apriorit

Case Study: **E-commerce Android App Security Testing**

Application Description

> *The client company provides online ticket purchase services for the variety of entertainment and sport events. Mobile apps are a component of their service ecosystem allowing customers to review upcoming events, buy tickets, and view already purchased ones from their devices.*

Mobile Security Testing Task

> ***The Apriorit team was assigned to perform black box vulnerability testing of a new version of the company's Android app. Source code reverse engineering activities were not included into the project scope.***

Priorities

The Apriorit team was informed that the app did not store any financial information on devices, applying requests to the web service to perform purchases using financial data stored in the company's cloud data center.

Thus, the main priority was to check the protection of account access data in motion as well as at rest.

Research and Recommendations

At the first stage, Apriorit experts attempted to install the app on both unrooted and rooted devices. They discovered that some protection against installation on a rooted device was applied. At the same time, after being **installed on an unrooted device**, which was subsequently rooted, the app started normally.

This finding produced the first recommendation with the normal priority: to add protection not only from the installation but also from starting the app on a rooted device. It can be performed, for example, by checking if the SU daemon is installed at each app start.

The second stage was analysis of the traffic between a device and web server. Apriorit researchers discovered that while SSL certificate was used to protect the communication channel, it was still pretty vulnerable for the **man-in-the-middle attack**.

Using sniffers, researchers got access to the message bodies. It was discovered that access token and user credentials were sent in the plain text to the server. To illustrate the potential extend of this security risk, investigators created a small C# application for PC to use sniffed information and set up communication channel with the server under the "hacked" user account. This **"fake" desktop application** could review and cancel all purchases as well as perform purchases using the financial

details of the “hacked” user account. At the same time, it could not get the financial data itself because of the corresponding logic of the web service.

This risk obtained the highest priority. A series of recommendations were included into the report:

- Incorporate SSL pining technique to protect from the man-in-the-middle attack.
- Do not transfer credentials in the **plain open format**. Generate access key on the basis of credentials, e.g. hash, with additional salt and perform message bodies encryption using a symmetric algorithm.
- Incorporate device ID into the access key(s) generation to prevent usage of a “fake” application.

One more security risk was detected after performing Android backup acquisition for unrooted device and complete **file system acquisition** for a rooted device. In both cases researchers managed to get a cache database in the plain, unencrypted, **SQLlite** format. Besides partial information about the recent purchases, Apriorit experts managed to get user credentials and current access token stored in this database.

The risk of using the plain SQL format is also related to the possibility to easily **recover deleted data**.

This risk also obtained the high priority. The corresponding recommendations were included:

1) Not to store credentials locally; use runtime generated access key on the basis of credentials.
2) Apply additional encryption for all the databases produced by the app.

Discussion and Improvements

The Apriorit team produced a detailed report containing description of all performed activities and researched attack vectors. This report was delivered to the client and properly discussed with the in-house development team.

The client asked Apriorit to illustrate the recommendation about the new way of building access keys for authorization without storing the user credentials on device. A **prototype** was created representing the scheme of runtime access key generation based on the credentials, device ID, and request time. The credentials’ hash with added salt was used as a key for the AES encryption performed on the messages body within the device-server data exchange.

The client team developed a new version of the Android app using provided prototype logic and other security recommendations. This new version successfully passed the **second-round check**. The only remark was that it was still possible to run the app on a rooted device using specific third-party software. It remains one of the major security problems of the Android platform: while iOS users are forced to upgrade and use the latest platform version, Android users tend to **use “middle” versions of OS** and thus are not protected from the recently detected and fixed platform vulnerabilities, in particular a number of ways to root the device.

Results

Re-considering the recommendations and potential problem scope, the client decided that investments into the further rooting protection would not be efficient, giving that the new scheme of authorization and significantly improved device-server communication minimized security risks.

The complete Android app security testing with all rounds of discussion took 60 man-hours. The prototype of the recommended access scheme took additional 40 man-hours to be delivered.