International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

SecureIT- Android Security App

Anushka Yadav
IT dept. SFIT
SFIT
Mumbai, India

Shriya Wade
IT dept. SFIT
SFIT
Mumbai, India

Mihir Kanade
IT dept. SFIT
SFIT
Mumbai, India

Giselle Coutinho
IT dept. SFIT
Mumbai, India

Dr. Minal Lopes
IT dept. SFIT
Mumbai, India

***Abstract*—Various anti-malware operating system suffers from serious flaws across the Android device. Due to the limitations of these devices, you will not be able to access or monitor the dynamic behaviour of your Android device's file system or installed apps. This includes downloading harmful application after installation. In this proposed methodology, the Android Security App will consist of 3 different features: 1. Internal Scans, 2. Detect Malware, 3. Boosting the RAM. In internal scan the user will get to know if any pre-installed apps are malicious or not. The pre-installed app will get scanned automatically. This feature is used for direct blocking the malicious apps. The next feature is detection of malware. Detect malware is based on signature detection. It will detect the app's signature and signatures stored in database. If the signatures match with each other, then the app will be detected as malicious. The Android Security App will have an option through which it can clear the cache data of all the applications existing on the Android device. This clearing of cache data will thus result in boosting the RAM of the device.**

***Index Terms*—Android, Security, Signatures, Cache data, Malware, RAM.**

I. INTRODUCTION

Cyber-security is used as shield to Internet-connected systems from digital threats, including equipment, operating system, and information. It is composed of two terms, 'cyber' and security'. Cyber refers to mechanization of systems, networks, and strategy or data. Security, on the other hand, is concerned with protection, which encompasses system security, network security, and application and information security. It is a collection of technologies, methods, and practises meant to safeguard networks, devices, programmes, and data against assault, theft, damage, alteration, or illegal access. Android is one of the leading operating systems today and yet the concerns of its security keep increasing along with the newly introduced viruses and techniques that could be used for causing harm to the system and its use The existing antiviruses work on pre-stored digital signatures of the viruses. Thus, when new viruses are introduced, the anti-viruses cannot detect them easily and then these viruses pose a great threat to the android device and its use: Operating system flaws produce vulnerabilities that can be exploited. Patches are used by vendors to try to remedy these issues. Malware for mobile devices has been steadily increasing. The emphasis is on destroying files and causing havoc. The amount of identity thefts through various malware embedded into various applications or software is increasing day by day, thus, causing harm to the user. There is a need for a software or application that can detect such viruses and notify the user about the threat in the app/software that can harm the user or the android device. As newer viruses arise day by day, there is also an increasing need for safety for the device as some anti-viruses might not be able to cope up with the increasing threat. [1]

II. LITERATURE REVIEW

In paper [1], the Android working framework uses a consent-based model. This allows Android applications to access customer data, framework data, gadget data, and external assets from your smartphone. The designer must announce the consent of the Android application. Customers need to confirm these approvals in order to effectively deploy Android applications. These approvals are confirmation. During setup, the application can access assets and data at any time, assuming that the permissions have been approved by the customer. There is no need to regain consent. Android operating systems are at the mercy of various security attacks due to security flaws. In this white paper, you can share information such as two-factor authentication due to improper and improper use of app authentication using spyware, information theft in Android apps, security vulnerabilities or attacks on Android, Android investigation, etc. Report on abuse of app consent using Android ID. From a security perspective, iOS and Windows working frameworks.

In paper [2,] the increasing usage of Android applications (apps) has put beginner users at risk of unwanted access to private data. Due to platform constraints, even antivirus software cannot access or monitor an Android device's file system or the dynamic behaviour of installed apps. This has serious implications for device security because any app may still download and execute dangerous files without being identified by the user. We propose and construct a Runtime detection and prevention system in this study. The suggested solution would provide a second layer of protection by locking apps that attempt to access device resources without the user's knowledge. Our testing results reveal that this new System is capable of overcoming all of the attacks by the apps to gather sensitive information, with small impacts on the utility of legitimate apps and the performance of the OS.

In paper [3], they report the main precise review on the Android refreshing component, zeroing in on its Package Management Service (PMS). They exploration uncovered another sort of

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

safety basic weaknesses, hit Pile Up blemishes, through which a malevolent application can decisively pronounce a bunch of honours and properties on a low-variant working framework (OS) and delay until it is moved up to heighten its honours on the new framework. In particular, they observed that by taking advantage of the Pileup weaknesses, the application cannot just obtain a bunch of recently added framework and mark authorizations yet additionally decide their settings (e.g., insurance levels), and it can additionally fill in for new framework applications, debase their information (e.g., cache, treats of Android default program) to take delicate client data or change security designs, and forestall establishment of basic framework administrations. Their examination additionally distinguished many adventures open doors the enemy can use more than huge number of gadgets across various gadget producers, transporters and nations. To relieve this danger without imperilling client information and applications during an overhaul, we additionally fostered another discovery administration, called SecUP, which sends a scanner on the client's gadget to catch the noxious applications intended to take advantage of Pileup weaknesses, in light of the weakness related data consequently gathered from recently delivered Android OS pictures.

In paper [4], Hackers employ malware to get access to target systems, according to paper [4]. Malicious payloads are often created with tools like Metasploit. In order to identify these malicious payloads and safeguard the victim PCs, the target computers use anti-virus programmes. As a result, hackers developed anti-virus evasion technologies to avoid detection by these antivirus programmes. How successful are these antivirus evasion methods, though? This study compares the effectiveness of many anti-virus evasion technologies, including Avet, Veil 3.0, The Fat Rat, PeCloak.py, Phantom-Evasion, Shellter, Unicorn, and Hercules, against the finest anti-virus solutions available on Windows and Android platforms.

In paper [5], the authors presented a straightforward yet effective malware detection algorithm that uses a subset of Android APIs as classification features. Then, for a specific suspicious app, we compute the total of inverse values of the rankings of the benign APIs utilised, as well as the sum of inverse values of the rankings of the malicious APIs used. The benign API list covers the most often used APIs by benign apps, whereas the harmful API list comprises the most frequently invoked APIs by malicious apps. More specifically, we find that a given app is benign if the total of inverse values based on benign applications is greater than the sum of inverse values based on hazardous apps. According to the experimental results, the suggested approach obtains an accuracy of 87.35 percent 89.93 percent for malware detection in android.

In paper [6], Android malware is becoming increasingly frequent these days, owing to the fact that applications are not generated by reliable sources. People enter personal information, save cards, and do other things in the hopes that these applications would keep them fit or remind them of important tasks that we often overlook in our hectic lives. In such circumstances, identifying malware before installing a programme would be quite beneficial. It may even deter a few criminals. We propose in this research to employ a fully linked deep learning model to identify Android malware. The proposed work's key characteristics are detection of

Android malware before installation, the identity of the Android malware, and version packages with demonstrated exceptionally high accuracy of about 94.65 percent. This model also learns all features from all feature combinations. It entails substantial investigation and testing in order to reach extremely high precision.

In paper [7], The authors aim to investigate the performance of many machine-learning algorithms, including Naive Bayes, J48, Random Forest, Multi-class classifier, and multi-layer perception. For regular apps, Google Play store app data from 2015 and 2016 is used, and for evaluation typical data sets of malwares are used. To show the accuracy of classification, multi-class classifiers were shown to surpass the other techniques. In terms of model development time, the Naive Bayes classifier outperformed.

In paper [8], to deal with the significant growth in the amount of Android malware, the authors have developed SIGPID, which is a malware detection system based on permission usage analysis. Instead of obtaining and analysing all Android permissions, we provide three stages of pruning by mining permission data to determine the most significant permissions that may be used to discriminate between benign and malicious apps. SIGPID then use machine-learning-based classification approaches to identify various malware and benign app families. According to our analysis, just 22 permissions are important. The performance of the technique used by authors, which uses just 22 permissions, is then compared to an approach that examines all permissions known as baseline approach.

III. METHODOLOGY

A. Architectural Design

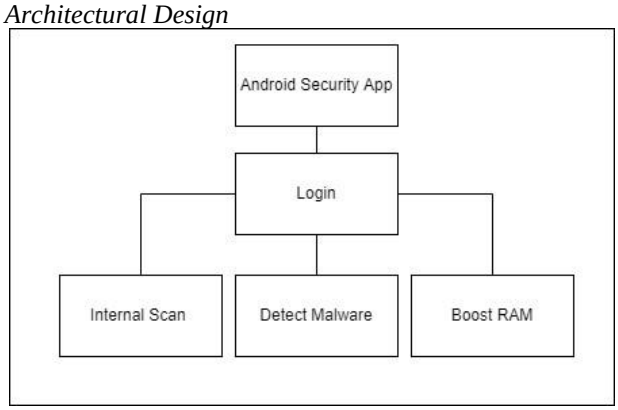

Fig. 1. Architectural design. Figure 1 shows the architectural design of the Secure-it application. The features of Android Security App are as follows:

- Internal Scan: After the user logs in into his/her personal account, user can perform internal scans. The pre-installed app will get scanned automatically. This feature is used for direct blocking the malicious apps. The list of installed apps will be scanned and if the app is malicious, it will be listed in blacklist. If the app is not malicious it will be notes as "Not in blacklist". Separate list of blacklisted apps will be shown to user in 'view blacklist' feature.
- Detect Malware: Detect malware is based on signature detection. It will detect the app's signature and signatures stored in database. If the signatures match with each other, then the app will be detected as malicious and user will get to know about it. If the signature does not match with any

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

signatures stored in database, then the app is not malicious and is safe for user to use.

- Boosting the RAM: After the user logs in into his/her personal account, the user can use the Boost RAM feature. The user needs to click on the BOOST RAM button to start the process. Once the button is clicked, the Security App will clear the cache data of all the apps present in the android device.

B. Methodology

For the scanning of apps signature-based detection method is used. This method can be divided into two steps, signature detection and the hashing.

The steps are explained below:

1. Signature Based Detection: Examining based on signatures entails comparing a collection of signatures to a given file. In general, it is a challenge for pattern matching on several levels. The collection of *N* signatures is given,

S = {Si} of lengths L = {Li}, 1< i < N, an input sequence of bytes F = (f1f2 ...fm), 0 < fi < 255, 1 < i < m. Now we must determine if position x exists such that:

$0 <= x <= (m - Lq + 1)$,

where *m* is the file size of the input file and *Lq* is the shortest signature's length in *S* and *x* denotes the beginning of a sub-sequence in *F* that exactly matches at least one signature in *S*.

The basic approach in relation to problem of pattern matching on several levels entails comparing each pattern to the one before it. A file of length *m* must be checked against *N* signatures, each of length *n* bytes, the worst-case running time is $O(N * m * n)$ which may be very inefficient. The KMP (Knuth-Morris-Pratt) method [10,11], which is based on dynamic programming, produces higher results for single pattern matching [9]. The Boyer-Moore method [12] is the most well-known approach for developing a single pattern matching solution.

Several strategies have been put forward to improve and widen the Boyer-Moore algorithm for pattern matching on several levels [13,14]. The Boyer-Moore algorithm-based methods maintain extra tables during the pre-processing step on the pattern set. This requires large memory other than the storage space which is consumed by patterns themselves. This is one of the weaknesses of Boyer-Moore algorithm. Similar to Boyer-Moore, another method that needs the memory storage of entire signatures in structure form during matching is the Aho-Corasick method. This method is based on finite automata. Hence, it may require more memory space on android device if the signatures to be matched are more and large. Another algorithm is Rabin-karp which currently uses hash approach for pattern matching at single level which can be improved to pattern matching at several levels with a little effort. Even though the worst time complexity of this algorithm is similar to that of the naive pattern matching algorithm, it gives much better results.

All the above-mentioned algorithm are useful for scanning purpose and may be used to solve any string-matching problem. Since, the solution which we are providing is focusing only on detection based on signature matching for android device, we make advantage of the finding that particular bytes are less likely than others to appear in non-malicious Android applications. By doing so, we can limit the number of signature comparisons.

As shown in Fig 2, the signatures of viruses are stored in database. The signature of app is extracted. If the signatures match with each other than the app is malicious, else it is safe to use the app.

2. The hash table: The most basic method for comparing a given file in opposition to *N* signatures is to compare each subsequence in the input file one by one. The above-mentioned method is extremely ineffective in terms of computation. Creating the hash table is a well-known technique for speeding up pattern matching. That being the case, the hash table contains each and every signature at a different index. The signature is used to calculate this index value. The index of any input sub-sequence that links to a hash table item is calculated. If each entry of the hash table has precisely one signature,

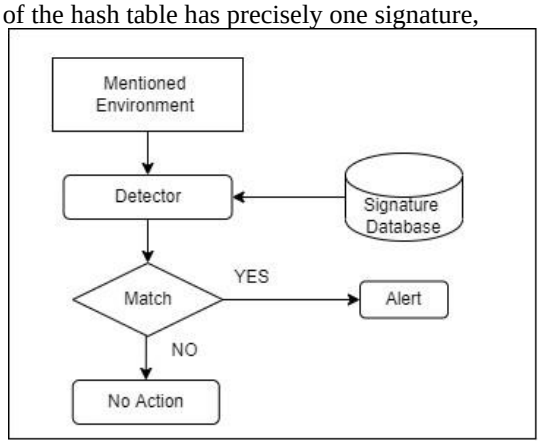

Fig. 2. Signature Based Detection

Fig 2 explains the working of signature-based detection method.

we know we only need to check each input succession in opposition to one signature instead of all *N* signatures. Amongst the most important aspects of creating the hash table is having an effectual/easy function, where the sequence is given and it calculates the hash table index for the sequence. The index for signatures only needs to be computed once, during the hash table construction. The index, on the other hand, must be computed for each succession in the given file. As a result, it is critical that the index be computed efficiently. In this scenario, we utilise a basic approach called a filter-hash to generate a hash value for a sequence of bytes, and then use that value to retrieve the index in the hash table.

For any byte sequence $F = (f1f2...fm)$, $0 <= fi <= 255$, $1 <= i <= m$, a continuous succession of *L* bytes beginning at position *j* is used to calculate a filter-hash value as [16]:

$$H(F, j, L') = \sum_{i=1}^{L'} (i * fi + j - 1) \quad (1)$$

Eq. (1) is used to generate the index for sequence F from the filter-hash value as follows:

$$H(H(F, j, L')) = mod(H(F, j, L'), Q), \quad (2)$$

Where *Q* denotes the hash table's dimensions. The filter hash cost has the important property of being able to be computed recursively for successive sub-sequences of a very uncommon place length for an entry file. The evidence supporting this is as follows: To begin with,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

$$H(F, 1, L') = \sum_{i=1}^{L'}(i * fi) \quad (3)$$

and the total of the first L consecutive bytes be,

$$Hs(F, 1, L') = \sum_{i=1}^{L'}(fi) \quad (4)$$

It is clear that:

$$H(F, j+1, L') - H(F, j, L') = \sum_{i=1}^{L'}(i * fi + j) - \sum_{i=1}^{L'}(i * fi + j - 1) \quad (5)$$

Simplifying equation (5) we have,

$$H(F, j+1, L') = H(F, j, L') + L' * fL' + j - Hs(F, j, L') \quad (6)$$

$$Hs(F, j, L') = Hs(F, j, L') - fj + fj + L' \quad (7)$$

As a result, using equations (6) and (7), we can read the filter hash values and, eventually, the hash desk index of successive sub-sequences of length L in the given file. The size of the hash desk, Q, is an important parameter in a hash desk. The length of the hash table should be adequate such that each signature may fill a single space on the table. This length is chosen to be more than the total number of signatures, N. As a result, there are Q spaces in the hash table, of which N are occupied with signatures and $Q - N$ are vacant. Having a bigger wide variety of empty slots impacts computational time when you consider that this will suggest that no signature matching is needed whilst a sub-sequence hash indexes an empty slot. On the alternative hand a completely massive hash desk also can boom computational time when you consider that we might require greater time for reminiscence get entry to within the desk. Hence, we run experiments with one-of a-kind hash desk sizes as a way to discover the scale which minimizes computational time.

IV. RESULTS

A. UI design

Fig. 3. Main page

Fig 3 represents the main page of the app its developed using Android Studio Version 2020.1.3. It will serve as the user interface for the developed application. Once the user opens the app he/she will be directed to this main page. On the main page the user has to authenticate herself/himself this will be done on the login page as shown in Fig 4.

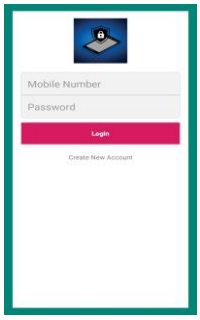

Fig. 4. Login page

Assuming the user has registered or already has an account he/she can go to the login page as shown in Fig 4. On this page, the user will be asked for the ID and Password. He/she will be redirected to the sign-up page in-case he/she doesn't have an account yet.

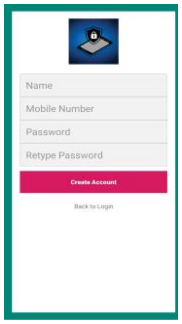

Fig. 5. Registration page

Once the user clicks on the sign-up option, the user will be redirected to this sign-up page as shown in Fig 5. Here the user will have to fill some details to setup an account.

Fig. 6. Home page

Fig 6 represents the home page. After the user is successfully logged in, the user will be redirected to the home page. Here, the user will be provided with multiple different options.

Fig. 7. Internal Scan

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

In Fig 7 the user will get the list of apps which are pre-installed in android mobile and are not malicious.

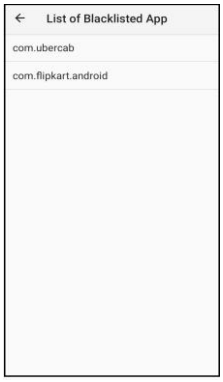

Fig. 8. Blacklisted Apps

In Fig 8 the apps which are malicious will be listed.

Fig. 9. Malware Detection

In the Fig 9 the apps will be scanned using the signatures and will inform user about the malicious apps.

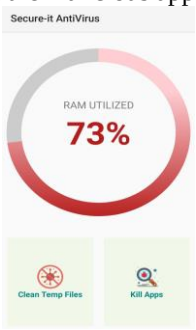

Fig. 10. Ram Utilized

Fig 10 shows when the user clicks on ram booster option, he/she will be redirected to this page. Here ram utilization will be displayed. User will get two options,' clean temporary files' and 'kill apps' which will clear the cache memory.

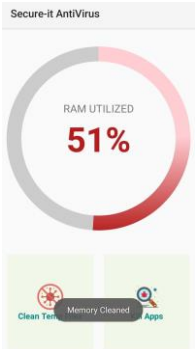

Fig. 11. Ram Booster

Once the user clicks on kill apps option the ram will be cleared. The user will be notified through pop-up message.

CONCLUSION

Android is widely used operating system. The security of android device is very important with respect to the user's privacy. The Secure-it app is created to secure the android device. This application provides three features internal scan, detect malware and boost ram. This app will scan the preinstalled apps and will display user about the malicious content. If the app is malicious, it will be listed into blacklist. Also, the boost ram feature is provided which will clear the cache memory, temporary files and close the running apps.

This application scans the apps which are already installed. If new app is to be downloaded the application will not notify the user. This application can be further improved by scanning the newly downloaded apps. To cope up with the possibility of newer viruses being introduced, AI and ML techniques will be used by the app to detect such viruses. Through this approach, antivirus software on the Android platform would reach a level of effectiveness significantly higher and comparable to that of desktop antivirus software.

REFERENCES

[1] Android Security Issues and Solutions(S. Karthick and S. Binu, 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, 2017, pp. 686-689, doi: 10.1109/ICIMIA.2017.7975551.)

[2] Dar, M. A. (2017). A novel approach to enhance the security of android based smartphones. 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS). doi:10.1109/iciiecs.2017.8275923

[3] L. Xing, X. Pan, R. Wang, K. Yuan and X. Wang, "Upgrading Your Android, Elevating My Malware: Privilege Escalation through Mobile OS Updating," 2014 IEEE Symposium on Security and Privacy, 2014, pp. 393-408, doi: 10.1109/SP.2014.32.

[4] Garba, F. A., Kunya, K. I., Ibrahim, S. A., Isa, A. B., Muhammad, K. M., and Wali, N. N. (2019). Evaluating the State of the Art Antivirus Evasion Tools on Windows and Android Platform. 2019 2nd International Conference of the IEEE Nigeria Computer Chapter (Nigeria Compute Conf). doi:10.1109/nigeriacomputconf45974.2019.8949637

[5] Jung, J., Lim, K., Kim, B., Cho, S., Han, S., and Suh, K. (2019). Detecting Malicious Android Apps using the Popularity and Relations of APIs. 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE). doi:10.1109/aike.2019.00062

[6] S. HR, "Static Analysis of Android Malware Detection using Deep Learning," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), 2019, pp. 841-845, doi: 10.1109/ICCS45141.2019.9065765

[7] P. R. K. Varma, K. P. Raj and K. V. S. Raju, "Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 294-299, doi: 10.1109/I-SMAC.2017.8058358.

[8] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," in IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, July 2018, doi: 10.1109/TII.2017.2789219.

[9] R.M. Karp and M.O. Rabin, Efficient randomized pattern-matching algorithms, IBM Journal of Research and Development 31(2) (1987), 249–260.

[10] T.H. Cormen, C.E. Leiserson and R.L. Rivest, Introduction To Algorithms, McGraw-Hill, 2002, 966–1057.

[11] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast Pattern Matching in Strings, SIAM Journal on Computing 6(2) (1977), 323–350.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 11 Issue 05, May-2022

[12] R.S. Boyer and J. Strother Moore, A fast string searching algorithm, Communications of the ACM 20(10) (1977), 762–772.
[13] B. Commentz-Walter, A string matching algorithm fast on the average, in: Proceedings of the 6th International Colloquium on Automata, Languages and Programming. LNCS, vol. 71, Springer-Verlag, Berlin, 1979, 118–132.
[14] S. Wu and U. Manber, Agrep – a fast approximate pattern-matching tool, in: Proceedings of the Usenix Winter 1992 Technical Conference, 1992, 153–162.
[15] A.V. Aho and M.J. Corasick, Efficient string matching: An aid to bibliographic search, Communications of the ACM 18(6) (1975), 333–340.
[16] Venugopal, Deepak Hu, Guoning. (2008). Efficient Signature Based Malware Detection on Mobile Devices. Mobile Information Systems. 4. 33-49. 10.1155/2008/712353.