# IMPLEMENTATION OF APPLICATION THAT DEMONSTRATES MOBILE APPLICATION VULNERABILITIES

**Karolína Šrůtková**

Master Degree Programme (2), FEEC BUT

E-mail: xsrutk02@stud.feec.vutbr.cz

Supervised by: Zdeněk Martinásek

E-mail: martinasek@feec.vutbr.cz

**Abstract**: Nowadays, using mobile applications is a daily routine for all of us. This trend is misused by hackers who attack all kinds of mobile apps from which they can get not only sensitive data of users. The main contribution of this paper is a design and implementation of a mobile application that demonstrates mobile application vulnerabilities of Android mobile operating system and that could help developers to build more secure applications.

**Keywords**: Android, mobile application, vulnerability

## 1 INTRODUCTION

During the day users of the smartphones use several types of mobile applications that make their work, school and everyday life easier. This fact should lead to a greater emphasis on user security through the implementation of security features in both mobile operating systems and mobile applications. The growing trend of using mobile applications is a good thing for attackers who are trying to exploit their vulnerabilities and obtain any user data, mostly sensitive one. The main benefit of this article a design and implementation of a mobile application that demonstrates mobile application vulnerabilities on the Android operating system. This implemented application brings to developers a possibility to explore different vulnerabilities in real time and in real environment. Understanding the principle of vulnerability and its exploitation leads to the implementation of countermeasures and therefore the resulting application is safer.

## 2 CURRENT VULNERABILITIES OF MOBILE APPLICATIONS

Like any other operating system, Android has its vulnerabilities. Its biggest vulnerabilities include the vulnerability of exported activity, missing authorization, the vulnerability of broadcast receivers and services, the vulnerability of internal storage and unauthorized access to data.

***The vulnerability of the exported activity*** means that when the exported activity is used, then all components of this activity can be accessed by any other application, and thus sensitive data can be easily obtained and processed by the application. ***The missing authorization*** is mostly mentioned in connection with API (Application Programming Interaction) calls and with Insecure Direct Object References (IDOR) vulnerability. IDOR vulnerability takes advantage of the fact that objects are accessed directly, most often using the user's ID (Identification). In an incorrectly implemented API request with method GET, it is enough to change the user ID in the URL (Uniform Resource Locator) path and the attacker will obtain information about any user.

The vulnerability of the other two basic components of the application, ***broadcast receivers and services***, is also based on exportability, as in the case with activities. Moreover, incorrectly set permissions of these components will not ask for permission to access the application from another application. Files stored on ***the internal storage*** can be easily obtained by attackers if the content providers are not used to access the files, but the parameters are set directly to the files. These are

the MODE_WORLD_WRITEABLE parameters for the edit option and MODE_WORLD_READABLE for the read option. This type of implementation does not provide the possibility to restrict access to data to specific applications. ***Access to data*** in the application is controlled by content providers, which in case of poor implementation can be exploited by vulnerabilities SQL (Structured Query Language) Injection and Path Traversal. SQL Injection vulnerability can be exploited when reading data from the database in case the insertion of a specific SQL query into the input is not treated according to rules, which may lead to gain or modification of sensitive data. To exploit the Path Traversal vulnerability, the content provider methods for opening files *openFile ()* and *openAssetFile ()* must be poorly implemented. In these methods the Uniform Resource Identifier (URI) parameters can be incorrectly verified and thus the attackers can gain unauthorized access to files and directories that are stored outside the root directory by specifying the absolute path to the file.

## 3 DESIGN OF THE MOBILE APPLICATION

The main benefit of this article is the design and implementation of a mobile application, which demonstrates the vulnerabilities of the mobile application mentioned above and vulnerabilities that are present in communication with the web server. The mobile application is being developed as part of the PenterepMail project, which is still under development and will serve as a complement to the ongoing Penterep project focused on penetration testing. Once the entire PenterepMail project is completed, it will contain a vulnerable web and mobile application in the form of a mail client. Both applications will communicate with the virtual server using the PenterepMail API. Figure 1 shows the communication blocks of all parts of the PenterepMail project. Further in the text, only the actual design of the mobile application is described.

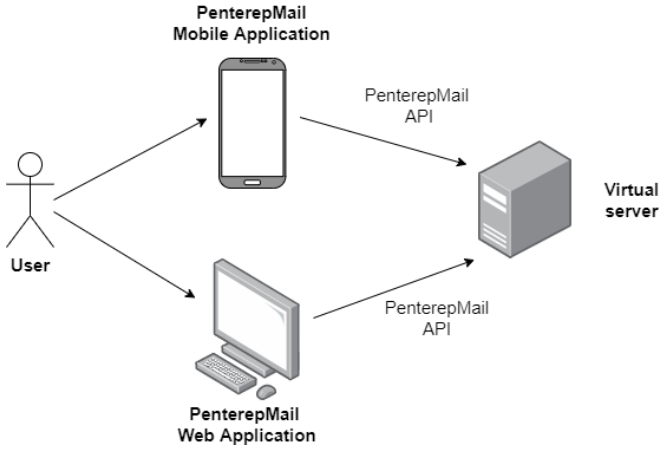


**Figure 1:** Communication blocks design of all parts of the PenterepMail project.

The mobile application is designed as a mail client. The user will be able to activate the application, log in and log out, view received and sent messages, send new messages and manage contacts, including their adding and removing. All these functionalities will be accompanied by the deliberate implementation of vulnerabilities based on the above analysis. The application is being developed in Android version 4.1 (API 16) for a clearer demonstration of vulnerabilities. Vulnerabilities will be also demonstrable on newer versions of the operating system, but not in a full scope.

After launching the mobile application, an activation screen will appear, through which the user must enter his phone number. The application sends an SMS to the entered number with a confirmation code, which was generated during the event for sending an SMS and which was saved in the application file. If the code stored in the application and the code sent in the SMS matches, the user is allowed to use the application.

The first screen after activation will be a welcome screen with options for login, for settings and for reading more information about how to use the application. Within the application settings, it will

be possible to set the IP address of the server in order to log in to the application. After logging in to the mobile application, the user will be able to view received and sent messages and add or remove contacts. It will also allow user to send new messages using the API and change the login details or IP address of the server. Screen designs before login and after login are available in Figure 2.

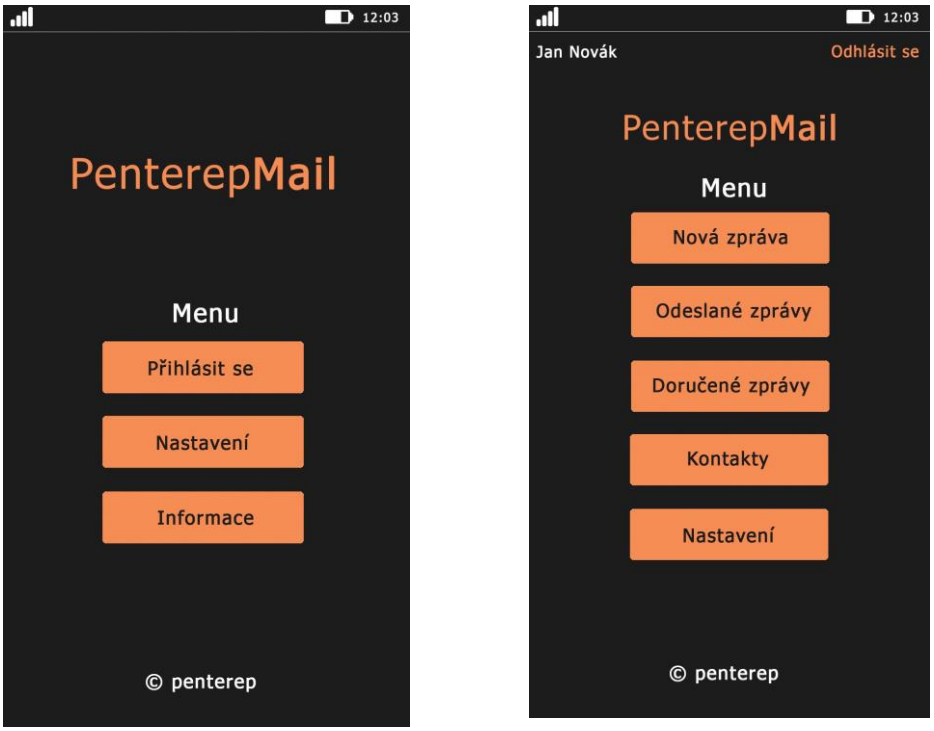


**Figure 2:** Screen designs before login (left) and after login (right).

The login credentials will be stored in the *settings.xml* file, which will be located in the internal storage. In addition, this file settings will have bad permissions, so it will be very easy to read its contents, including credentials. It will also be possible to access this file by exploiting the Path Traversal vulnerability. Registration will be allowed to the user only through the web application. Only a registered user will be able to log in to the mobile application. After a successful login, the user ID will be returned from the server with which the application will continue to work.

The application will be delivered as a trial version, from which it will not be possible to send messages. Messages can only be sent after entering the license key. The "New Message" activity will only run when the license is activated. However, the *settings.xml* file will contain a boolean value that indicates whether the application license is already activated. Due to the wrong permissions of the *settings.xml* file, the user can change this value from *false* to *true* and therefore achieve activation of the application license. It will also be possible to change the value while debugging the application. Another approach to bypassing the license will be a possibility to decompile the application. After decompilation, the user will be able to change the condition for starting the activity "New message" and after the subsequent compilation it will be possible to send messages also in the trial version of the application. The license key will not be provided to the user in any way. The user either bypasses entering the key using the mentioned steps above or finds it directly in the application code. In the code, the entered license key will be verified against the SHA1 hash, which is very easy to crack.

Contacts stored via the mobile application will be stored in a database in the device's internal storage. The database will be encrypted, but the password will be stored directly in the application code. An unexported content provider will be used to access the contacts and will be vulnerable to SQL Injection. The application will allow user to download attachments to the device's internal storage. Access to the files will be through a content provider which will be vulnerable to Path Tra-

versal. Sending messages will be solved by broadcast receivers, which will be exported and will be without appropriate authorization. Adding and removing contacts will be handled through exported intents. Sending an intent from another application will allow user to add or remove contacts without knowing the login. The application will also implement exported services running in the background, which will trigger requests at given intervals to see if a new message has arrived.

## 4 CUSTOM IMPLEMENTATION OF THE MOBILE APPLICATION

The first step in developing the application was to implement an activation screen. The activation screen is used in the application to demonstrate the vulnerability of the exported broadcast receiver. After entering the phone number and confirming it, the application calls an exported broadcast receiver called *SendSMSReceiver*. Its task is to send an SMS with the generated activation code to the given phone number. In order to send this broadcast receiver, an intent with the selected action name SEND_SMS must be passed to it. This intent also contains extra data representing the entered phone number and the generated code.

To test this vulnerability, mobile application penetration testing tool called Drozer was used. The tool is controlled from the command line. With its help, it was possible to eavesdrop on the sent data and find out to which telephone number and with what content the message is sent. With the acquired knowledge, it was also possible to use this broadcast receiver to send a message to any number with any message. The test result can be seen in Figures 3 and 4 as the output of the Drozer tool. Figure 3 shows a list of exported broadcast receivers in a given package. Figure 4 shows a command after which a message is sent to the selected number with the selected content.

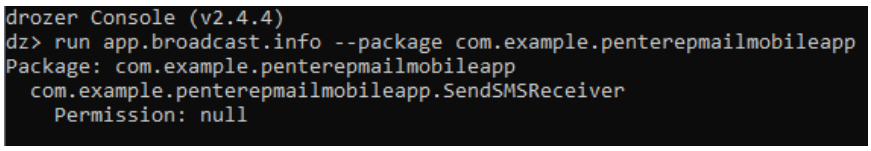


**Figure 3:** List of found exported broadcast receivers in a given package.

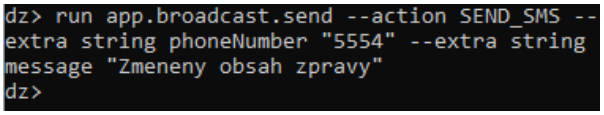


**Figure 4:** Command for sending message to the selected number with selected content.

## 5 CONCLUSION

The resulting implementation of the application will be used primarily by developers to explore the vulnerabilities of the mobile application in a real environment and time, because it is a fully functional mail application. They will be able to test what can be caused by poor implementation of individual components and what is the impact due to leakage or modification of user data. The created mobile application should primarily serve as a tool that will contribute to the creation of more secure applications.

## REFERENCES

[1] Kotipalli, S. R; Imran, M. A. Hacking Android. Birmingham: Packt Publishing Ltd., 2016. ISBN 978-1-78588-314-9.

[2] Nidecki, T. A. What Are Insecure Direct Object References [online]. 23. 3. 2020 [cit. 8. 3. 2021]. Dostupné z: https://www.acunetix.com/blog/web-security-zone/what-are-insecure-direct-object-references/

[3] Security tips [online]. Google Developers, 2020 [cit. 7. 3. 2021]. Dostupné z: https://developer.android.com/training/articles/security-tips