Automatic Malware Detection in Mobile Apps

Pedro Miguel Carapuça Maltez, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

Abstract—Mobile devices have an important role in lives of their users, especially smartphones. The various functionalities offered by these devices have attracted more and more users, who use them in the most varied tasks. With the increasing number of users and the possibility of performing operations such as payments and online bank account management, there has been an increase in the number of cyber-attacks on these devices, especially on the ones based on Android operating system. This project aims to offer a solution to increase the security of these devices, through a system that allows to isolate applications. The proposed system runs applications on a server and virtualizes them on users’ mobile devices through a client application. The server intercepts functions such as the user interface composition and access to other local resources sending this information to the client application on the mobile device, so that it can rebuild the user interface and interact with the user. The client application sends information about user interactions to the server. With this system, personal data stored in mobile devices is isolated from applications. In order to position this solution with other solutions that implement similar systems, a comparison was made between the solutions, which also allowed to understand the important aspects that should be addressed in order to provide a more secure environment to users.

Index Terms—Android, Virtual Mobile Infrastructure, Mobile Device Security, Server, Client Application.

I. INTRODUCTION

Mobile devices have an important role in lives of their users. Increasing their processing power, memory and data storage made these devices allow a wide variety of features to their users. Within the group of mobile devices, smartphones stand out, which in addition to the features offered by traditional mobile phones (voice calls, SMS, etc.), also allow the use of applications that offer features similar to a computer.

These features have made smartphones devices used by millions of people around the world. The number of smartphone users is increasing annually and is expected to reach 2.32 billion users by 2017. This number continues to grow, with a target of 2.87 billion users in 2020 [1].

The increasing number of users and the usage of mobile devices to perform activities related to managing bank accounts or paying expenses online has made these devices a target for cyber-attacks. These threats have not only increased in number, but also in terms of sophistication and complexity [2].

In 2016, Kaspersky Labs detected around 40 million attacks from malware installed on mobile devices. About 78% of these attacks happened on smartphones, with Android devices being the most targeted with 74% of the infections reported on.

In order to avoid these attacks, solutions have been developed to increase the security in mobile devices. It is also with the aim to offer to mobile users, a secure and dynamic environment where they can use the applications that the proposed solution is presented as proof concept.

In the proposed solution applications run in a secure environment on a server and will be virtualized on users’ mobile devices, through a client application. In this way the applications are isolated on the server, without access to the confidential information of the users.

The proposed form of virtualization has as its differentiating point, in the format which the information is transmitted between the server and the client application. Unlike other solutions, the information is transmitted inside a String.

This dissertation also presents a comparison between existing solutions, relating them to the purpose of this dissertation and identifies which characteristics can be implemented to complement the proposed system.

This document consists of six sections describing the process of developing the proposed system. In the first section is introduced the project and presented the goals of this work. In the second section is presented a review of the solutions related to this theme. In the third section are presented some features of the Android operating system, in order to contextualize and help to justify the decisions taken during the implementation of the solution. In the fourth section the proposed system is described, presenting its architecture and its components. The fifth section presents the way the solution was evaluated. In the sixth section the final conclusions and some characteristics that should be implemented in the future are presented.

II. RELATED WORK

A. Solutions with Operating System Modifications

Solutions based on modifications in the operating system take advantage of the isolation offered by it, modifying only some libraries in order to complement this isolation.

The [3], [4] solutions implement a set of policies, which aim to control the communication of applications with other components and control their access to the user’s data. The policies are activated according to a set of attributes from the other components or through variables (location, network, etc.) that define a context in which the application is being used.

The implementation of these policies was accomplished
through modifications in Android middleware.

The [5] solution monitors system calls made by applications by using the ptrace function and determines whether an application is running with unlawful intent through analyses of the pattern of its system calls.

The solution also lets to monitor communication through Binder and enable or disable security policies through a service accessible to users.

The [6] solution allows the isolation of applications using a system of security levels. If two applications have different security levels, they cannot communicate with each other. This solution analyses applications and data and classifies them into a "colour" system. The classification of the applications is based on information about the application, about its programmer, about the permissions it requests or through certificates.

These are some examples of solutions that aim to increase the security in Android through modifications in its structure. This type of solution offers a strong isolation to the applications, since it combines the process-level isolation offered by the system with a control of the communication with other components and a control of the access to the resources.

Since this control is achieved at the operating system level, it is difficult for applications to avoid it. However, despite the strong isolation provided, these solutions require changes to the operating system image or require the application to run with root privileges. This causes difficulties in the distribution of these solutions to users, since to use these solutions they would have to install new versions of the operating systems on their mobile devices which for most users would not be trivial.

B. Solutions without Operating System Modifications

These solutions increase the security of mobile devices without requiring any modifications to the Operating System. They inject code into the application source code to monitor its operation.

The [7] solution allows the user to restrict the permissions of an application through the usage of security policies. This solution injects the logic that allows the verification of policies within the application source code.

Solutions similar to [7], although they do not modify the operating system image which makes it easy to distribute it to users, have some limitations. They do not create a strong barrier between the application and the monitor component (monitor component is in the application code. Runs with the same UID) and it may lead to applications being able to bypass the control performed by the monitoring component and be able to take advantage of their permissions.

The solution [8] monitor the application through a set of new permissions, more restrictive than the permissions used by Android. The solution overrides the permissions of the applications injecting a set of new permissions. It also injects a library that intercepts sensitive methods and communicates with a monitor service to evaluate if the applications have the necessary permissions to perform the actions through these methods.

The solution [9] loads the applications source code into an isolated process. An isolated process runs with its own UID and unprivileged, not being able to communicate with other applications and services through IPC or make system calls.

It uses a component that intercepts system calls and communications with other components, and after analysing these communications decides whether to forward them or not.

The solution [8] mitigates the problem identified for the [7] solution as it separates the monitoring component into a service (monitor and application do not run on the same UID) which prevents the applications from taking advantage of the solution permissions. The same applies to the [9] solution, which runs the application in an isolated process (different UID), also avoiding the problem of a possible use of the privileges of the monitoring component by the applications.

C. VMI Solutions

1) Virtual Mobile Infrastructure (VMI)

Virtual Mobile Infrastructure [10] is a technology that hosts a mobile operating system on a server, which can be accessed by several clients. Applications running on the server are virtualized on the client's mobile device through an application.

Applications and their data never leave the server and what is presented to the client on his mobile device is a virtualization of the application. The server sends an image of the applications’ user interface and the client sends information about the user's interactions to the server.

2) Common features

The fact that solutions are based on VMI platforms means that there are features that are common to all solutions.

All the solutions offer a workspace with the applications, to each user. These applications run on the server and are never installed on the mobile device. In this way applications never have access to personal information stored on users' mobile devices.

Although the mobile operating system running on the server is Android, the client application is independent of that operating system. This allows users of IPhone, Blackberry, Window phone, etc. to be able to use Android applications. This feature also facilitates the work of app developers, since they only need to develop applications for Android [11].

The server sends information about the user interface to the client application and receives information about the user interactions. Information about the user interface is received in the client application in the form of an image that cannot be changed or analysed on the side of the mobile device [12]. These solutions also allow the use of hardware resources present in the mobile device, such as GPS, camera, etc.

These are the common features, below will be presented the unique features to each solution.

3) SierraVMI

The solution [13] authenticates users through multi-factor authentication or through single sign-on authentication. Communication between the server and the client is encrypted using the SSL protocol.

This solution allows an instance of the virtual machine to run multiple clients and enables it to monitor user activity to detect cyber-attacks. Supports 3D games and the access to the system can be limited depending on the time of day or location.
of the user.

4) **NuboVMI**

Nubo’s solution [14] uses a patented Remote Display Protocol that allows: an automatic adaptation to the dimensions of the screen of the mobile device; efficient bandwidth utilization; and use of the system in 3G networks. Communications between the server and the client are protected with TLS 1.2 and with algorithms specified by NSA Suite B.

This solution also allows an instance of the virtual machine to run multiple client sessions. NuboVMI isolates users through a sandbox. Clients are authenticated with company credentials and later with a password each time they log in. The information used by applications is stored on a different machine than the one that runs the operating system, facilitating the backup, and is encrypted with a different key for each user.

5) **Raytheon Secure Access Mobile**

In this solution [15], the users authenticate through identification tokens and certificates. Allows to monitor user activity and uses a gateway to provide a secure connection between the server and the mobile client application, enforcing the access policies. This gateway also makes sure that no data is passed between the server and the client. This way the data never leaves the server. Communication between the client and the gateway is done through a VPN.

6) **Hypori ACE**

The Hypori solution [16] provides a web application that allows companies to manage the entire system. Users authenticate with the service through a user ID, password and a certificate. Communication between the server and the client is secured with TLS 1.2.

This solution can scan the user’s mobile device to see if there is any malware that could compromise the device. Of all the solutions described, this is the only one certified by Common Criteria.

Recently, Hypori has developed a system that allows to access a specific application without the need to access the workspace.

7) **Avast Virtual Mobile Platform**

Avast Virtual Mobile Platform [17] allows to limit users’ access based on their location or network. It lets to know which applications are being used by users, for how long and by whom they were used. The users authenticate with the system by a two-factor or single sign-on authentication and allows the use of the system on 3G and 4G networks.

8) **Trend Micro Virtual Mobile Infrastructure**

Trend Micro’s solution enables single-sign-on authentication and secure server-to-client communication using TLS.

The Remote Display Protocol used adapts the coding of the information (images) to the different types of mobile devices of the clients (Android, iOS, Windows) [18].

As can be seen by some characteristics of the solutions, they have the ability to solve the initial problem.

Ensuring the security of the information transferred between the client and the server through encryption, allowing the use of the system in public networks is an advantage that these solutions offer.

However, the fact that some solutions manage to monitor user activity and the entire system as administrators invade user privacy.

Another problem arises from the fact that, in the Remote Display protocols the information passed from the server to the client is essentially composed by images. This is a problem for users who are connected through the operator’s mobile data, increasing the communication cost.

### III. ANDROID

#### A. Android Security Model

1) **Linux Kernel Security**

At the level of the operating system, the Android platform offers the security of the Linux Kernel, as well as a component that allows the communication between the different applications and services that run in different processes (IPC), in a safe way.

In Android each application is seen as a different user and runs in a different process from other applications. This way of treating the applications allows to isolate the applications and their resources in a Sandbox. Android assigns to applications at its installation time, a unique UID. Each application has its own data space that can only be accessed by itself. So applications are isolated, both at the file and process level.

Although applications are isolated, sometimes it is necessary to ensure access to device components, files, network or other services provided by the operating system. For this, Android provides a permission’s system. These permissions will be analysed by the operating system, which will then decide whether to grant them or not.

2) **Inter-Process Communication**

To enable secure communication between processes, Android uses a framework called Binder, which combines a framework with a kernel driver. However, inter-process communication is also limited by security measures. The Binder driver makes sure that the process that initiates the communication has the necessary permissions to communicate with the destination process, comparing UIDs.

#### B. Binder

Binder’s main objective is to offer a mechanism to provide inter-process communication. It allows to invoke methods in services that run in other processes, as if were invoking them locally. This feature is similar to Remote Procedure Protocol (RPC).

In terms of security, Binder checks the permissions of the application or service that is invoking the method, to ensure that it has permissions to do it. The structure of Binder is implemented in three levels in the structure of the Android.

1) **Binder in the Application Layer**

Whenever a user wants to use a feature offered by a service, they must use a specific manager for that service [19]. These managers are java objects that link to the Binder framework, abstracting developers from its structure.

Managers are initially unknown by applications and services. To get one they need to communicate with the Service Manager that keeps a record of the services and maps them through their
names to the respective managers.

When the user evokes a method through the manager, it will call a proxy that will evoke through Binder the method in the destination service.

Proxies are objects that receive requests to evoke methods in services from managers and send them through Binder. Communication in Binder is called by Binder Transaction and the information is passed through Parcel objects. The proxy serializes the data and places them within the Parcel.

On the services side there is an object called a stub, which is responsible for receiving the requests for the service. It handles the information coming from Parcel, identifies which method to evoke and evokes it in the service.

2) Libbinder

The communication between processes can only be performed in the Kernel, so there must be a middleware between the binder java part (which allows interaction with the applications) and the Kernel (where information passes between processes). This middleware exists in the form of a shared object (.so) written in C ++ and is named by Libbinder.

The structure of Libbinder is a "reflection" of the Binder structure implemented in Java [20]. In Libbinder there are also proxies and stubs for each service. When the java proxy evokes the transact function, it is passed to the proxy object for the intended service in Libbinder through the JNI. Parcels, when passing through the JNI, are converted into Parcel objects implemented in C ++.

Upon receiving the transaction information, the Libbinder proxies will evoke their own transact function. This function will call the transact function of the IPCThreadState class, which is the class responsible for connecting the proxies with Binder Kernel Driver. The information about the transaction is placed inside a structure called binder_transaction_data which will later be placed inside a binder_write_read structure [21].

Communication between the IPCThreadState and the Binder Driver is done through the ioctl function, in which the binder_write_read structure is passed.

The Binder Driver, after receiving the transaction information, will copy this information into the service memory space. To communicate between the Binder Driver and the stub, the IPCThreadState class is used. This classe receives the information through an ioctl function. The information will be taken from the binder_write_read and binder_transaction_data structures and forwarded to the libbinder stub object. This object will pass the information to the stub implemented in java, through the JNI.

3) Binder Kernel Driver

The Binder Driver is the part of the Binder implementation where inter-process communication actually happens. It is a small Kernel module implemented in C that uses the open, mmap, release, poll and ioctl functions to allow the communication with the upper levels of the Binder structure [22]. From the functions available to communicate with the Driver, the one that is used by the upper levels of Binder to transfer information is the ioctl function. The ioctl function receives as arguments a code with the type of operation to be performed by the Binder Driver and a buffer to receive and send data.

The BINDER_WRITE_READ code is the most interesting code for this discussion, since it signals the transfer of information to other processes. When the ioctl function receives this code, the function's buffer argument for sending and receiving data is a binder_write_read structure.

The Kernel Driver copies information from the memory space of the client process to its own memory space, and then copies it to the memory space of the destination process. The information is copied using the copy_from_user and copy_to_user commands of the Linux kernel.

C. Android Graphics

1) User Interface Draw Process

An activity organizes the views that compose its user interface in a tree. When an activity gets focus, it will begin to draw its user interface (UI).

For the UI to be drawn, the tree will be traversed and each element will draw itself. The UI design process requires two passes through the tree. The first step is for views to measure themselves and the second pass aims to position the elements on the screen. The parent elements will position their child elements according to the measurements obtained during the first pass. At the end of these passages the draw function will be called, which will traverse the tree and make the views draw themselves [23].

Two ways can be used to draw the views: through the Canvas API or through OpenGL ES. The Canvas API [24] is the default API used by views and is the most used by developers. When a window needs to be drawn, it will lock its Surface [25] and return a Canvas. This Canvas will then be passed to the views, so they draw with it.

A Canvas contains the methods necessary to draw the views and a bitmap [26]. A bitmap points to a set of pixels and usually points to the buffers of a Surface. The Canvas API uses the Skia library to implement the lowest level calls [27]. To prevent two graphics producers from writing to the same buffer, the lockCanvas and unlockCanvasandPost functions are used. These functions block and unblock a buffer and are used at the beginning and at the end of the process.

2) Surface, Surfaceflinger and Hardware Composer

A Surface has a buffer associated with it and represents the part of the producer in a queue buffer. Views render through Canvas in the Surface buffer that will, normally, be consumed by the SurfaceFlinger [28] service. This service is usually who creates the Surfaces and who consumes them. It is responsible for accept the buffers and compose them.

The screen usually updates at a rate of 60 frames per second (fps). In order for applications not to update user interface content in undue time, the system sends VSYNC signals that signal the application when it should update its user interface. When the VSYNC signal arrives at SurfaceFlinger, it walks through its list of layers looking for new buffers. Once it has received all the buffers, asks the Hardware Composer how the buffers should be composed.

Hardware Composer [28] is responsible to find an efficient way to compose the buffers for the hardware available. After
informing the SurfaceFlinger how the buffers should be composed, it will receive the result of the composition from it and send it to the screen.

3) **BufferQueue**

   Passing buffers between the producer (activity of an application) and the consumer (SurfaceFlinger) is done through a BufferQueue [29]. BufferQueue provides two interfaces for communication: IGraphicBufferProducer for the application side, and IGraphicBufferConsumer for the SurfaceFlinger side [30].

   When the application wants to draw the user interface of an activity, it will evoke the lockCanvas method, blocking the Surface and returning a Canvas that will be used for drawing the views. This call will evoke the dequeueBuffer method so that the application receives the buffer with the specified characteristics (width, height, format of pixels and flags). When the UI drawing process is finished, the application will call the unlockCanvasAndPost function, which in turn will evoke the queueBuffer method, inserting the buffer back into the queue. These are the interactions on the producer side (application). On the consumer side (Surface Flinger), the acquireBuffer method will be evoked in order to receive the buffer. When the content handling process is finished, the buffer will be returned to the queue through the releaseBuffer method.

4) **OpenGL ES**

   In addition to Canvas, there is the OpenGL ES API [31] that allows to render the user interface. However, OpenGL ES only contains the primitives to draw the components, requiring the joint use of the EGL library.

   The EGL library creates an EGLSurface in which the user interface will be rendered. The EGLSurface is created with the eglCreateWindowSurface function, which accepts as arguments a SurfaceView, SurfaceHolder or a Surface (objects that can be used by BufferQueue).

   Any rendering made for the EGLSurface results in a buffer that is subject to the interactions in the BufferQueue. Unlike Canvas, the EGL library does not provide lock/unlock functions. The drawing functions are called and then the eglSwapBuffers function is invoked to submit the buffer in the queue.

   **IV. PROPOSED SOLUTION**

A. **Architecture**

   The proposed system uses a server to run Android applications that will be virtualized on the user's mobile device. In this way, applications are never installed on the mobile device, having no access to the user's personal information.

   The user downloads the MobiSec application (client application) from the Google Store and authenticates himself with the server. After authentication, a workspace will be provided to the user, which contains the applications to be virtualized.

   Applications that will be virtualized are commercial off-the-shelf (COTS) applications that can be obtained, for example, through the Google Store. The applications will be intercepted on the server and virtualized on the user's mobile device through the MobiSec application.

   The information sent from the server to the MobiSec application refers to the application’s user interface that is running on the server. This information will be received and analysed in the MobiSec application to rebuild the user interface. Conversely, information about the interactions between the user and the application is sent to the server, so that the COTS application responds to these actions.

   The system should allow multiple users to coexist on the server, where to each user will be offered a workspace containing their applications. To promote user privacy, user authentication must be required and communications between the server and the application must be protected.

B. **Server**

   The main function of the server is to host a mobile operating system, which allows the user to access Android applications securely. The server communicates with the client application (MobiSec) in order to virtualize the applications in it.

   It is also responsibility of the server to allow multiple users to access the system through authentication and provide them with a workspace so that they can access the applications.

   To implement the server was used a version of the Android operating system provided in [32]. The version used was the Marshmallow version available from the branch android-6.0.1_r41. To allow the interception and virtualization of applications and the communication between the server and the client application, were implemented some changes in the source code of the operating system. These changes will be explained in the next section.

C. **Implementation**

   1) **First Approach**

   The first approach was to virtualize the application through its interception in Binder. The server's Binder was changed in order to intercept the applications and send the information to the client application. In the client application the intercepted information would be injected into the local Binder.

   Since the transact function of IPCThreadState class is the point where all the Binder communication of an application, converge (Binder section) and it is also a point where it is possible to decipher the information about the operation to be performed this is the point to intercept the application.

   The intercepted information will be sent by a socket and consists of a Parcel, a service identifier and a function code for the method that will be executed by the service.

   In order to send a Parcel to client application, the Parcel: IPCData function of the Parcel class in C++ was used. This function gives access to the serialized information contained in the Parcel and that can be sent through the socket. On the client application side the parcel was reconstructed with the information received from the socket.

   To inject the communication into the binder of the client application, the target service would have to be identified and the function code decoded. Identifying the target service was achieved using a String with the name of the target interface. This String is the first object within a Parcel and is called token.
interface. To decode the code that identifies the methods, it generally corresponds to the position in which the method is declared in the java and AIDL files of the respective service. Their identification is possible with a prior study of these files. With all the information handled, communication would now be made with the Binder.

In order to communicate with a service it is necessary to acquire a specific manager for that service, which is obtained through the Service Manager. However, obtaining managers proved to be the major problem associated with this approach. Most managers are not available from Service Manager and some of the available ones do not allow the use of all the methods implemented, to developers.

2) Second Approach

The second approach tested was based on the interception of the applications in the Binder, in the same point previously mentioned, that is the transact function of class IPCThreadState. However, since the problem was related to the injection of communications in MobiSec's Binder, this approach diverged from the first, in that the information would be analysed first. After interpreting purpose of the communication, the intended behaviour would be recreated programmatically in the MobiSec application.

To implement this approach, the recreation of the applications user interface was tried. The goal was to intercept communication between the application and the service responsible for displaying the user interface on the screen, analyse its content and implement its components on MobiSec.

However, in the communication between the application and the SurfaceFlinger, the information is passed in buffers. This prevents the user interface components from being related to java-level objects and therefore, the user interface was not rebuilt in the MobiSec application. Due to this problem this approach was abandoned.

3) Demonstrator

Tracing the application code with the DDMS tool from Android Studio and studying the class allowed to conclude that the ViewRootImpl class was the class that should be intercepted [33].

In addition to containing the views tree, this class is also responsible for starting the drawing process of the UI. The performTraversals function will traverse the tree and call the functions responsible for drawing the UI.

The access to the views tree allows to retrieve the necessary information for the user interface reconstruction on the side of the user's mobile device and also allows injection of user actions in the views, forcing them to trigger the normal behaviour of the application. The fact that this function is always called when the application starts drawing its user interface allows to always present to user the current user interface. Due to these characteristics, it is concluded that this function would be the point where the applications should be intercepted.

To implement the communication between the server and the client application, two threads were used. One thread is used to receive information about user actions and the second thread is used to send information about the UI. However, these two threads need to be in constant communication with the main thread (UI thread).

Android provides a mechanism that allows to perform tasks on different threads. This mechanism consists of a Looper [34] that sends messages from a MessageQueue [35] to Handlers. The messages sent can be Message [36] or Runnable [37] objects. Each thread has a MessageQueue associated and can only have one active Looper. Handlers are associated with the thread that initialize it and handle the messages received, in the handleMessage function.

When the application begins the user interface draw process, the performTraversals function is invoked and a thread (ServerThread) and an Handler (ViewHandler associated with the main thread) will be instantiated. The ServerThread is used to receive information about user interactions and has as attributes: (i) a ViewHandler, to allow communication with the main thread; (ii) a socket, to communicate with the client application; and (iii) a thread (ClientThread) that sends information about the user interface to the client application.

At the initialization of the ServerThread, the communication application through the socket will be established. The ClientThread will also be initialized at this point receiving the socket. This thread was initialized from ServerThread due to the impossibility of using sockets in the main thread. This was the only way to use the socket in both threads.

Both threads were initialized and activated, and the system is ready for use. When a message is received from the socket, it will be analysed in order to know what interaction was done by the user. After the analysis, a Message object will be built to send the information to the main thread, through the ViewHandler.

The ViewHandler of the main thread, when receives the messages will evoke the method referring to the type of interaction made by the user. This method will traverse the views tree and will inject the action performed in the respective view. The view is identified by its id, since the views created in MobiSec are given the id of the view they are re-creating.

To send the information with the constitution of the UI, it will be used the ClientThread object. This thread contains a socket and an Handler. Whenever the performTraversals function is called, a function that aims to traverse the views tree and retrieve its information will be evoked. However, the first view in the tree is a view of type PhoneWindow$DecorView, common to all applications and, consequently, will not be implemented or retrieved any of its information.

The information extracted is placed in a String, where the different components of the UI are separated by the character ‘,’ and the different attributes of the view are separated by the ‘:’ character. The String will be sent to the ClientThread through its Handler and later sent to the client application through the socket.

D. MobiSec

The MobiSec application represents the client side of this system. It is a mobile application that aims to receive information from the server about the application’s user interface and recreate it. It is also responsible to intercept any
user interaction with the application and send that information to the server.

Two threads are implemented in the application to allow the communication. One thread is used to receive the composition of the UI and the second thread is used to send information about the user's actions. The main activity where the views will be recreated is initialized in blank waiting for information about the UI.

When information about the constitution of the UI is received, a Message object containing the String will be created and sent to the Handler of the main activity.

In the Handler, the String is decomposed and the respective methods for each type of view are invoked. These methods construct views from the attributes received in the String. A very important attribute in the construction of views is the ID. This attribute allows to match the re-created views on the MobiSec side with the existing views in the tree on the server.

These methods also define other methods that are intended to intercept user-generated interactions in the view. The definition of these methods depends on the attributes of the view, for example, a view with the “android: clickable” attribute reacts to the user's touches and a method must be implemented to intercept this type of interaction.

As on the server, it is the ServerThread that activates the ClientThread. Android does not allow the use of sockets in the main thread and the ClientThread has to be initialized with the socket.

The ClientThread contains a socket and a Handler and is responsible for sending information about the user's actions to the server. When the user interacts with a view, the method defined to intercept these interactions will send information regarding the interaction to the ClientThread, through its Handler, which will send the information in a String to the server, through the socket.

E. Test Case

To prove that it is possible to intercept and virtualize an application in the user's mobile device using the method described in sections IV.C.3 and IV.D, an application consisting of a LinearLayout containing a Button, an EditText and a Text View was used. Its operation is simple: the user enters text in EditText and when the button is touched, the text is shown in TextView.

At the beginning of the UI drawing process, the performTraversals function will be invoked and the ViewHandler and the ServerThread will be initialized (which will initialize the ClientThread).

After the initializations, the function that traverses the views tree is evoked and collects information about the UI. In this case, the information will refer to a LinearLayout, a Button, an EditText and a Text View. Since the goal was to prove the interception and re-creation of the application's UI on the mobile device side is possible, only the information needed to recreate the user interface was extracted.

The constitution of the String for each element is: (i) "linearlayout; width; height; orientation" to initialize the LinearLayout; (ii) "edittext; width; height; hint; text" for the EditText case; (iii) "textview; width; height; weight; text; gravity" for the TextView case; (iv) "button; width; height; text; gravity" for the Button case; (v) "finishes layout", to inform that the layout composition is finished. As mentioned above, the ';' character is used to separate the different elements within the string and the character ';' separates the attributes of the view. Finished the construction of the String, it will be sent to the ClientThread through its Handler and to the client application through the socket.

In MobiSec, the String is received in the ServerThread and sent to the main thread through an Handler. In the handleMessage method of this Handler, the String will be handled in order to use the information to recreate the views.

Once handled the String, the methods responsible for creating the elements of the UI will be evoked. These methods evoke the constructors of their classes and define the functions responsible for capturing user interactions. In this case, only the Button and EditText allow these interactions. For the Button, the onClick method was defined, which sends to the ClientThread a signal that the Button was touched. For the EditText, since it allows editing of text through an input service, a TextWatcher has been defined in order to intercept the text that is being introduced in this view. The intercepted text is sent to the ClientThread.

The ClientThread receives this information in the handleMessage method of its Handler and sends it in a String through the socket.

On the server side, this information is received in the ServerThread, which identifies the type of interaction performed and sends that information to the main thread. After analysing the information received, the function corresponding to the interaction performed by the user will be called.

If the interaction represents a Button touched, a function that traverse the views tree looking for the respective Button will be evoked and the action injected through the performClick method. If the interaction corresponds to a text input in the EditText, a function that goes through the views tree looking for the respective EditText and injects the text, through the setText method, will be evoked.

F. Comparison with VMI Solutions

1) Common Features

Some features offered by the solutions presented in section II.C must be adopted by the proposed solution, since they are essential for the operation of the proposed system (authentication system, multiple user access, etc.). However, they also present features that would be harmful to the system or that their implementation would only complement the proposed system.

These solutions allow access to multiple users and offer a workspace, containing the applications, to each user. In addition to this feature, the solutions presented allow the virtualization of applications that use hardware resources on user's mobile device (GPS, camera, etc.). These features must be implemented by the proposed solution, since they are essential for the system to offer all the functionalities of the applications to multiple users.
One advantage that comes with the usage of the proposed system is the fact that this solution presents the user interface through the reconstruction of views and not through an image. This allows users of mobile devices other than Android to use Android applications with a user interface native to that device’s type.

Another advantage is that the use of the proposed solution from the mobile data may lead to less communication costs, since the user interface is not passed on images as in these solutions.

2) SierraVMI

The SierraVMI solution authenticates users through a multi-factor authentication or through a single sign-on authentication. Both forms of authentication would be beneficial to the proposed system, however, single sign-on authentication would suffice.

Other features of this solution that would bring benefits to the system are: (i) protection of the communication between the server and the client application through SSL 4096-bit ECDHE; (ii) several users in the same virtual machine instance, reducing the number of servers required and, consequently, the cost of system implementation.

The possibility to monitor user activity and to limit the access to the system could be a necessary measure to maintain the security of the system.

3) NuboVMI

Users authenticate in this system through a password and an ID and then, with a new password whenever they log in. This form of authentication presents a valid solution to the authentication problem. The communication is protected with TLS 1.2 and with algorithms specified by NSA Suite B and represents a form of protection that could be implemented by the proposed solution.

Like the SierraVMI solution, it allows one instance of the virtual machine to host multiple client sessions. To isolate the users, Nubo’s solution uses sandboxes. This feature allows to reduce the costs of the system implementation due to the decrease in the number of servers and also guarantees the privacy of the users through its isolation in sandboxes, being a characteristic to be implemented by the system.

4) Raytheon Secure Access Mobile

This solution uses ID tokens and certificates to authenticate users with the system. This system of authentication is a good solution to the authentication problem in the proposed solution. The use of identification tokens or certificates in order to complement an authentication through a user ID and password is also a feature to take into account.

Raytheon Secure Access Mobile uses a gateway to prevent the transfer of sensitive information from the server to the mobile device. A variant of this functionality could be used in the proposed system. Using a gateway that would not allow the passage of confidential information from the mobile device to the server would increase the security of the system.

The fact that the solution allows to monitor the activity of the users is a measure to maintain the security of the system.

5) Hypori ACE

The solution uses an authentication system based on a user ID and password, complemented with a unique certificate per user. This is a good solution to the system authentication problem. The protection of the communication between the server and the mobile device through TLS 1.2 is also a valid option to be implemented by the solution.

The possibility of using an application without the access to a workspace facilitates the use of the system for users who do not need the use of a workspace. However, the implementation of this functionality by the proposed system would only complement it not being crucial its implementation to the system.

The solution also scan users’ mobile devices for malware. Implementing this functionality would be beneficial to the system, since increasing devices’ security also contributes to increase system’s security.

6) Avast Virtual Mobile Platform

The authentication method of this solution consists of a two-factor or single sign-on authentication and presents a valid solution to be implemented by the system.

The fact that it allows to monitor the user activity and to limit the access to the system depending on the network and location of users is a measure to maintain the security of the system.

7) Trend Micro Virtual Mobile Infrastructure

The features presented by the Trend Micro solution that bring benefits to the system are: (i) single sign-on based authentication; and (ii) protection of communication through TLS.

V. EVALUATION

To evaluate the proposed solution, the system was tested with the test case presented in section IV.E. The goal was to show that it was possible to virtualize an application which is running on a server, in a client application on a mobile device, through the presented methodology.

The application to virtualize was simple and its user interface was composed by a Button, an Edit Text and a Text View. This application consisted in showing the text that had been entered in Edit Text in Text View when the Button was touched.

In the server component was used the Marshmallow version available from branch android-6.0.1_r41. Some modifications in the source code of the operating system were made in order to intercept the application and obtain the composition of its user interface. Besides this point, the modifications allowed the communication with the client application and the injection of the user interactions in the application. When the source code is compiled, an Android emulator is available, which in this case represented the server with the applications.

The client application (MobiSec) was implemented in order to receive information about the user interface and to recreate it. This application runs on the mobile device of the user, which in the test case was represented by a separate Android emulator.

In these conditions it was possible to prove the virtualization of the application in the client application. The user interface has been replicated to MobiSec successfully, being identical to the one presented in the modified emulator that represents the server.

The user interactions with the application were intercepted
and these actions were injected into the application on the server, successfully. In this environment, no changes were detected in the application resulting from the fact that the application is running on the server and not on the mobile device.

VI. CONCLUSION

A. Conclusion

This dissertation presents a solution that aims to increase the security of mobile devices, offering users a secure environment to use applications. The proposed system consists in a server where the applications will be executed and a client application that virtualises the applications on the user's mobile device.

In this document some solutions that aim to increase the security in Android devices were reviewed. Some concepts of the Android operating system were also introduced in order to justify the decisions taken during the implementation of the solution.

The approaches tested during the implementation of the system were explained and, in the case of problematic approaches, the problems encountered and the reason for abandoning these initiatives were justified.

The implementation of the final approach was described, both in the server the client application component.

The solution presented is able to virtualize the applications that run on a server, in a client application on the user’s mobile device, fulfilling the initial objective of presenting a system that offers a more secure environment for users of mobile devices to use applications.

The proposed solution also presents features that can bring advantages over the other VMI solutions. The ability to offer to users of non-Android mobile devices, the use of Android applications with a native user interface, to which they are accustomed with is one of these points. Another advantage is that the proposed system through has lower communication requirements (e.g. bandwidth and amount of data transmitted), can a lower cost with mobile communications than others VMI solutions, in which the server sends images of the UI to the client application.

However, in order to provide users with a complete experience using the applications and to provide a secure use of the system, it is necessary to implement additional functionalities in the solution, since the implemented components were aimed to prove the possibility of virtualization of the test application.

B. Future Work

The solution was implemented as proof of concept of the feasibility and applicability of application virtualization through the interception of calls to the operating system libraries for mobile security. To make it usable in a real-world scenario, some functionalities must be implemented in the solution.

Since the system was implemented only with the type of views present in the test case application and that during the collection of the information, only the necessary information for the test case was collected. It is then necessary to implement all types of views and to collect all the information about the views. In order to virtualize all the applications it is necessary to implement a mechanism that allows the use of resources such as GPS, camera, etc. This mechanism can be implemented in the transact function of the IPCThreadState class of Libbinder.

A mechanism that allows access to multiple users and that provides a workspace to each user should be implemented since the proposed system contemplates this possibility. Different users should be isolated on the server through a sandbox.

Users must be authenticated by the system in order to be able to use it. An authentication system must be implemented. Solutions to this point are given in section IV.F. Like authentication, the protection of the communication between the server and the client application should be assured. Solutions to this point are given in section IV.F.

The features mentioned above are the essential features for the system to provide a secure environment to users and to virtualize a large number of applications.

The implementation of a gateway that controls information passed between the server and the application or the implementation of a tool that scans users' devices for malware, with their permission, are examples of measures that could be implemented to complement the solution.

VII. REFERENCES


