Privacy Protection Framework for Android

Aiswarya K Shaji
(ASI19CS014)
Hani HAridas P
(ASI19CS054)
Jannath Parvin
(ASI19CS061)

Guide :Prof Sabitha MG

Abstract- Numerous creative and clever Android applications have emerged as a result of the Android platform's recent growth in popularity and users (apps). Numerous of these applications are extremely interactive, programmable, and dependent on user data to function. While this is practical, user privacy is the main issue. It is not assured that these applications don't break algorithms or keep user data for their own use. Android secures and safeguards user data through a permissions mechanism. During installation or runtime, the user can grant permissions to the necessary resources. However, in fact, this mechanism is frequently misused by requesting extra authorizations that are not necessary for service provision. If they are not updated, these apps will stop functioning.If these apps don't get all the permissions they need, they will stop functioning. Consequently, a safe privacy framework is created in this document to stop the program from stealing user data by limiting any pointless rights. By forecasting the permissions needed by a specific application utilizing collaborative filtering and mining algorithms with frequent permission settings, unnecessary permissions are found. In order to alter the permission data within the target application, the proposed model interacts with it. The results of the experiments demonstrate that the suggested approach not only ensures the correct operation of the application but also safeguards user data.

Keywords: instrumentation, authorization model, security, and privacy in relation to Android.

I. INTRODUCTION

Android is the most popular operating system (OS). reaching mobile platforms According to figures from throughout the world, iOS had a 25% market share in the mobile OS industry in June 2020, while Android OS held close to 75% of the market. Users favor Android because it is free and open-source and supports a wide variety of applications. Because Android is open-source, developers favor it over the competing iOS. Applications are what Android apps, which are mostly built in Java, are known as. A crucial component of applications is security. Because of the nature of Android applications, it is challenging to rely on traditional, static, and dynamic systems for malware research. Google developed a programme to improve the security of Google Play applications in order to provide security services to Google Play application developers for the security of their applications. Applications are screened for potential viruses before they are released to the Play Store.In order to increase device security and Play Store usage, Google tried to detect malware and potentially hazardous applications in 2017. Play Protect by Google. To prevent background apps from having access to the camera, microphone, and sensors, Google limited the access of background sensors in Android 9. Google has included a feature for hardware-backed up keys to protect user information and passwords. Additionally, a Safe Browsing app programming interface (API) is available for defense against misleading websites.Some apps that collect user data may still be hazardous, despite considerable advancements in platform security, application development security, and a safe Android operating system.It might be difficult to distinguish whether apps in the Google Play Store are dangerous or may gather user data for the purpose of analysing their behaviour or selling it to a third party because there are already over 2.7 million of them available. Android relies on application permission to prevent data security issues and malicious use, which means that applications must request the rights required for their applications from the user and that Android only provides access to the APIs for that permission if the user approves it. Normal, hazardous, and signature permissions are the security level categories for Android permissions.Just the necessary permissions must be mentioned by the developer in the manifest file. Users are not asked for these permissions by Android. Unsafe permissions may cause significant concerns with data breaches. Therefore, the user should be prompted to grant the program permission to use these permissions within the application. These permissions should be listed in the manifest file. allowed users to utilize the program without actually giving any permissions access. The data leakage issue was still unresolved, though, as several programs began to break when the user refused to provide permission. As was already mentioned, the mission methods used by the Android OS platform. Some applications demand a lot of permissions. These permissions were created with the intention of examining Android's current security procedures, user privacy issues, and data trading.Many of the applications have been examined, and it was discovered that the developer was utilizing the user's permission to steal their private and personal information. 6.0 Android (Marshmallow) Android offers allow/deny permission support, allowing users to choose which rights an app needs and to refuse any that they deem unnecessary. However, if particular dangerous permissions are disallowed, this frequently results in apps that the developer deliberately crashed. Due to the lack of security in dangerous permissions linked to sensitive APIs, rogue programmes violate user privacy by controlling users and application services.This circumstance inspired us to develop a solution that would enhance the application's usefulness while guaranteeing responsive data. This document suggests a data protection system that will guarantee data security and work to prevent malicious crashes caused by applications that mistakenly assume they have access to the necessary data. This article addresses how to safeguard data privacy and prevent application faults when permissions are denied.Due to all of these problems, the service is constrained to look for potentially harmful permissions inside the installed programmes on the user's device.The suggested framework offers services that anticipate the permissions needed by the application and give instructions on how to prevent failure

dynamically. Installing an Android Package (APK) file on the target device is fundamental.Any time a potentially harmful API is used, it interacts with the server service in real time. The mobile client uses an unprotected public channel for communication (the Internet).. As a result, schemes of key agreements and anonymous authentication are also intended to protect communication without disclosing the identity of the client. As a result, the suggested framework safeguards user data without compromising the functionality of the application. The following sections make up the post's structure. In Part II, we talk over Android's security features. Part V deals with the suggested model's application through a case study. The planned framework and communication plan, as well as the justification and earlier work in this field, are all explained in Sections III and IV.Section VII is a list of the conclusions and upcoming theses.

II. SECURITY ON ANDROID

(a) ANDROID ARCHITECTURE:

An open-source software programme based on the Linux OS is called Android (OS). The OS design, shown in Fig. 1, which separates resources and availability in later layers, is the foundation for platform security. Every layer considers that the layer below it is safe. Later layers are harder to access. The Linux kernel, which operates at the most fundamental level, is in charge of carrying out fundamental operating system functions like managing processes, memory, and other device-related tasks. Linux is run on top of the Hardware Abstraction kernel layer. the layer (HAL) is charge of giving hardware manufacturers a uniform interface so they can design their own hardware without interfering with upper layers.Now it is possible to upgrade the OS without having to modify or reconfigure the hardware. Additionally, it gives HAL an advantage. By preventing HAL in a process from having access to the same permission list as the rest of the process, it upholds the idea of least privilege. Android offers cryptographic services with hardware-backed key protection. A security channel for user data authentication is provided by stored keys. Verified Boot is used to examine the system's condition at startup. It validates the soundness of the system. Google released Project Treble with Android 8.0 (Oreo) to improve low-level security. Project Treble separates vendor-level hardware code implementation from the open-source Android OS. It improved the security of the devices and the speed of updates.

(b) APPLICATION SECURITY:

Android utilizes a permission scheme to stop apps from accessing private information and resources that are not necessary for them to function. To communicate with the underlying system using an API, applications must have the necessary rights. The application manifest file must provide a list of all the permissions the application has been granted. The three categories of authorizations—normal, risky, and signature permissions—reflect the varying degrees of risk and security attached to the resources and API. Normal permissions include those that are required for apps to communicate with resources outside of the sandbox but don't compromise user privacy.Bluetooth, KILL BACKGROUND _PROCESS, and the Internet all require standard permissions. When an app is installed, signing permissions are granted, enabling it to use signed permissions that are identical to certificates. The permissions for signatures contain VPN SERVICE. Permissions that potentially jeopardize user privacy or provide a security risk are considered dangerous. Each of these rights that the program requires after installation must be approved by the user. Some of the permissions in this category include SMS, storage, and camera missions.To protect consumers' phone data, Android has rigorous regulations for critical APIs. In Android 8.0 and later, GET ACCOUNTS is no longer sufficient to give full access to the list of active accounts on the device..For instance, even though Google owns Gmail, the user must now provide permission for the Gmail app to access your Google account on your device. The ID entered is Settings, for example. Secure. SSAID or ANDROID ID should be used by all applications. In order to prevent misuse of the ANDROID ID variable, Android 8.0 contains a system that waits enabling the ANDROID ID to change when reinstalling the app until the package name and key match. Build, additional features. Until the caller obtains permission from the PHONE, getSerial () returns the actual serial number device. In order to prevent apps from misusing the serial device number, Android 8.0 deprecated the use of this API.The security features of Android have improved. However, it should be mentioned that as of October 2020, just 40.35 percent and 22.5 percent, respectively, of devices run Android 10.0 and Android 9.0, respectively (Pie). More than 35% of users have outdated Android phone versions. Users are frequently tricked into using excessively privileged programmes because of a lack of user awareness and a lack of security in sensitive APIs.

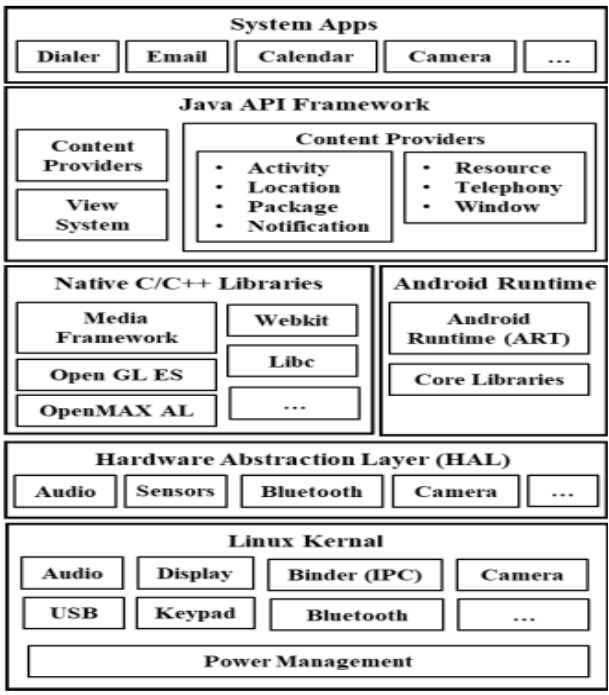

FIGURE 1. Android platform architecture.

III. RELATED WORK

Malicious applications have also emerged as Android usage has increased, and numerous studies and investigations have been conducted. An extensive analysis of the Android permission mechanism was proposed by Iman and Aala [4]. They offered

important insights into how the permission system evolved over time and how, by 2020, authority utilization in the most popular applications had increased to 73.33%. Machine learning (ML) algorithms that extract Android permissions from applications were suggested by Sants et al. [11] as a way to detect fraudulent Android applications. The Android permissions machine learning model was used to categorize the application as malware based on the permissions that were extracted from the app's Android manifest file.Mr. Karim Al Using collaborative filtering techniques, associative rule mining, and Bayesian text mining, we propose permissions for Android applications. By mapping 4243 to a related application, I attempted to estimate what permissions an application would want in order to use it. It's intended to assist developers in determining the kind of permissions that their app requires. User opinions are not included. A malware detection framework for Android called NATICUSdroid was presented by Mathur et al. [7]. Based on statistically chosen native and modified Android permissions, it investigated benign malware and categorized them as features of several ML classifiers.These methods, however, could only analyze and classify data using mobile resources. These methods also lacked learning skills and dynamic processing, which made it impossible for them to halt these malicious operations within the programme and limited their success. Azim and Neamtiu [6] used static data flow analysis of the app's bytecode to evaluate Android apps systematically and discovered that there were connections between different app activities. constructed a high-level control flow diagram. They have developed a technique for creating deep flows between various tasks that mimic the user's movements. Although it had great potential and served as the foundation for dynamic analysis, this method was still unable to train by itself. He conducted the analysis using mobile computer capabilities, however his methodology was flawed.For Android apps, Ricardo et al. [10] developed a framework that gives the app injection to monitor harmful activity. The foundation of the proposed study is this method, which employs dynamic analysis. Even with this strategy, however, it was unable to apply past outcomes, therefore there was no app-based learning. The amount of apps that need to be sandboxed to determine whether they are dangerous is reduced thanks to the method presented by Sharia et al. [9]. In order to identify malware programmes, they employed Latent Semantic Indexing (LSI), however this was only effective in this context.Work on the Terminator Framework is presented by Sadegi et al. [5], which can provide efficient yet continuous defence against authorization-induced assaults by locating a secure state of the system and regulating authorization based on it. did. Without altering the implementation logic of the programme, permissions were granted and those that were deemed hazardous were removed.

IV COMPONENTS

(a) ANALYSIS AND INSTRUMENTATION OF THE APK:

The resources and permissions that the application needs are used to analyse it. It is done to analyse permissions in relation to API calls and application utility. In order to determine whether the programme needs further rights in order to steal user data, the permit suggestions are employed. Permission set mining and cooperative filtering algorithms are frequently employed for this purpose. The data used to create the training set was gathered for several applications across various categories.Instrumentation tools that support APK tools to support the function to call the When the findings from the two algorithms are combined, Data Protection Service sends the result, which includes dangerous and special authorization.Both algorithms rely on the classification of an app's permissions depending on the Play Store category it belongs to and the data collection that has been obtained.

(b) DATA PROTECTION SERVICE:

With the help of a broadcast receiver and the Android framework, users can register for events analytically and dynamically. The lifespan of a dynamic application component depends on whether it has enabled Context.registerReceiver () and Context.unregisterReceiver (). Static receivers have the same lifetime as the application and are defined in AndroidManifest.xml. To override the SDK call, the receiver employs a callback method, namely BroadcastReceiver.onReceive ().

1) DATA COLLECTION:

There are two parts to the data assortment strategy. Through the event and growth one information assortment application, the initial information assortment is completed. About 300 users have downloaded this app, and through that, information on 1,000 different programmes has been gathered. The information was then checked for permissions, permissions granted by the user were retrieved, and the 0.5 probability rule was used to determine if a privilege is beneficial or detrimental. As a result, unique application information is added to the information whenever the algorithms that execute on the server provide value to it. This might help the dataset get better and the proposed framework develop over time.

2) ANALYSIS AND INSTRUMENTATION:

The server's engine is operating in order to look into and evaluate APKs (Fig. 3). Decompiling the app exploitation Apktool, which is used to reverse-engineer automaton apps, is the initial stage. The decompiled programme is run in assembly code called Smali by the Dalvik Virtual Machine, which is a component of Android's Java Virtual Machine. The parser and instrumentation engine processes the decompiled code in the following steps. The application is afterwards repackaged using Apktool. Information Protection Service loads it onto the user's automated phone.

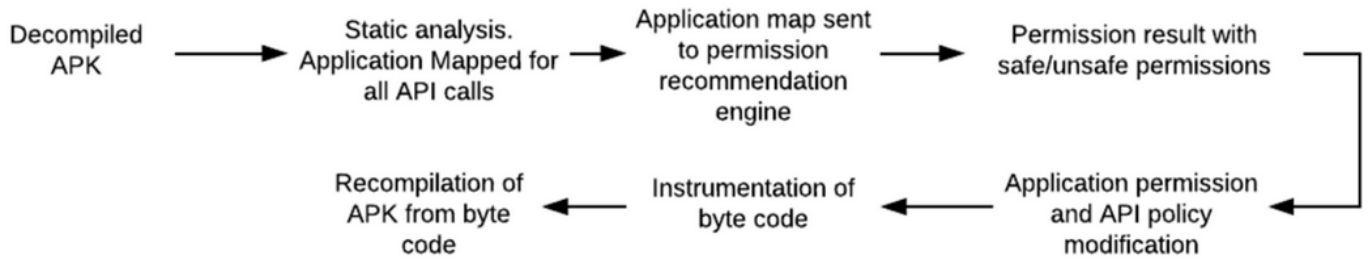

FIGURE 3. **The flow of instrumentation engine.**

PERMISSION ANALYSIS:

Cooperative filtering and regular permission set mining are the two methods that were most frequently used to determine the permissions that an app requires. The coaching package includes apps from every category available in the Google Play store.

COLLABORATIVE FILTERING:

(i) Finding the feature vector within the projected engine:
The collaborative filtering engine uses feature vectors based on the app permissionsAn application's permissions are retrieved as an enormously long vector, V = P1, P2,..., Pn >, where Pi might take values between [0, 1] depending on whether the programme accepts the given permission.We extract feature vectors from the coaching information set's apps. The engine first pulls the applications from the same class as the examination app.Then, for filtering and recommendation, all of the permissions are retrieved from the feature vectors. Ai = AP1, AP2, AP3,..., and APn where APi gets the value from the set [0, 1].

(ii) Evaluation of similarity:
A similarity score can indicate how closely related two things are. The Jaccard similarity score is used to determine how similar an app is to all other apps in its class: S (Ai, At) = F11 / (F01 + F10 + F11).
Ai is the application from the same-class information set. The test app can be found at. F11 is the frequency of matches between Ai and At's permissions. In the case of Ai, F01 is the frequency of permissions one, while in the case of At, F01. The frequency of permissions at frequency 10 is one in the case of At and zero in the case of Ai.

(iii) Recommendation of Permissions
The app At generates a recommendation score for each permission request. Victimization could be determined as RScoreCF (Pi) = S (Ai, At). In this case, the vote of the majority is taken into consideration, with the weight of the vote proportionate to the similarity score obtained higher than. The RScore that is produced is normalised. if the app is recommended to use it, the permission is listed as safe; if not, it is considered unsafe.

FREQUENT PERMISSION SET MINING:

The second rule of suggestion is based on anticipating permission combine values that happen concurrently. The relationships and patterns of the simultaneous requests for authorization are investigated. Regarding the connection that two permissions have, permissions are advised for joint applications. Frequency is the foundation for the preliminary support estimate in event prediction. Support is used to identify connections between entities.Let's say that an event (event B) chosen from a dataset of N events occurs f times (frequency of event B).
Sup (B) = Frequency (B) / N is the formula for event B's support. Look at the project's permissions in the app. Depending on whether the app requests the permission, Pi will use values from the set [0, 1] at the square measured extracted in V = P1, P2,..., Pn >. The permissions of coaching knowledge apps are included in vectors with the formula Ai =, where APi selects values from the range [0, 1].

V. RESULT AND CASE STUDY

Nine privileges' behaviour is examined in this study. These rights (mentioned in Table 3) allow users to "read" their phone's stored user data. The suggested framework starts by searching her Smali code for these permissions and manifest files. Use Brightest Flashlight (golden-shores-technologies. Brightest-flashlight.free), Peacock Flashlight (com.peacock. Flashlight), and Flashlight to see the suggested framework in action (Output Comp. Apps. Flashlight). To function, each app needed a unique set of permissions. The Peacock flashlight asked for location (loc) and storage rights, the Brightest Flashlight app asked for a number of permissions, and the Spend Apps flashlight asked for none. These applications were submitted to the suggested framework in order to receive the barest amount of instrumentation and permissions. Decompiling the app produced Smali code for each APK. Android is indicated here by AP and ACC.

A. STATIC ANALYSIS

Static analysis is the first stage in this procedure. The permissions and classes declared in the AndroidManifest.xml and their corresponding methods from the Smali code are delivered via the engine's Smali parser. A map displaying method tracing and data flow is derived from the Python parser, as explained in section V of Analysis and Instrumentation. AP.WRITE EXTERNAL STORAGE and READ PHONE STATE. Similar code analyses of the two applications reveal that Peacock Flashlight has risky permissions while AP does not.

B. PERMISSION ANALYSIS

The permission recommendation algorithm receives permissions parsed as described in the preceding subsection as input. Each permission is assessed by the algorithm, which then generates result vectors. Each result vector has 9 entries, each of which can either be 0 or 1. If not, the value is zero. Run the Brightest Flashlight recommender with a 0.1 threshold for collaborative filtering. The generated vector looks like this:
rp denotes the resulting permissions and is equal to [0, 0, 0, 0, 0, 0, 0, 0, 0].
Each privilege's RScoreCF is below the threshold. As a result, none of his permissions are needed for this app, and all three of his permissions—AP.ACC FINE LOC, AP.ACC COARSE LOC, and AP.READ PHONE STATE—are considered unsafe. The support for each permission was estimated and graded against the average value using frequently used permission set mining as detailed in Section V. The result vector can be obtained as follows:
rp = [0, 0, 0, 1, 1, 0, 0, 0, 0].This is the Access Grant AP ACC FINE LOC; safe are AP.ACC COARSE LOC, but 'AP. The three extra rights that the brightest flashlight has to have are as follows: Permission Execution of AP.ACC FINE LOC and AP is advised for Peacock Flashlight. I found out you need the dangerous.ACC COARSE LOC permission. Since this app didn't need any additional permissions, Splendid Torch didn't perform a privilege analysis.

C. INSTRUMENTATION AND FINAL RESULTS

We discovered that the flashlight application's LOC and READ PHONE STATE permissions were vulnerable during the permissions analysis stage. Target devices were outfitted with instruments and installed with Brightest Flashlight and Peacock Flashlight. The instrumented app had runtime interactions with the background service. The programme received false location information, but following instrumentation, it was discovered to function as intended. Following the completion of the entire procedure, the suggested framework produced the following outcomes:
1) To function, a flashlight app needs access to the camera.
2) The two applications' remaining permission requests are deemed dangerous. Applications' results were added to the dataset for later use.

3) The application was repackaged and instrumented to restore its original functionality while safeguarding user information that might have been utilised for illicit purposes.

To study the activity patterns of three apps in the same category and utility, we looked at them. However, three similar apps worked differently since they were given various permissions that had nothing to do with the functionality that was actually listed. Two applications need new entitlements, according to the recommendations made by the entitlement recommender. Our measurements revealed that they had not changed in behaviour, which suggests that their functionality had not been impacted. The user's location was also shielded from his possibly harmful Android applications at the same time.

VI.CONCLUSION

Device security has become a significant concern as a result of the smartphone market's recent explosive growth and consumers' inclination to utilise them as personal data storage devices. The risk of data and privacy breaches rises as technology develops. To find dangerous behaviour in Android applications, several study approches have been provided. By limiting unneeded permissions with the use of instrumentation and programme repackaging, a secure privacy protection framework is presented to stop applications from stealing user data. These permissions were found by predicting the necessary permissions for a specific Android app using collaborative filtering and a common permission set mining technique. As a result, the target app is engaged in interaction with by the proposed model, which changes the authentication information it contains. The suggested framework has a security layer to keep hackers from listening in on conversations. As a result, the suggested framework outperforms rival models in terms of efficiency and security. Experimental findings demonstrate that the suggested model not only safeguards user data but also ensures the appropriate operation of the relevant applications. This strategy may not work well for sealed and protected apps, which often fall into the financial/payments category because they are constructed with more security. Therefore, following instrumentation, these programmes cannot be installed. The framework might change in the future to support further protections for the application.

REFERENCES

[1] Mobile Operating System Market Share Worldwide | StatCounter Global Stats. Accessed: Oct. 29, 2020. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Android Security 2017 Year in Review 2. Accessed: Oct. 17, 2020. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf

[3] K. W. Y. Au, Y. F. Zhou, Z. Huang, P. Gill, and D. Lie, "Short paper: A look at smartphone permission models," in Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, 2011, pp. 63–67, doi: 10.1145/2046614.2046626.

[4] I. M. Almomani and A. A. Khayer, "A comprehensive analysis of the Android permissions system," in IEEE Access, vol. 8, pp. 216671–216688, 2020, doi: 10.1109/ACCESS.2020.3041432.

[5] A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, and S. Malek, "A temporal permission analysis and enforcement framework for Android," in Proc. 40th Int. Conf. Softw. Eng., May 2018, pp. 846–857, doi: 10.1145/3180155.3180172.

[6] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of Android apps," ACM SIGPLAN Notices, vol. 48, no. 10, pp. 641–660, 2013, doi: 10.1145/2544173.2509549.

[7] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," J. Inf. Secur. Appl., vol. 58, May 2021, Art. no. 102696, doi: 10.1016/j.jisa.2020.102696.

[8] M. Y. Karim, H. Kagdi, and M. Di Penta, "Mining Android apps to recommend permissions," in Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng., Mar. 2016, pp. 427–437, doi: 10.1109/SANER.2016.74.

[9] H. Shahriar, M. Islam, and V. Clincy, "Android malware detection using permission analysis," in Proc. IEEE SOUTHEASTCON, Mar. 2017, pp. 1–6, doi: 10.1109/SECON.2017.7925347.

[10] R. Neisse, G. Steri, D. Geneiatakis, and I. N. Fovino, "A privacy enforcing framework for Android applications," Comput. Secur., vol. 62, pp. 257–277, Sep. 2016, doi: 10.1016/j.cose.2016.07.005.

[11] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "PUMA: Permission usage to detect malware in Android,"in International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions. Berlin, Germany: Springer, 2013, pp. 289–298, doi: