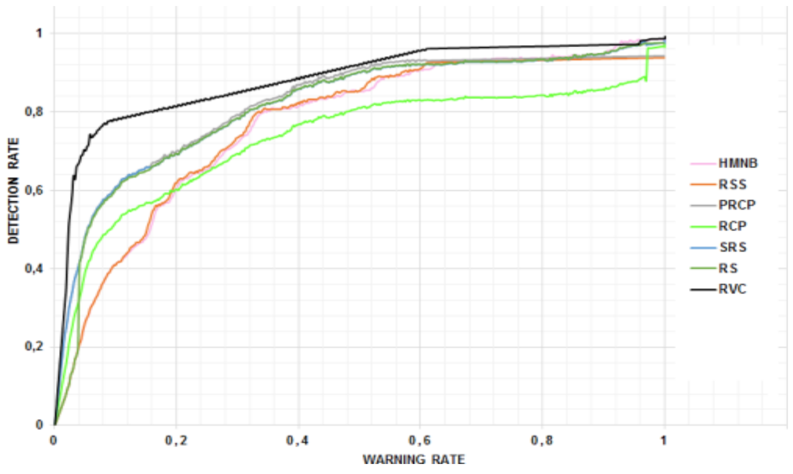

Figure 4. Measure the risk rate [36]

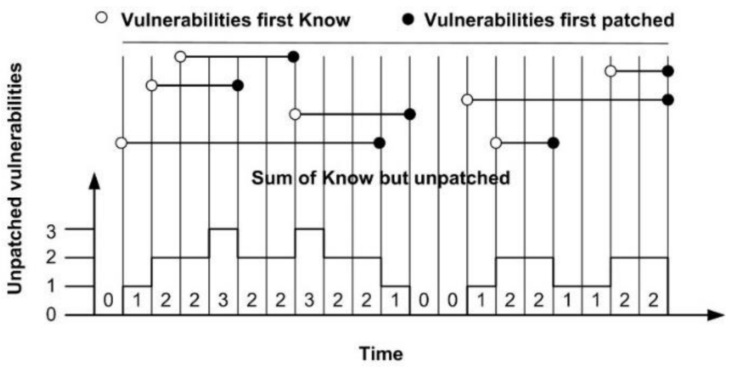
Vulnerabilities first Know
Vulnerabilities first patched
Sum of Know but unpatched
Unpatched vulnerabilities
0
1
2
3
0 1 2 2 3 2 2 3 2 2 1 0 0 1 2 2 1 1 2 2
Time

Figure 5. Vulnerabilities discovered and patched over time [2]

3. METHOD

A growing area of concern for developers and users, regarding data security in mobile applications, is proper access control. Applications that do not take consideration in implementing an access control model can be taken advantage of to access private user data. This is an area of research that is still looking into developing frameworks and models to use as a metric to incorporate access control for mobile applications. The following subsections present and discuss the access control security in Android applications.

3.1. Mobile application development affected by access control vulnerabilities

A cornerstone to access control in Android applications is permission-based access. Although most applications redirect permissions to the OS, some applications require select permissions which grant them access to additional users' information. This has the possibility of being exploited by having applications communicate therefore expanding permissions and allowing user data to be more easily accessible. The challenges to current permission-based access have mostly to do with user and developer permission comprehension [36]. If the stakeholders of a mobile app do not comprehend when permission access to resources on a mobile device is suitable, then it can be an opening for malicious behaviour. The vulnerabilities of access control may not be acknowledged by most users of popular applications [37]-[40], however, this can impede development in mobile applications that require the security of confidential information such as medical records. Having a metric for access control security could help developers have a better understanding of appropriate permissions while minimizing the risk of possible malicious behaviour.

3.2. Metrics for access control security

Access control security is essential in some industries such as the medical field due to the importance of private medical info rmation. For mobile applications to be used in sharing critical information it is important to have a model or framework to assess data security risks. Although there are a few standards in place to help secure private information, there is ongoing research on creating a set of metrics to best assess the risk of access control functionality in mobile applications. Below are three recently proposed access control metrics and frameworks that developers can use to moderate data access, some of which are inspired by securing private medical information. Socio-technical risk-adaptable access control (SoTRAACE). Model of risk assessment: health care professionals rely heavily on role-based access control. Due to access control vulnerabilities in mobile Android applications, it can be a great security risk if roles could obtain permission to unauthorized medical information. To reduce security risk a new hybrid risk assessment to the access control model called SoTRAACE has been proposed.

The risk assessment metric is calculated using risk ranking with a respective risk factor, and the weight of the risk which was determined by experts in a New Delphi study [19]. The prototype app that is used to implement SoTRAACE handles risk-adaptable decisions by utilizing the user's location and connections [19]. This permission-based access control can be largely customizable based on quantitative and qualitative data provided by its users. However, there is little available research to compare the hybrid-risk assessment procedures of this model to others. Also, the risk factors used to calculate risk assessment are small and could include more context to the data being assessed. OWASP Top 10 Mobile Risks: Another healthcare inspired access control framework for developers is open web access security project (OWASP) Top 10 Mobile Risks. The framework consists of 10 metrics that allows a technical auditor to identify the security vulnerabilities of their application as demonstrated in Figure 6 [41]. The metrics were used to evaluate WebMD Allergy and Track My Medical Records mobile applications. Using the OWASP Top 10 Mobile Risks checklist a security risk of Track My Medical Records was discovered [26]. Although the risk checklist proved that track my medical records had a security vulnerability, the effectiveness of the framework needs to be tested on additional samples.

HybridGuard permission and policy framework: There has been a rise in web-based mobile applications that have proven to be a security risk factor for data access. Since most web-based mobile applications rely on JavaScript it can be hard to deduce where the requested data access is coming from and if exposed APIs are being taken advantage of. HybridGuard is a framework that uses principle-based, stateful policies, to provide access control to web-based mobile applications without modifying hybrid frameworks or mobile applications [19]. When an API invocation is received it will be tested by the stateful policies within a policy manager acting as a monitor. If the API invocation is permitted, then access will be granted as elucidated in Figure 7 [42]. The HybridGuard framework can be useful for developers in the creation of access control policies and metrics to assess security by recording instances of unauthorized access, what data was trying to be accessed, and data exposure by authorized API invocations. Although HybridGuard has been tested only on 6 mobile applications more testing needs to be done with promises of HybridGuard allowing users to download the framework as an app and set their policies.

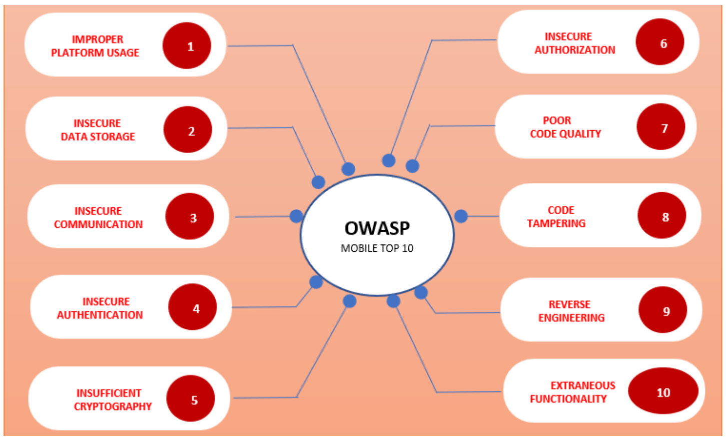

Figure 6. OWASP top 10 mobile risks [41]

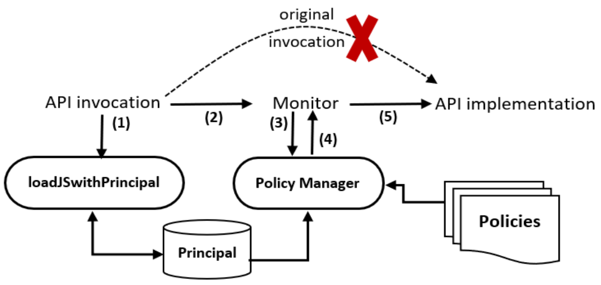

Figure 7. The architecture of the HybridGuard framework [42]

4. RESULTS AND DISCUSSION

As mentioned before security vulnerabilities in Android applications can affect the user of the application or the device itself. As engineers, it is a necessity to consider various measures to ensure the security of your app. We will always start with the measures that have worked in the past for millions of applications, but we must also consider new solutions that could provide more advanced security. We chose to take a deep dive into how engineers have one would assume that if a threat can be detected long before it reaches the application that it would be highly effective, predictive analysis and machine learning techniques explore this assumption deeply. The first report we looked at addressed the following: During the study, about 200 open-source applications that were available on the Android app store were considered. This experiment necessitated the use of an application's source code. FBReader was the application of choice (free e-book reader). The application had about 39K lines of code and had received over 3700 updates in the code shop. Classes that were discovered in the vault and had a position with outside libraries were ignored. The application has been downloaded more than a million once and has received positive feedback. For more information, see Figure 1. We downloaded several copies of the source code from the FBReader source code databases. Table 2 describes the FBReader versions for source code analysis as reported in [43]. We found vulnerabilities for each edition by utilizing fortify's source code analyzer (SCA), a robotized static code review platform that tests program source code then produces a report containing all of the vulnerabilities discovered, along with areas and portrayals [44]. For each iteration, we have calculated a comprehensive collection of object-oriented code metrics. The JHawk 5 tool was used [45], which gathers about 40 separate measurements for each level. Finally, we created a categorization model using support vector machines (SVM) that uses code measurements to predict when a Java class is vulnerable (result).

Table 2. FBReader versions for source code analysis [43]

Metrics	Description
0.75	OCT 2010
0.99.0	JAN 2011
1.0.0	APR 2011
1.1.0	JUN 2011
1.20	OCT 2011

Since 2008, the number of Android flaws has risen. As a result, it is reasonable to assume that the Android ecosystem's security status is deteriorating year after year. One potential explanation is that, as the mobile Internet evolves, the number of software available for consumption grows, but the majority of developers are inexperienced [41]. Static analysis tools are used to figure out the number of vulnerabilities an application might have for numerous reasons. The first reason is that thousands of applications are added to the google play store with only a small number of vulnerabilities reports. Furthermore, the vulnerabilities that are reported are not complete and lack the historical data related to the vulnerabilities that are needed to build the predicting model. The second most important reason is that the static analysis tool can calculate the vulnerabilities repeatedly. The static analysis tool provides the ability to get plenty of vulnerabilities for thorough analysis. However, some of the source code analyser (SCA) vulnerabilities are false positives that we further discuss in the following sections.

The fortify source analyser was used as a static analysis tool to scan Android mobile applications. This tool reports the location of the vulnerabilities along with the criticality and the category of vulnerabilities. the vulnerabilities are computed using the following: for every java class if the variable is equal to zero its mean there were no vulnerabilities reported for this java class. On the other side, if the value is equal to one it means that vulnerabilities are reported for this java class. Table 3 represents the results of the predicted vulnerable components of classes. The results have been characterized into the following five sections: type of prediction features used, the foundation of the vulnerability data, the kind of prediction technique, the applications used for the validation, and the available performance meters [19], [42]. For the training data collection, we use the 0.7.6 variant of the FBReader and the SVM vector machine to construct the model for predicting the vulnerable classes and components. The radical base function (RBF) is utilized to build the classifier. We set the parameters for the RBF function such as COST 100.0 and the value of gamma is 0.001. The total number of the components or java class is 215 for the training from which 59 is labelled as vulnerable.

We use five parts of cross-validation to evaluate the performance of the training data. Figure 8 represents the accuracy of each part. The performance of the model is evaluated by using three metrics which are accuracy, precision, and recall.

- Accuracy: is the percentage of accurate outcomes, such as true positive (TP), the weakest (vulnerable) class that is correctly classified as weak) and true negative (TN), the strongest (vulnerable) class that is correctly classified as strong) (TN, the non-weak class that is correctly classified as negative).
- Precision: refers to the probability of a vulnerable component being identified as such. It is determined using the formula TP/ (TP+FP).
- Recall: refers to the probability that a part identified as vulnerable is not vulnerable. It's estimated using the formula TP/ (TP+ FN).

Figure 9 represents the distribution of the vulnerabilities for the components or classes of version 0.7.6 it displays as a density function. After 20 vulnerabilities the function of density reaches the x-axis. For the remaining 4 versions, the vulnerabilities are distributed as the same with a slight difference of collisions. Moreover, Figure 10 illustrates the vulnerability evolution over time that have been reported by Krutz in his work in [46]. From the figure, it is clear that the severity 3 class has the smallest number of vulnerabilities on the considered versions with up to 50 vulnerabilities. However, the figure also denotes that the class of severity 2 has the highest number of vulnerabilities with almost 370 vulnerabilities.

The version of FB Reader is analyzed and the evolution is represented in Figure 11. We notice that the number of 3.0 criticality vulnerabilities remains consistent across versions. As a consequence, changing the number of vulnerabilities to 2.0 vulnerabilities changes the criticality. Once the results and the view are provided it is possible to achieve high accuracy and precision to construct the prediction and classification model. The black plane line in Figure 11 represents model accuracy, whereas the black dotted line represents the classifier's accuracy of any java class listed as not vulnerable. Many other classes are also predicated as non-vulnerable. The dotted line that shows accuracy is taken as a baseline to relate the performance of the prediction model.

The model we created is somewhere in the range of 6.0% and 8.4% more precise than the gauge. Note, the separation between the two lines recoils for some time. Our method has a high level of precision, with a success rate of more than 80%. The accuracy of the prediction model for each of the four variants tested is also seen in Figure 8. Precision varies between 75.9% and 81.5%. This means that when a Java class is flagged as vulnerable by the model, it is extremely accurate. However, as seen in the figure, the recall estimate is poor. Recall rates range from 37.9% to 42.3%. As a consequence, the model is unreliable when it predicts that a Java class also isn't vulnerable. In conclusion, the model can then be used to coordinate the audit of Java classes or modules that pose a high risk of vulnerabilities. In any case, the overall list of extremely vulnerable classes could be greater than expected or projected.

Table 3. Vulnerabilities prediction models proposed by Stevenson [47]

Title of Studies	Predictors used	Vulnerability sources	Prediciton technique	Applications	Performace
Evaluting complextiy, code churn, and developer activity metrics as indicators of software	Software metrics, code churn, developer activity metrics	MFSA, Red Hat Bugzilla, Red Hat pacakge management system (RPM)	Logistic regression	Firefox, Red Hat Linux kernel	Firefox---3% recall: 79-86, fall-out: 2225%; Red Hat Linux Kernel – precision: 5%, recall 80-90%, fall-out 22-25
Predicting vulnerable software components	Imports, function calls	MFSA	SVM	Mozilla	Precision: 70%, recall: 45%
Can traditional fault prediciton models be used for vulnerability prediciton	Software metrics, code churn, developer activity metrics	MFSA	Logistic regression	Firefox	Precision: 12%, recall: 83%
Is complexity really the enemy of the software security/ An emperical model to predict security vulnerabilities using code complexity metrics	Code complexity metrics	MFSA/CVE/Bugzilla	Logistic regression	Firefox JSengine	Recall: 3-93%, accuracy 43-98% fall-out: 0 – 58%
Using complexity coupling, and cohesion metrics as early indicators for vulnerabilities	complexity coupling, and cohesion metrics	MFSA/Bugzilla	Naïve bayes, C4.5 Decision Tree, Random Forest, Logistic regression	Firefox	C4.5 decsion tree- mean precision: 72%, mean recall: 74%, mean accuracy: 73%, mean fall-out: 29%
Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista	code churn, code complexity, dependency measures, code coverage, organizational metrics, actual dependencies	NVD	Logistics regression/SVM	Windows vista	Median precision 60-67%; median recall 20-40%
Toward non-security failures as a predictor of security faults and failures	Non-security failure reports	Cisco security reports	CART	Cisco software system	Recall: 57%; fall-out: 48%
Predicting vulnerable software components with dependency graphs	Component dependency graphs	MFSA/CVE/Bugzilla	Bayesian network, Naïve bayes, neural networks, random forest, SVM	Firefox JS Engine	Precision: 61-68%, recall: 60-61%, accuracy: 84-85%, fall-out: 9-10%
Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities	SQL hotspots	Trac issue report	Logistic regession	Wordpress, Wikkawiki	Wordpress – average precision: 28%, average recall: 24%, Wikkawiki – average precision: 62%, average recall: 39%

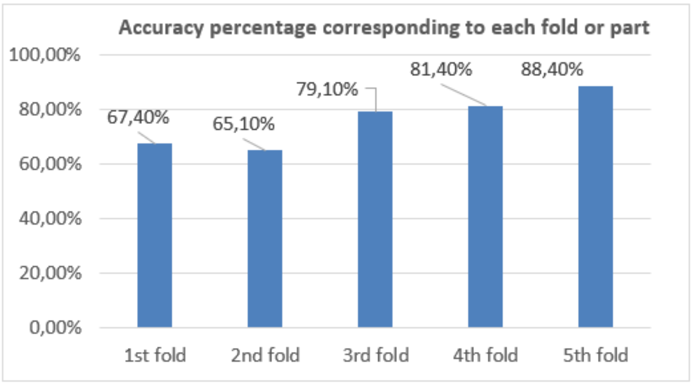

Figure 8. Accuracy percentage [48]

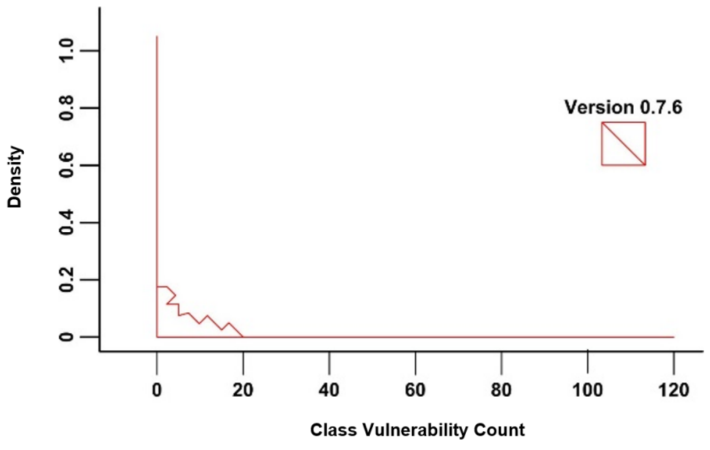

Figure 9. Scattering of vulnerabilities [49]

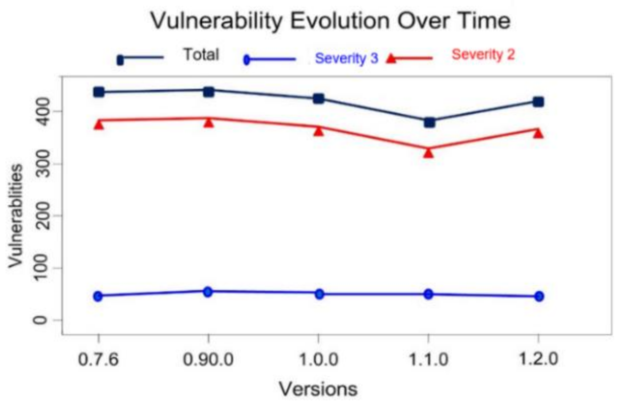

Figure 10. Progress of vulnerabilities [46]

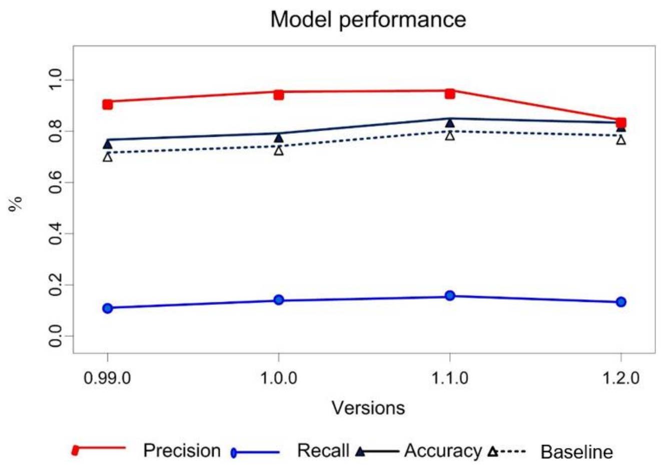

Figure 11. Results of classification model [50]

5. CONCLUSION

The success of Android technology and applications has made them more tempting to cybercriminals. The malware detection program could be introduced and delivered in the form of a smartphone application that communicates the scanning results to the customer in an easy-to-understand manner. Protecting and making users aware of the malicious applications before installing is the first step towards the security of the Android applications. Even though timely delivery can prevent critical vulnerabilities, not many applications provide prompt updates to patch the pitfalls. FUM scoring system can help overcome the asymmetry between manufacturer and consumer. Often users are duped into downloading malicious applications that seem to be legitimate. This arises largely because consumers are more concerned with the app's success and user reviews than with the consent invasions. MAETROID framework can tag such applications as per the threat score before installing them, so users are aware of the consequences of accepting the permissions clause. Assessments done by the RVC method shows the high-risk rating for the malicious applications which can be used as a metric when choosing an app. RVC uses a large database to use the historic data and compares the high-risk applications against them to analyse the risk rate. To properly control third-party mobile app marketplaces, we need a robust vetting process.

ACKNOWLEDGMENT

The authors are very grateful to the University of Mosul/College of Agriculture and Forestry for their provided facilities, which helped to improve the quality of this work.

REFERENCES

[1] S. Parasuraman, A. T. Sam, S. W. K. Yee, B. L. C. Chuon, and L. Y. Ren, "Smartphone usage and increased risk of mobile phone addiction: A concurrent study," *International Journal of Pharmaceutical Investigation*, vol. 7, no. 3, p. 125, 2017, doi: 10.4103/jphi.jphi_56_17.

[2] D. R. Thomas, A. R. Beresford, and A. Rice, "Security metrics for the Android ecosystem," *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 87–98, 2015, doi: 10.1145/2808117.2808118.

[3] Y. Yang, X. Du, and Z. Yang, "PRADroid: Privacy risk assessment for android applications," *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, pp. 90–95, 2021, doi: 10.1109/csp51677.2021.9357608.

[4] J. Xiao, S. Chen, Q. He, Z. Feng, and X. Xue, "An android application risk evaluation framework based on minimum permission set identification," *Journal of Systems and Software*, vol. 163, p. 110533, 2020, doi: 10.1016/j.jss.2020.110533.

[5] J. Cui, L. Wang, X. Zhao, and H. Zhang, "Towards predictive analysis of Android vulnerability using statistical codes and machine learning for IOT Applications," *Computer Communications*, vol. 155, pp. 125-131, 2020, doi: 10.1016/j.comcom.2020.02.078.

[6] H. Gonzalez, A. A. Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani, "Exploring reverse engineering symptoms in Android apps," *Proceedings of the Eighth European Workshop on System Security*, 2015, pp. 1-7, doi: 10.1145/2751323.2751330.

[7] S. Rastogi, K. Bhushan, and B. B. Gupta, "Android applications repackaging detection techniques for smartphone devices," *Procedia Computer Science*, vol. 78, pp. 26-32, 2016, doi: 10.1016/j.procs.2016.02.006.

[8] G. Cicirelli, R. Marani, A. Petitti, A. Milella, and T. D'Orazio, "Ambient assisted living: A review of technologies, methodologies and future perspectives for Healthy Aging of Population," *Sensors*, vol. 21, no. 10, p. 3549, 2021, doi:10.3390/s21103549.

[9] K. Kim, J. Kim, E. Ko, and J. H. Yi, "Risk assessment scheme for mobile applications based on tree boosting," *IEEE Access*, vol. 8, pp. 48503–48514, 2020, doi: 10.1109/access.2020.2979477.

[10] H. H. Goh, S. y. Sim, J. Shaari, N. A. Azali, C. W. Ling, Q. S. Chua, and K. C. Goh, "A review of lightning protection system - risk assessment and application," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 8, no. 1, pp. 221-229, 2017, doi: 10.11591/ijeecs.v8.i1.pp221-229.

[11] P. F. Ordoñez-Ordoñez, D. D. Herrera-Loaiza, and R. Figueroa-Diaz, "Vulnerabilities in banking transactions with mobile devices Android: A systematic literature review," *Communications in Computer and Information Science*, pp. 104-115, 2018, doi: 10.1007/978-3-030-05532-5_8.

[12] X. Zheng, L. Pan, and E. Yilmaz, "Security Analysis of Modern Mission Critical Android mobile applications," *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1-9, 2017, doi: 10.1145/3014812.3014814.

[13] I. Khokhlov and L. Reznik, "Data Security Evaluation for mobile Android devices," *2017 20th Conference of Open Innovations Association (FRUCT)*, pp. 154-160, 2017, doi: 10.23919/fruct.2017.8071306.

[14] V. Venugopal, S. Poonguzhali, S. Sadhana, S. T. Venkateswaran, and K. Maheshkumar, "Impact of short term mobile phone abstinence on undergraduate medical students: A qualitative study," *Journal of Biomedical Research & Environmental Sciences*, vol. 1, no. 7, pp. 299–302, 2020, doi: 10.37871/jbres1158.

[15] J. Chen *et al*., "Semantics-aware privacy risk assessment using self-learning weight assignment for mobile apps," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 15–29, 2021, doi: 10.1109/tdsc.2018.2871682.

[16] A. Mehmood, A. Nadeem, M. Ashraf, M. S. Siddiqui, K. Rizwan, and K. Ahsan, "A fall risk assessment mechanism for elderly people through muscle fatigue analysis on data from Body Area Sensor Network," *IEEE Sensors Journal*, vol. 21, no. 5, pp. 6679–6690, 2021, doi: 10.1109/jsen.2020.3043285.

[17] K. O. Elish, H. Cai, D. Barton, D. Yao, and B. G. Ryder, "Identifying mobile inter-app communication risks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 90-102, 2020, doi: 10.1109/tmc.2018.2889495.

[18] L. Er-rajy, M. A. El Kiram, and M. El Ghazouani, "New security risk value estimate method for Android applications," *The Computer Journal*, vol. 63, no. 4, pp. 593-603, 2019, doi: 10.1093/comjnl/bxz109.

[19] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Risk analysis of android applications: A user-centric solution," *Future Generation Computer Systems*, vol. 80, pp. 505–518, 2018, doi: 10.1016/j.future.2016.05.035.

[20] P. Hendikawati, M. Z. Zahid, and R. Arifudin, "Android-based Computer Assisted Instruction Development as a learning resource for supporting self-regulated learning," *International Journal of Instruction*, vol. 12, no. 3, pp. 389–404, 2019, doi: 10.29333/iji.2019.12324a.

[21] P. Weichbroth and Ł. Łysik, "Mobile security: Threats and best practices," *Mobile Information Systems*, vol. 2020, pp. 1–15, 2020, doi: 10.1155/2020/8828078.

[22] A. M. Radhi, "Risk assessment optimization for decision support using intelligent model based on fuzzy inference renewable rules," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 2, p. 1028-1035, 2020, doi: 10.11591/ijeecs.v19.i2.pp1028-1035.

[23] B. R. Greene, S. J. Redmond, and B. Caulfield, "Fall risk assessment through automatic combination of clinical fall risk factors and body-worn sensor data," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 3, pp. 725–731, 2017, doi: 10.1109/jbhi.2016.2539098.

[24] Z. Tian, J. Xiao, H. Feng, and Y. Wei, "Credit risk assessment based on Gradient Boosting Decision tree," *Procedia Computer Science*, vol. 174, pp. 150–160, 2020, doi: 10.1016/j.procs.2020.06.070.

[25] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel, and M. Smith, "Sok: Lessons learned from Android Security Research for appified software platforms," *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 433–451, 2016, doi: 10.1109/sp.2016.33.

[26] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "IOT privacy and security: Challenges and solutions," *Applied Sciences*, vol. 10, no. 12, p. 4102, 2020, doi: 10.3390/app10124102.

[27] R. F. Rahmat, S. Purnamawati, H. Saito, M. F. Ichwan, and T. M. Lubis, "Android-based automatic detection and measurement system of highway billboard for tax calculation in Indonesia," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 14, no. 2, pp. 877-886, 2019, doi: 10.11591/ijeecs.v14.i2.pp877-886.

[28] A. ALFoudery, A. A. Alkandari, and N. M. Almutairi, "Trash basket sensor notification using Arduino with Android application," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 10, no. 1, pp. 120–128, 2018, doi: 10.11591/ijeecs.v10.i1.pp120-128.

[29] T. S. Gunawan, A. Mutholib, and M. Kartiwi, "Design of automatic number plate recognition on Android smartphone platform," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 5, no. 1, pp. 99–108, 2017, doi: 10.11591/ijeecs.v5.i1.pp99-108.

[30] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Towards automated risk assessment and mitigation of mobile applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 571–584, 2015, doi: 10.1109/tdsc.2014.2366457.

[31] N. Kalbo, Y. Mirsky, A. Shabtai, and Y. Elovici, "The security of IP-based video surveillance systems," *Sensors*, vol. 20, no. 17, p. 4806, 2020, doi: 10.3390/s20174806.

[32] 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," *Proceedings of the International Workshop on App Market Analytics*, 2016, pp. 8–14, doi: 10.1145/2993259.2993260.

[33] J. M. Yoon, "Siem owasp-zap and angry-IP vulnerability analysis module and interlocking," *Jouranl of Information and Security*, vol. 19, no. 2, pp. 83-89, 2019, doi: 10.33778/kcsa.2019.19.2.083.

[34] P. Moura, P. Fazendeiro, P. R. Inácio, P. Vieira-Marques, and A. Ferreira, "Assessing access control risk for mHealth: A delphi study to categorize security of health data and provide risk assessment for mobile apps," *Journal of Healthcare Engineering*, vol. 2020, pp. 1–14, 2020, doi: 10.1155/2020/5601068.

[35] A. Dashti, "Mobile Cloud Computing Security frameworks," *Cyber Security and Threats*, pp. 501–520, 2018, doi: 10.4018/978-1-5225-5634-3.ch027.

[36] M. Deypir, "Entropy-based security risk measurement for Android mobile applications," *Soft Computing*, vol. 23, no. 16, pp. 7303–7319, 2018, doi: 10.1007/s00500-018-3377-5.

[37] P. H. Phung, A. Mohanty, R. Rachapalli, and M. Sridhar, "Hybridguard: A principal-based permission and fine-grained policy enforcement framework for web-based mobile applications," *2017 IEEE Security and Privacy Workshops (SPW)*, pp. 147–156, 2017, doi: 10.1109/spw.2017.34.

[38] S. R. Mat, M. F. Razak, M. N. Kahar, J. M. Arif, and A. Firdaus, "A bayesian probability model for Android malware detection," *ICT Express*, 2021, doi: 10.1016/j.icte.2021.09.003.

[39] F. H. da Costa *et al.*, "Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for Android malware identification," *Journal of Systems and Software*, vol. 183, p. 111092, 2022, doi: 10.1016/j.jss.2021.111092.

[40] Y. Nan, Z. Yang, X. Wang, Y. Zhang, D. Zhu, and M. Yang, "Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in Mobile apps," *Proceedings 2018 Network and Distributed System Security Symposium*, 2018, pp. 1-15, doi: 10.14722/ndss.2018.23092.

[41] Govindraj Basatwar, "Global Business HeadA Techo-Commerical evangelist who create, Owasp Mobile top 10: Comprehensive guide to counter mobile app risks," *AppSealing*. 2021. Accessed: 26-Nov-2021 [Online]. Available: https://www.appsealing.com/owasp-mobile-top-10-a-comprehensive-guide-for-mobile-developers-to-counter-risks/

[42] P. H. Phung, R. S. V. Reddy, S. Cap, A. Pierce, A. Mohanty, and M. Sridhar, "A multi-party, fine-grained permission and policy enforcement framework for hybrid mobile applications," *Journal of Computer Security*, vol. 28, no. 1-2, pp. 1-31, 2020, doi: 10.3233/jcs-191350.

[43] I. I. Karayalcin, "The Analytic Hierarchy Process: Planning, priority setting, resource allocation," *European Journal of Operational Research*, vol. 9, no. 1, pp. 97–98, 1982, doi: 10.1016/0377-2217(82)90022-4.

[44] L. Yu, "From Android bug reports to Android bug handling process," *International Journal of Open Source Software and Processes*, vol. 7, no. 4, pp. 1–18, 2016, doi: 10.4018/ijossp.2016100101.

[45] A. Abraham, "Configuring Mobile Security Framework for static analysis," *Automated Security Analysis of Android and iOS Applications with Mobile Security Framework*, pp. 5–6, 2016, doi: 10.1016/b978-0-12-804718-7.00002-3.

[46] D. E. Krutz *et al.*, "A dataset of open-source Android Applications," *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, doi: 10.1109/msr.2015.79.

[47] J. Stevenson, "Android versions," *Android Software Internals Quick Reference*, pp. 3–5, 2021, doi: 10.1007/978-1-4842-6914-5_2.

[48] S. Salva and S. R. Zafimiharisoa, "Detection of intent-based vulnerabilities in Android Applications," *Emerging Trends in ICT Security*, pp. 397–417, 2014, doi: 10.1016/b978-0-12-411474-6.00024-4.

[49] E. Yasasin, J. Prester, G. Wagner, and G. Schryen, "Forecasting IT security vulnerabilities – an empirical analysis," *Computers & Security*, vol. 88, p. 101610, 2020, doi: 10.1016/j.cose.2019.101610.

[50] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014, doi: 10.1109/tse.2014.2340398.

BIOGRAPHIES OF AUTHORS

Radhwan M. Abdullah is currently a lecturer at College of Agriculture and Forestry, University of Mosul, Iraq. He earned his Master of Computer Science (Distributed Computing) in 2008 and Ph.D. in Computer Science (Networking) in 2014 from Universiti Putra Malaysia (UPM), Malaysia. His research interests include networking, Cloud and Fog Computing, IoT, Wireless Network, Routing in WiMAX. He can be contacted at email: radwanmas@uomosul.edu.iq.

Abedallah Zaid Abualkishik received the Master and PhD degree in Software Engineering. Dr. Abedallah is passionate about the managerial process of software development, coding themes, Big Data, Blockchain and Data Science. His research interests include software functional size measurement, software functional measures conversion, cost estimation, empirical software engineering, database, Big Data and Data Science. Currently, he is working as an assistant professor at college of computer information technology, American University in the Emirates, Dubai, UAE. Dr. Abedallah has published several high reputable refereed papers in high impact factor journals and international conferences. He is serving the scientific community as a regular reviewer for several high impact journals. In addition, he is working as a consultant for regional software development company to prepare accurate estimation for project deliverables. He can be contacted at email: azasoft1@gmail.com.

Najla Matti Isaacc is currently a lecturer at College of Agriculture and Forestry/Branch of Basic Science, University of Mosul, Iraq. She obtained his Bachelor and master’s degrees in Computer Science from University of Mosul. Her research interests include Pattern Recognition, Image Processing, Artificial Intelligence, Neural Network, Image Watermarking, and Cryptography. She can be contacted at email: najla.matti@uomosul.edu.iq.

Ali A. Alwan is currently an assistant professor at School of Theoretical and Applied Science, Ramapo College of New Jersey, United States. He received his Master of Computer Science in 2009 and Ph.D. in Computer Science in 2013 from Universiti Putra Malaysia (UPM), Malaysia. His research interests include databases (mobile, distributed and parallel), preference queries, web databases, probabilistic, incomplete and uncertain databases, query processing and optimization, data management, data integration, location-based social networks (LBSN), recommendation system, data mining, database in Cloud, Big data management, and crowd-sourced database. He can be contacted at email: aaljuboo@ramapo.edu.

Yonis Gulzar is currently an assistant professor at King Faisal University (KFU), Saudi Arabia. Before joining KFU, he was a part-time lecturer, teaching assistant as well as a research assistant in the Department of Computer Science at International Islamic University, Malaysia. He obtained his Ph.D. in Computer Science in 2018 from International Islamic University Malaysia. He received his Master in Computer Science in 2013 from Bangalore University, India. His research interests include preference queries, skyline queries, probabilistic and uncertain databases, query processing and optimisation and management of incomplete data, data integration, location-based social networks (LBSN), recommendation systems, and data management in cloud computing. He can be contacted at email: ygulzar@kfu.edu.sa.