Automated security testing of Android applications for secure mobile development

[1]Francisco Palma, [1]Nuno Realista, [1]Carlos Serrão, [1,2]Luís Nunes, [1,2]João Oliveira, [1,3]Ana Almeida
[1]ISTAR - Information Sciences and Technologies and Architecture Research Center
[2]IT - Instituto de Telecomunicações
[3]CISUC - Centre for Informatics and Systems
ISCTE - Instituto Universitário de Lisboa
Lisboa, Portugal
Francisco_Livramento, Nuno_Realista, carlos.serrao, luis.nunes, joao.p.oliveira, ana.almeida{@iscte-iul.pt}

***Abstract*— Mobile computing is on the rise. More and more users rely on mobile applications and mobile devices to execute the most basic tasks of their lives while depositing their most private and critical data on them. Mobile application stores act as the trust anchors that sit between applications developed by third parties and the user's mobile devices. Therefore, app stores should provide the means to ensure that the apps installed by the users follow high security and quality standards, minimising the user's data exposure risks. A critical path towards that security and quality standards is to early test and detect mobile application vulnerabilities resulting from incorrect development practices and to provide developers feedback about the problems found and some additional information on how to correct them. This paper presents a system, developed to help Android app stores (in this case, on the Aptoide app store) timely detect vulnerabilities on submitted apps and provide appropriate feedback to developers. The provide feedback makes developers aware of the secure development processes while improving the quality and security of their apps before they are made available to end-users and installed on their devices.**

Keywords*— *vulnerabilities, android, mobile, security, tests, software, development, developers, feedback

I. Introduction

Mobile computing has achieved a level never seen before (estimates are that the number of smartphones will reach 6,1 billion by 2020)[1]. The two major mobile platforms (Android and iOS) completely dominate the market and users continue to adhere massively to these significant mobile platforms. Users are switching from more traditional data processing platforms (such as desktop computers) and embrace mobile platforms increasingly. Messaging, e-commerce, productivity tools, health and fitness, home banking, payments and many more are just some of the examples of mobile applications that handle end-users' data. The mobile platform has become the more personal and intimate, and at the same time critical, of all the other end-user devices [2].

As the trust of end-users on these mobile platforms and applications augments, more and more users adopt and use them daily. However, as the number of users increases and the amounts of critical information deployed on these mobile platforms, they become more attractive to malicious attackers that will try to obtain unauthorised access to mobile devices and users' data [3]. Attackers are targeting mobile platforms increasingly, both on the two major mobile platforms - iOS and Android [4]. Android, due to its market penetration (around 80% of all mobile devices in the World) and openness (Android is free and open, and smart devices manufacturers use it as the basis for their systems) make it more attractive to attackers [5].

Mobile platforms are becoming increasingly popular, and the number of developers is growing. This fact makes these platforms more prone to development mistakes that are most of the times, translated into vulnerabilities that can be exploited by attackers to target end-users [6]. Therefore, developers need to have caution when writing their applications to assure that they are free from these vulnerabilities. Most developers have good knowledge about mobile development frameworks but have difficulties understanding mobile software development risks and how to mitigate them to increase the quality of the developed mobile applications. In terms of security, lowering the number of development-introduced security vulnerabilities will result in applications, that pose lower security risks for the end-users.

Therefore, developers require training to tackle these security risks that might translate into vulnerabilities that might compromise end-user security. Currently, developers can use multiple sophisticated tools to discover and analyse weaknesses. However, these tools are not simple to use, and numerous tools often generate different results, making the developers task a daunting and exhaustive experience. Moreover, this experience repeats itself each time newer versions of the application are released. This paper, considering these problems, presents a developer's security feedback system that simplifies mobile applications secure development. Also, it helps developers discover and correct security vulnerabilities when submitting applications to app stores, providing feedback and helping them understand the risks and correct those vulnerabilities. The provided feedback will help increase the overall security of the mobile application before it is made publicly available on the app store, downloaded and installed on the end-user mobile device.

Android (and other platforms) app stores act as the trust gateway between the mobile application developers and end-users' smartphones. When the user installs a mobile application on their device, they should have the assurance that the application passed a robust quality assurance process to assure the security of the application when it executes on the mobile device. Currently, the most well-known Android mobile app stores have quality assurance processes that evaluate the applications' security with signature-based mobile antivirus and antimalware tools but are not looking specifically into insecure mobile application development practices.

978-1-7281-1075-2/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSTW50294.2020.00046

Aptoide (www.aptoide.com) is an Android app store with over 220 million unique active users and partnerships with over 15.000 app developers, having almost 1 million apps available. Being an entity between developers and users, the app store has the means for analysing the apps received according to security and quality parameters before making them available to consumers. Despite being considered by independent research, one of the most secure marketplaces [7], Aptoide continues to strive to face the moving-target nature of mobile malware and poorly developed applications. Fostering safe mobile software development practices is considered a prevention strategy because by reducing the security vulnerabilities in the apps distributed, the app store makes it hard for malicious applications to exploit or steal personal data/assets from its users. This objective can be achieved by analysing apps vulnerabilities as presented in this paper and by providing developers with action-oriented feedback about how to mitigate the security of their apps breaches and make their user-base safe.

This paper starts by providing a brief introduction about the secure development of mobile applications context. The architecture of modern mobile applications and the identification of the significant risks of mobile application development will help introduce the secure software development theme. The second part of this paper is devoted to the description of the developed system. The system is capable of analysing Android applications development vulnerabilities through the integration of multiple scanning tools and provide detailed feedback to the developers. The provided feedback will help them understand the security problems that affect the final quality of mobile applications. It will also provide education and training, so developers implement appropriate modifications and security mitigations. Finally, at the end of this paper, we present several conclusions from this work and introduce some future work directions.

II. SECURE MOBILE APPLICATIONS DEVELOPMENT

In the following section of this paper, we approach the secure mobile applications development. Developers play an important role on the security of the application's execution on the end-users' devices and how they can be targeted by other malicious applications or users to compromise the user data or conduct other types of obnoxious activities. Developers must understand the different risks that affect mobile applications and know which are the recommendations that can help mitigate those security risks.

A. The architecture of a modern mobile application

Most of the times, there is a misconception on what relates to mobile applications. They are often only looked as software applications that execute on the mobile device. However, modern mobile applications result from the aggregation of different elements and components that when joined together, provide the necessary environment for the provision of the needed functionalities to end-users. Modern mobile applications are composed of the following parts:

- **Application**: this is the result of a programmer activity to produce the necessary source code and compile it to the appropriated format (APK, in the case of Android). They are made available for installation on the end-user device through some application store (in Android, under certain conditions, users can install applications on their devices directly without requiring a proper application store). Applications running on the device may access the underlying OS services, access and store data, access hardware sensors, communicate with other applications or services and communicate through the available network.
- **Device**: the physical device that executes the client-side part of the mobile application, composed of all the hardware, sensors, operating system (Android or iOS), and specific operating services and applications.
- **Network**: the different mobile devices can use several means of communication that allow direct interaction with other nearby devices. Also, they can communicate with external services (servers or cloud) to exchange information. The communication medium can suffer from third party interference or sniffing.
- **Server**: consists of multiple external services (cloud) on distributed remote locations accessed through the Internet. These remote services provide extra functionality to the mobile application on the device through the provision of APIs to exchange information with the application.

Therefore, a modern mobile application is the intelligent combination of all these four previously described components to offer the desired functionalities to the end-user. From a security perspective, security risks are affecting all the different mobile application components. In the context of this paper, we only address the security risks related explicitly to the development of the mobile application.

B. Mobile applications development security risks

As referred before, multiple risks could affect the development of a mobile application. One of the primary references existing on what concerns the identification and classification of risks on mobile applications development is OWASP. OWASP has a different mobile application security-related projects integrated on the OWASP Mobile Security Project [8].

The OWASP Top 10 Mobile Risks [9] is a list that resumes the most prevalent risks affecting the security of mobile applications. It presents the following ten risks: (M1) Improper Platform Usage, (M2) Insecure Data Storage, (M3) Insecure Communication, (M4) Insecure Authentication, (M5) Insufficient Cryptography, (M6) Insecure Authorisation, M7: Client Code Quality, (M8) Code Tampering, (M9) Reverse Engineering and (M10) Extraneous Functionality.

Apart from the OWASP Mobile Top 10, which lists the most common and prevalent mobile applications security risks, OWASP also provides two other relevant sources of information related to mobile applications security development. The first one is the Mobile Security Testing Guide [8], which is a comprehensive manual for mobile application security testing and reverse engineering devoted to the iOS and Android mobile platforms.

The second one is the OWASP Mobile Application Security Verification Standard [8] used by software architects and developers seeking to develop secure mobile applications, as well as security testers to ensure completeness and consistency of the security test results.

ENISA has also been developing some work on this field of mobile application security development, and one of the most essential and relevant initiatives is the ENISA report on

"Privacy and data protection in mobile applications" [10]. This report presents a study on the mobile application development ecosystem and the technical implementation of the GDPR (EU regulation on privacy). The scope of this document is to provide a meta-study on privacy and data protection in mobile apps by analysing the features of the app development environment that impact privacy and security, as well as defining relevant best-practices, open issues and gaps in the field.

ENISA also has another exciting initiative in the field of secure mobile application development called "Smartphone Secure Development Guidelines" [11]. This report is a technology-neutral document produced for developers of smartphone applications as a guide for developing secure mobile applications. This guide comprises the following aspects, also addressed by OWASP: (1) Identify and protect sensitive data; (2) User authentication, authorisation and session management; (3) Handle authentication and authorisation factors securely on the device; (4) Ensure sensitive data protection in transit; (5) Secure the backend services, and the platform server and APIs; (6) Secure data integration with third-party code; (7) Consent and privacy protection; (8) Protect paid resources; (9) Secure software distribution; (10) Handle runtime code interpretation; (11) Device and application integrity; (12) Protection from client-side injections; and (13) Correct usage of biometric sensors.

NIST also has some work on this mobile applications security field being the most visible the "Vetting the Security of Mobile Applications" report [12]. The purpose of this report is to "help organisations understand the process for vetting the security of mobile applications, plan for the implementation of an application vetting process, develop application security requirements, understand the types of applications vulnerabilities and the testing methods used to detect those vulnerabilities, and determine if an application is acceptable for deployment on the organisation's mobile devices".

As observed in this section of the paper, it is possible to conclude that there are a different set of initiatives that address specifically the mobile applications security development. With minor differences between them, they all propose a set of security requirements that need to be addressed by the different stakeholders to have more secure mobile applications that do not pose security risks for users or organisations.

C. Secure software development methodologies

These methodologies are not specific for secure mobile application development since all software development methodologies that consider application security will need to adhere to the appropriate methodology practices to attain such objectives. Secure software development methodologies add to the traditional software development engineering processes the necessary steps to enable the design and implementation of secure software through a holistic and integrated security vision. Secure software development methodologies often cover aspects such as security requirements, threat modelling, secure coding, static and dynamic code reviews and specific security tests.

Microsoft developed their own Secure Development Lifecycle (SLD) [13] that consists on a set of practices that support security assurance and compliance requirements, helping developers to build more secure software by reducing the number and severity of vulnerabilities in software while reducing development cost. The Microsoft SDL includes the following activities: training, requirements, design, implementation, verification, release and response. To implement good security software engineering practices both on traditional and agile software development lifecycles can use this methodology [14].

Another relevant initiative in terms of secure software development is BSIMM (Building Security In Maturity Model) [15]. BSIMM is a study of existing software security initiatives that quantifies the practices of many different organisations. It describes the common ground shared by many as well as the variations that make each unique. BSIMM model is composed of 116 activities grouped into four domains: Governance, Intelligence, Secure Software Development Life- cycle Touchpoints and Deployment [15]. The Secure Software Development Lifecycle Touchpoints includes those practices associated with analysis and assurance of particular software development artefacts and processes, such as architecture analysis, code review and security testing [16].

OWASP also has a relevant initiative in this field that is SAMM (Security Assurance Maturity Model). SAMM is a framework from OWASP that can help the organisations to assess, formulate, and implement a strategy for software security. SAMM can integrate with existing Software Development Lifecycle. SAMM is fit if the organisation is mainly developing, outsourcing, or instead, focusing on acquiring software, or independent of the software development method [17].

III. VULNERABILITIES IDENTIFICATION AND DEVELOPERS FEEDBACK SYSTEM

This paper describes the architecture and implementation of the Vulnerabilities Identification and DEVelopers feedback system (VIDev, for short). VIDev is a system developed to implement the identification of potential vulnerabilities on Android applications (APKs) and to provide the necessary feedback for developers on how to mitigate or eliminate such vulnerabilities using information from different sources.

VIDev is mostly composed of two significant elements. The first system component is an API used to feed the system with new APKs that require security vulnerabilities revision and also to get feedback for the developers about analysed APKs. The second system component is a collection of software components, written in Python, that regularly executes to perform a security analysis on the Android APKs that require it and provide feedback to developers.

One of the primary objectives on the development of VIDev system was the possibility of integration with the existing automated Aptoide app store application quality and security assurance processes. These processes require the system to be able to handle hundreds of new mobile applications or existing applications updates submissions per day. VIDev should be able to test each of the new app submissions, determine the existence of security vulnerabilities (classified according to the OWASP Mobile Top 10 security risks) and return the appropriated feedback to the developers. VIDev can either operate directly over the APK file uploaded to the system, use a remote APK location or use the unique app store identifier (the Aptoide store uses an MD5 hash identifier). The developer receives appropriate educational feedback containing detailed information about

each of the potential vulnerabilities discovered and how to conduct the proper measures to mitigate them.

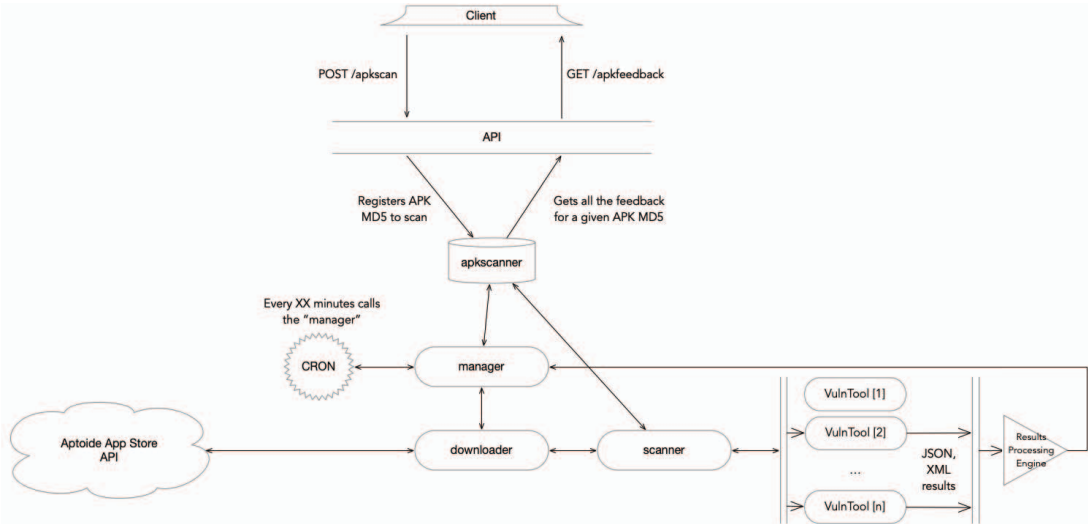

Figure 1. The overall architecture of the vulnerabilities scanning engine

A. VIDev system architecture and components

The VIDev system is supposed to receive APK identifiers from external systems, perform heavy testing on those APKs and provide feedback about the security-related findings to the developers. VIDev exposes to external clients through an API that isolates the information exchange processes from the internal operations completely.

The following image (Figure 1) depicts the overall system. As it is possible to observe, the API separates the external systems from the internal operation of the VIDev system, allowing it to operate independently and concurrently.

B. Information storage

The system stores all the necessary information and the results from the vulnerability analysis in an internal database (developed using the MySQL relational database). The database is composed of the following tables:

- **apk**: Contains all the information about a downloaded APK, so all the metadata information that exists about the APK that is being analysed by the system.
- **apk2scan**: Contains the information about the APKs that have been submitted to the system but have not yet been processed (scanned). It is a pool of APKs to be analysed.
- **apkresults**: Contains information about the results obtained after applying the appropriate scanning tools to a given APK. This structure contains information about the different collected analysis results over time.

These tables aggregate all the necessary information that the system requires to analyse the submitted APKs and stores information about the analysis results.

C. VIDev system API

One of the critical components of the VIDev system is an API that allows external systems to interact with the VIDev system internal operations. This API will provide the necessary capabilities for the analysis of APKs as well as for the collection of results from the analysed APKs. In resume, this API possesses two main capabilities: a) the ability to receive the indication of a new APK to be scanned (**apkscan**) and b) the capability of providing feedback about an analysed APK (**apkfeedback**).

The **apkscan** API entry is used by external entities to send a new APK to the VIDev system for analysis. The client should pass (POST) an MD5 unique identifier (**md5**) of the APK to scan. MD5 is used by the Android App store to identify uniquely the different APKs that are submitted. This API entry operates in the following manner:

1. The client sends a POST request for the **/apkscan** API entry, passing the identifier of the APK for scanning (**md5**);
2. API server receives the request, checks for the existence of the APK MD5 identifier;
3. API server stores the APK identifier on the internal system database;
4. API server returns the **result** of the operation to the client.

The result of the operation is a JSON-formatted message that has the following structure:

- **status**: a Boolean that can be either "true" or "false" depending on the success or not of the API call;
- **message**: a string, that contains a message with some further details of the result of the operation.

The **apkfeedback** API entry is used by an external client entity to request the results for a previously submitted APK to the VIDev system for analysis. The client should request (GET) the results passing the MD5 unique identifier (md5) of the APK for which he requests feedback. This API entry operates in the following manner:

1. The client sends a GET request for the **/apkfeedback** API entry, passing the identifier of the APK for scanning (**md5**);
2. API server receives the request, checks for the existence of an MD5 identifier;
3. API server requests to the database the existing results for the given APK;
4. API server returns the results to the client.

The answer to this request is a JSON-formatted message and has the following structure, depending on the situation:

- If an error occurred or if the requested APK is not yet processed, a JSON-formatted message is received containing a **status** field indicating the success or failure of the API call and a message field with some further details of the result of the operation.
- If the system has already processed the APK, and the results are already available, a JSON-formatted message is received. This message contains a **status** field, a **results_history** field with information about the history of the analysis of this APK throughout the time (if the APK has been submitted for review more than once) and a **results** field with the APK analysis results and some additional feedback information. In the next section, we provide a more detailed description of this message.

In the following section, we detail the internal operation of the vulnerability scanning engine and the feedback rules system.

D. VIDev vulnerabilities analysis and internal feedback system

This section describes the core of the system and how it integrates with the previously described API that allows external entities to communicate with the VIDev system to perform APK security analysis and receive detailed feedback. Here, the internal operations of the different components of VIDev system will be detailed – each of these components interacts with the others to perform the services exposed by the API. The operation of the internal components to analyse the APKs are entirely independent of the API invocation. This way it allows multiple clients to invoke the API endpoints without having to wait for the vulnerability scanning operations to complete (some of these tasks may take a long time, depending on the type of APK, its dimension and the number of scanning tools to apply). The VIDev system checks for the existence of new APKs to scan, batch processes them, and stores the results. The client can perform regular periodical calls to the API to check if a given APK analysis is completed or not and receive the appropriate feedback (we will consider other client notification options in the future). The sequence diagram (Figure 2), describes the internal operations of VIDev system.

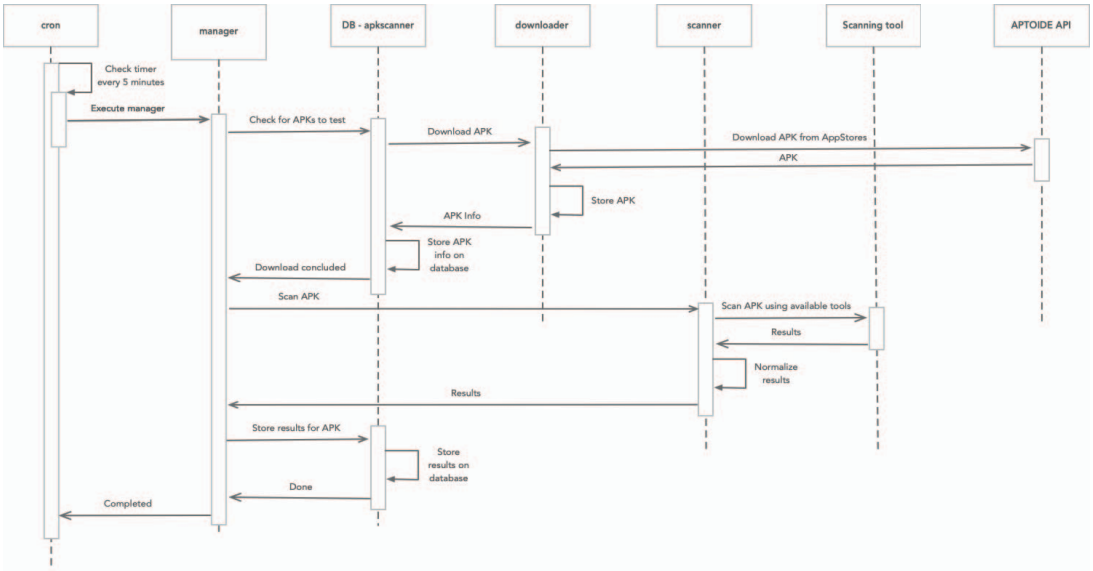

Figure 2. Overview of the scanning engine internal operations

The system operates using the available vulnerability scanning tools over the requested Android APKs, collecting the results, while returning the appropriated feedback to developers. Feedback will allow them to improve their application development through the mitigation of development vulnerabilities that might represent security threats for the end-users. The sequence of operations conducted by the system sums up in the following:

- In order not to exhaust the resources on the server, the system uses the Unix/Linux "cron" daemon to execute the necessary operations over a group of APKs. Cron is set up to run according to a specific time interval and executes the VIDev "manager". The "manager" is responsible for the central operations over the APK to analyse, for the collection of the results and for providing developers feedback;
- When the "manager" starts, it connects to the "apkscanner" database and checks for the existence of newer APKs to test – these APKs have been added previously by some external system using the API. If

there are new APKs to scan on the database, the "manager" will order the "downloader" to download the APK from the app store to initiate the vulnerabilities analysis scanning.

- The "manager" connects to the Appstore (in this specific case, the Aptoide Android app store) API and downloads and saves the APK locally to analyse;
- The "manager" stores APK metadata information on the database;
- The "manager" invokes the "scanner" component;
- The "scanner" checks for the availability of existing and integrated scanning tools on the system (VIDev uses a pluggable system that enables the integration of multiple specific vulnerability analysis on the system, permitting the extensibility of the system);
- For each of the scanning tools discovered, execute the appropriated "plugin" (scanning tool);
- Each scanning "plugin" executes their specific tests over the selected APK file;
- Each scanning "plugin" temporarily writes their results to the filesystem.
- After the "scanner" finalises its work, the "scanner" normalises the findings obtained. The normalisation process involves going through each of the vulnerabilities identified by the multiple scanning "plugins" and eliminate the duplicate findings and providing adequate feedback for each of the unique identified vulnerabilities;
- Finally, the "manager" writes the "normalised" results to the "apkscanner" database;
- The work is completed, until the next round of execution requested by "cron" daemon.

This core part of the system is independent of the API, allowing the system to receive multiple APKs to analyse without depending on the time that the actual analysis takes to complete. Since the purpose of VIDev is to integrate with an app store, where thousands of new Android apps (or new versions of already existing apps) are submitted every day is essential to make the APIs and the system core independent from each other. It is not the purpose of the work described here, but the authors already considered that in the future other ways to load balancing the analysis of APKs, across multiple nodes (servers).

E. VIDev pluggable system

One of the objectives in the design and implementation of the system was the ability to allow the system to scale and adapt to newer vulnerabilities and analysis tools. There are already a plethora of multiple Android application static and dynamic vulnerabilities analysis tools. Tools such as Androbugs [18], DroidstatX [19], AndroWarn [20], Cuckoo-Droid [21], EviCheck [22], Quick Android Review Kit [23], StaConAn [24], Mobile Security Framework [25] and others are used to detect different types of security vulnerabilities. The VIDev system can support those and any other new scanning tool that might appear in the future. Therefore, any developer can select a new scanning tool, write a simple plugin for the tool, and make it available to the VIDev system. For that, the developer must create a Python plugin file, using a specific supplied integration template file, and adapt it to the specific scanning tool. Also, the developer has to define a results integration dictionary that maps the specific results from the scanning tool into VIDev system results and feedback to be provided to the developer (Figure 3).

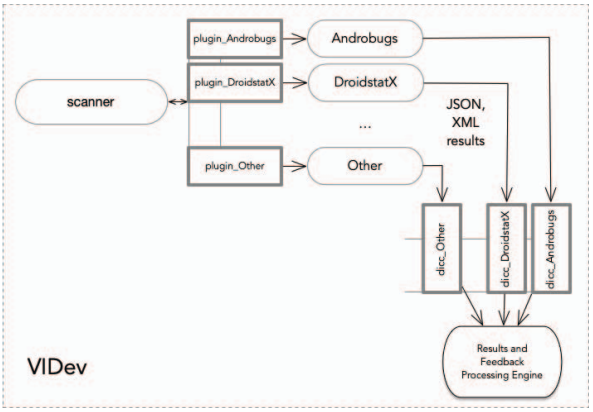

Figure 3. Vulnerabilities analysis plugin system

Currently, VIDev system already contains two plugins that offer support for two different Android static vulnerabilities analysis tools – Androbugs and DroidstatX. These two plugins are executed side-by-side over the Android application file (APK) and produce a set of specific results that are integrated and managed by the VIDev system. As more plugins (and vulnerability scanning tools) get integrated, better will be the obtained results about the existence of vulnerabilities and better will be the feedback returned to the developers about the vulnerabilities detected on the applications and how to mitigate them.

Any developer willing to develop a plugin for the VIDev system can accomplish that easily by using the source code of the plugin template provided. With that template he can implement the specificities of the scanning tool inside that template – each scanning tool has their internal mechanisms and behaviours, and that should be adopted by the developed plugin to allow the scanning task to accomplish its objectives and provide results. Also, another critical requirement for the plugin developer is to provide a specific dictionary (a simple file in JSON format). This dictionary maps the results discovered by the specific scanning tool (developed for the plugin) into the generic, integrated and categorised results format that's being provided by VIDev system. The dictionary uses the OWASP Mobile Top 10 risk classification methodology, aggregating the tools results in the "M1..M10" OWASP risks methodology.

F. Results and feedback

As presented in the previous sections of this paper, the ultimate objective of the VIDev system is to automate APK testing and produce appropriate feedback for Android mobile application developers about existing vulnerabilities on their applications. Also, it can provide appropriate feedback that can help developers to learn how to mitigate those vulnerabilities. When integrated on an app store mobile applications pipeline, VIDev can contribute to the reduction of insecure Android applications. It can accomplish this objective by warning the app store managers and application

developers about possible security vulnerabilities that require mitigation.

Therefore, the system produces integrated security vulnerability analysis, using a set of specific existing mobile applications security analysis tools. VIDev collects the findings of each one, and aggregates, classifies and categorises them into a single results file that follows the OWASP Mobile Top 10 risks methodology (the risks range from "M1" to "M10"). These aggregated results are returned to the client (via the **/apkfeedback** API entry, as previously presented) using a JSON formatted response composed by several different structures. It is possible to interpret these structures by various means (other tools, a mobile or web application, or any other). Another important aspect is the useful educational feedback provided to the developers for each of the identified vulnerabilities. Developers receive additional help (URL's for web sites, videos, books and other resources) that will allow them to understand better the reported vulnerabilities, how they work and how attackers exploit them, and how to correct or mitigate them.

The "**response**" that contains the appropriate vulnerability scanning results and the developer's feedback is JSON-formatted message that is composed by multiple structures - "**status**", "**results history**" and "**results**" (Figure 4). The first one, "status", is a simple JSON object that contains a Boolean value that indicates if the API request has succeeded or not, and an optional "**message**" that contains some additional information in the event of an error.

The second element on the result is the "results history" element composed by an array of JSON objects that represent the history of the different vulnerability analysis conducted over the time on a given Android mobile application or on the different versions of that application. The results history is essential because it allows tracking the security maturity of a given application over time. Each of the "results history" elements have the following structure:

- "**created at**": represents the time and date of the performed scanning activity;
- "**details**": a string of data that provides extra details about the performed scanning activity. These extra details may include details about some specificities of the tools used or some specificities about the execution conditions that were present;
- "**id**": each of the scanning activities is uniquely represented and identified by the system. This identifier is part of the JSON object;
- "**md5**": this is the unique representation of the Android mobile application APK in the App Store. In this case, the Aptoide App Store represents each APK uniquely in the system with an MD5 hash;
- "**results location**": the server stores the different results produced by the multiple vulnerability scanning tools used. This field is the server storage location containing the specific scanning results with further detailed information;
- "**scantools**": this represents the identification of the multiple specific scanning tools used to perform the security analysis and vulnerabilities identification on this run;
- "**status**": this is the representation of the success or not of the specific scanning operation that took place at this moment in time.

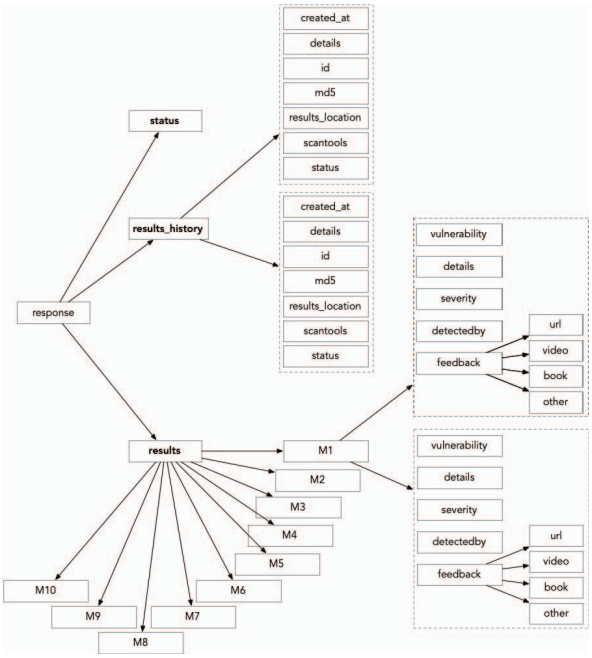

Figure 4. Feedback JSON formatted message

Finally, the last element on the result is "results", composed by a specific subset JSON objects, and named according to with the OWASP Mobile Top 10 nomenclature, ranging from "M1" up to "M10". VIDev gathers the specific results from the multiple vulnerability scanning tools used, categorising and classifying them according to the OWASP naming. This approach will help organise and enumerate the multiple vulnerabilities into a general risk classification metrics allowing developers to quickly understand the security risks of their Android applications and learn how to take the appropriate actions to mitigate them. Each of the "M1..M10" structures is composed of an array of multiple vulnerabilities (if they exist), that has the following structure:

- "**vulnerability**": this corresponds to a single sentence that clearly describes the vulnerability identified in the conducted analysis. This vulnerability might be identified by a single tool or by a different set of tools;
- "**details**": in this field, a more detailed description of the vulnerability is presented, providing a more verbose explanation of the detected vulnerability with all the relevant details as well as some possible CVE or CWE enumeration, if it exists;
- "**severity**": not all the vulnerabilities detected share the same severity classification. Vulnerabilities are grouped according to their critically. The higher the level of severity the more critical they are, ranging from a highly exploitable vulnerability that can compromise the end-user easily to something that is less exploitable and can let an attacker get some information about the execution environment. If more than one scanning tool reports the same vulnerability

with different severity levels, then the average level of those severities is considered;

- "**detectedby**": this field indicates which scanning tool has detected this vulnerability – if different scanning tools detected this vulnerability then they are enumerated here;
- "**feedback**": one of the objectives of VIDev system is to provide appropriate feedback. This feedback includes not only information about the existing development security vulnerabilities that are present on their Android applications, but also information about the way the detected vulnerabilities operate and how they affect the security of the end-users and how they can correct or mitigate them. Thus, this feedback mechanism intends also to educate mobile application developers not only about development risks but also about good development practices that need to be employed to correct these development security risks. VIDev system returns a plethora of additional information in a specific format. First, an "**url**" field with a list of URLs that point to online resources containing relevant information about the vulnerability and how to correct it. Second, a "**video**" field containing links to videos with information relevant about the vulnerability. Third, a "**book**" field indicating books that refer to the vulnerability. Fourth, an "**other**" field containing any other relevant information related to the vulnerability.

This structure returns upon invoking the API by any client application that submits some Android application for analysis that can process it and present to the developer. Currently, VIDev integrates with Aptoide's security and quality assurance system, and a web-based system presents the results allowing further analysis and configuration to personalise how to deliver the end-user feedback.

IV. Tests and Results

To evaluate the VIDev system and to assure that the automated APK testing and developer feedback mechanisms were effective to detect Android mobile application security vulnerabilities, we have conducted several tests. These tests used the available data (both existing APKs and corresponding metadata) on the Aptoide app store. The testing procedure was entirely automated without any manual intervention and consisted of two phases. On the first phase, we have selected from each of the categories existing on the Aptoide app store (44) the top popular applications, in terms of downloads, by the end-users. The selection process resulted in a total of 1193 Android applications to be tested by the VIDev system and the appropriated identification of possible security vulnerabilities. The selected APK had an average size of 27MB, with the larger APK having 1.37 GB and the smallest one having 27 KB in size. In terms of downloads from the Aptoide app store, the average number of downloads of the selected apps was 263 million (most downloaded app: 3625 million downloads; least downloaded app: only six downloads).

The tests executed on a single Dell OptiPlex 790 server equipped with an Intel Core i3-2120 CPU running at 3.30GHz with 4GB of RAM. The server was running Ubuntu Server with the 4.15.0-72-generic kernel as the operating system. The configuration of the VIDev used three different vulnerability analysis plugins - Androbugs, Droidstatx and Androwarn. These plugins executed against all the selected Android applications. The execution took a total of 21 hours and 32 minutes to complete, with an average analysis time for each app of 1 minute and 5 seconds (the most laborious application to analyse took 33 minutes, while the easiest one took just 1 second).

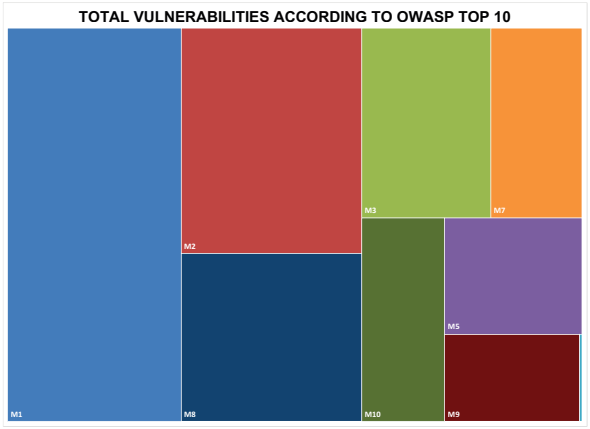

Figure 5. Distribution of the detected vulnerabilities according to OWASP Mobile Top 10

After the VIDev system concludes the security analysis of the 1193 Android apps, it was possible to observe that one of the plugins (Droidstatx) failed to analyse and produce results for 79 Android apps (6,6%). However, the other two plugins used (Androbugs and Androwarn) run flawlessly on the totality of the Android apps. This fact highlights one of the advantages of the VIDev system because it allows the collection of test results from a multiplicity of vulnerability scanning tools, offering a 100% coverage of all the testing targets.

VIDev system has produced a set of evidence that resulted in the identification of 16730 security vulnerabilities. These vulnerabilities were mapped to the OWASP Mobile Top 10 risk identification methodology, aggregating the test results from the different vulnerability scanning plugins used by VIDev (Figure 5). From this analysis, it is possible to identify (M1) Improper Platform Usage, (M2) Insecure Data Storage and (M8) Code Tampering as the most prevalent vulnerabilities identified on the tested mobile apps. These three types of OWASP risks account for 88% of the detected vulnerabilities (Figure 6). It is also interesting to notice that the automated VIDev tests detected no vulnerabilities related to insecure authentication (M4). This aspect reveals that most mobile application developers put determined efforts on this specific aspect of their mobile apps.

Furthermore, the VIDev system automated tests conducted revealed that the "Shopping", "Weather", "Maps and Navigation" and "Transport" categories are the ones that contain the more significant number of detected security vulnerabilities (Figure 7). All these vulnerabilities were also organised and categorised according to the OWASP Mobile Top 10 risk methodology.

The conducted tests resulted in 1193 security feedback reports sent to the developers. These reports included information about each of the security vulnerabilities discovered by the VIDev system. They also contained specific feedback that allowed developers to get a better insight about

the security vulnerability and its risks as well as some advice on how to further investigate the vulnerability and correct it.

From the tests, it was possible to conclude two main aspects. First, the required average time to conduct the proper security tests and analysis by VIDev on a specific Aptoide app stored submitted Android app is around 1 minute and 30 seconds. Depending on the complexity and size of the app, this time can grow up to 30 minutes (on the conducted tests hardware configuration). VIDev used three different scanning plugins, so the average time to conduct these types of security tests would grow with the increase in the number of plugins. Second, VIDev was able to detect an average of 14 security vulnerabilities per Android app and an average of 380 security vulnerabilities per app category. This number shades a light about the security of the applications that end-users install on their own devices. Also, it reveals how these vulnerable Android applications evade the quality assurance processes implemented on the app stores and how they are made available for download and install on millions of end-user Android devices.

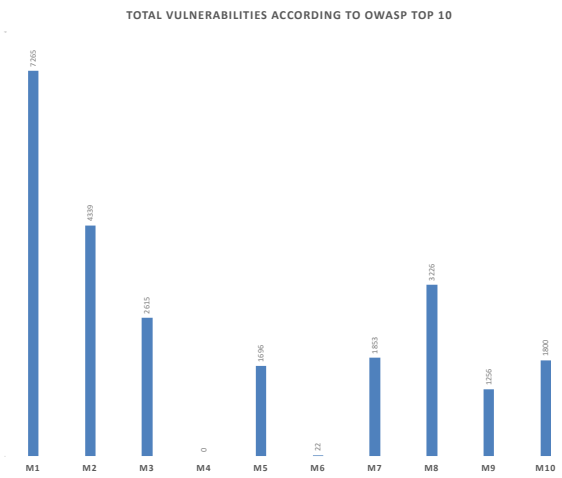

Figure 6. Number of vulnerabilities identified according to the OWASP Mobile Top 10

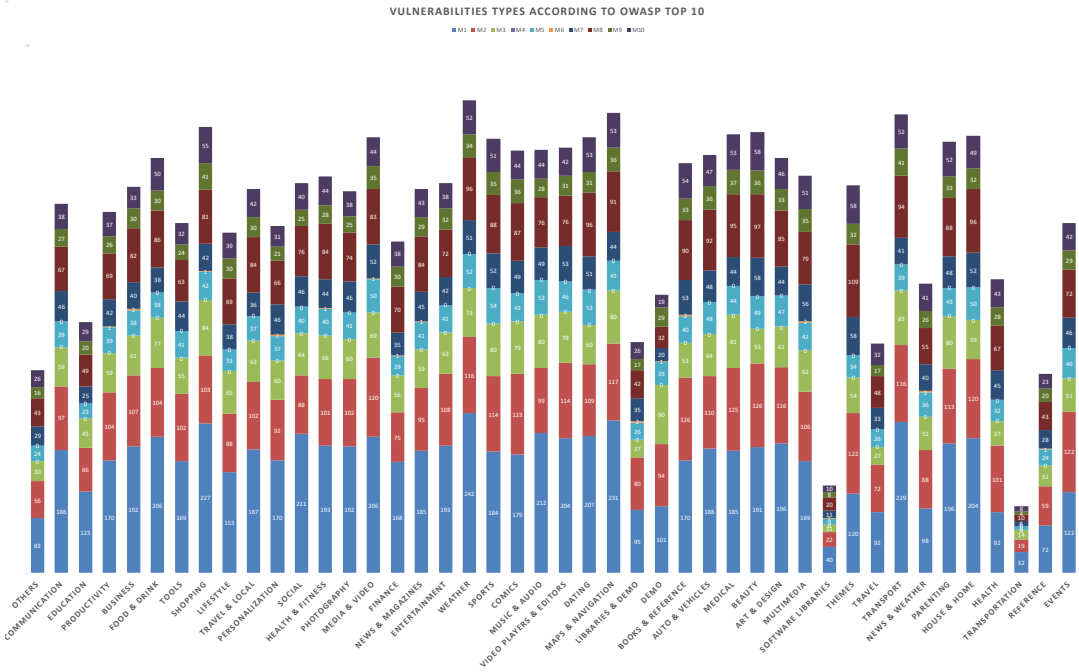

Figure 7. Number of vulnerabilities discovered by Android app category according to OWASP Top 10

V. CONCLUSIONS AND FUTURE WORK

As the number of users using smartphones and mobile applications continues to grow, also the security risks to which user's data is exposed is climbing. These mobile platforms are already a target of choice for attackers/criminals that are exploring several attacks to compromise the users.

Developers of mobile applications have an essential role in the security of mobile applications because there are responsible for their design and implementation. Some implementation errors often lead to security vulnerabilities. Most of the times, developers have in-depth knowledge of the technologies used to develop mobile applications but are not aware of good development security practices. VIDev is a system proposal aiming to improve this panorama. It provides to Android mobile application developers the opportunity to automate security testing and evaluate the development security vulnerabilities on their apps and receiving adequate feedback that will help developers correct their apps and mitigate vulnerabilities. VIDev integrates with the Android app store quality assurance processes – in this case, Aptoide. The system tests and analyses the multiple submitted applications and report the security problems to developers

before the applications are accepted on the app store and made available to the millions of end-users and their devices.

Although VIDev contributes to the mobile application security improvement and developer's education, it uses an automated test process that detects security vulnerabilities. As any fully automated process, it has some limitations. The most important limitation is the existence of false positives, i.e. the erroneous detection of security problems that need manual confirmation. Therefore, as any security analysis, any specific finding needs to be adequately verified. VIDev can help pinpoint potential security problems, but they need to be further confirmed by the developers. Also, as the VIDev system continues its development, we are currently studying the possibility of using machine learning techniques to improve the developers provided feedback about the security vulnerabilities found, and to improve the vulnerability identification process by reducing the number of false positives.

ACKNOWLEDGEMENTS

This work is part of the AppSentinel project, co-funded by Lisboa2020/Portugal2020/EU in the context of the Portuguese Sistema de Incentivos à I&DT - Projetos em Copromoção (project 33953).

REFERENCES

[1] J. Clement, "Mobile app usage - Statistics & Facts," *Statista*, 2019. [Online]. Available: https://www.statista.com/topics/1002/mobile-app-usage/.

[2] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, and S. Ge, "An empirical study of investigating mobile applications development challenges," *IEEE Access*, vol. 6, pp. 17711–17728, 2018.

[3] J. Khan, H. Abbas, and J. Al-Muhtadi, "Survey on mobile user's data privacy threats and defense mechanisms," *Procedia Comput. Sci.*, vol. 56, pp. 376–383, 2015.

[4] P. Faruki, V. Laxmi, A. Bharmal, M. S. Gaur, and V. Ganmoor, "AndroSimilar: Robust signature for detecting variants of Android malware," *J. Inf. Secur. Appl.*, vol. 22, pp. 66–80, 2015.

[5] I. Mohamed and D. Patel, "Android vs iOS security: A comparative study," in *2015 12th International Conference on Information Technology-New Generations*, 2015, pp. 725–730.

[6] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis, "Rise of the planet of the apps: A systematic study of the mobile app ecosystem," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 277–290.

[7] Y. Ishii *et al.*, "Understanding the security management of global third-party android marketplaces," in *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics*, 2017, pp. 12–18.

[8] OWASP, "OWASP Mobile Security Project." [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project. [Accessed: 11-Dec-2019].

[9] OWASP, "OWASP Mobile Mobile Top 10 (2016)." [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10. [Accessed: 11-Dec-2019].

[10] ENISA, "Privacy and data protection in mobile applications," 2018.

[11] ENISA, "Smartphone Secure Development Guidelines," 2017.

[12] S. Quirolgico, J. Voas, T. Karygiannis, C. Michael, and K. Scarfone, "Vetting the Security of Mobile Applications," 2015.

[13] M. Howard and S. Lipner, *The security development lifecycle*, vol. 8. Microsoft Press Redmond, 2006.

[14] M. Howard, "Building more secure software with improved development processes," *IEEE Security and Privacy*, vol. 2, no. 6. pp. 63–65, Nov-2004.

[15] G. McGraw, "Software security and the building security in maturity model (BSIMM)," *J. Comput. Sci. Coll.*, vol. 30, no. 3, pp. 7–8, 2015.

[16] B. Chess and B. Arkin, "Software security in practice," *IEEE Secur. Priv.*, vol. 9, no. 2, pp. 89–92, 2011.

[17] G. McGraw, B. Chess, and S. Migues, "Building security in maturity model," *Fortify & Cigital*, 2009.

[18] Androbugs, "AndroBugs Framework." 2015.

[19] C. André, "DroidstatX." 2019.

[20] D. Thomas, "AndroWarn." 2019.

[21] I. Revivo and O. Caspi, "Cuckoo-Droid." 2017.

[22] M. N. Seghir and D. Aspinall, "Evicheck: Digital evidence for android," in *International Symposium on Automated Technology for Verification and Analysis*, 2015, pp. 221–227.

[23] Linkedin, "Quick Android Review Kit." 2017.

[24] V. Cox, "Static Code Analyser." 2017.

[25] MobSF, "Mobile Security Framework - MobSF." 2019.