SmartDec

Infinito Wallet mobile application security analysis

Abstract

In this report we consider the security of the Infinito Wallet project. Our task is to find and describe security issues in:

- The mobile application written for iOS and Android platforms
- Backend and API to the application

Procedure

In our analysis we consider the code of the mobile application, backend, and API (Source Code-20171214T113844Z-001.zip, md5sum 192d6e9cb97f39289a551294b5f173e7).

The latest version of the code

We have performed the check of the fixed vulnerabilities in the latest version of the code (InfinitoWallet.zip, md5sum c5e18ec36b58a2f505d5f0ab7d45381f). The critical vulnerabilities that have been found in the first version of the code were fixed.

Procedure

We perform our audit of the project according to the following procedure:

- We analyze the project in order to find the code borrowed from existing free and proprietary software.
- If third-party code is found:
    - The version of third-party code is determined
    - We search for the changes in third-party code
    - We analyze these changes
    - We use our knowledge base of vulnerabilities for code from existing software
    - A new code is isolated for further analysis
- We analyze a new code developed by customer:
    - This code is preliminary analyzed in order to configure the automated tools for finding vulnerabilities, bugs and other issues.
    - We analyze new code with third-party code analysis tools.
    - We analyze new code with our proprietary tools.
- Our experts manually check the issues found by code analysis tools.
- We manually analyze the code in order to get more accurate list of issues and develop recommendations for their elimination.
- We run compiled applications for iOS and Android platforms on devices in order to proceed with further security investigation of backend and API by using the dynamic methods of software analysis.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the application.

Checked vulnerabilities

We analyzed the project taking into account that the mobile application is implemented for iOS and Android platforms. Thus we also investigated the code for vulnerabilities specific for these mobile platforms. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Interworking vulnerabilities (unencrypted connections, requests to third-party web servers, TLS connection settings and certificate validity).
- Vulnerabilities according to The WASC Threat Classification, OWASP and OWASP Mobile Classification.
- Weak login / password pairs or brute-force attack possibilities
- Insecure data storage
- Inter-process communication vulnerabilities
- Cryptographic vulnerabilities (weak/self-deployed/out-of-date)
- Insufficient verification of input data
- Insufficient protection of application package: absence of obfuscation, absence of integrity checks, vulnerabilities associated with privileged unauthorized access to devices (like Jailbreak)

About the project

The project consists of mobile application, API and backend. The application has various functionality:

- Generating wallets with mobile stored private keys for ETH, BTC, BCH
- Displaying balances of wallets
- The user can add different tokens for ETH and show number of them
- The user can send ETH, BTC, BCH tokens to other addresses
- Creating a list of contacts

Backend with API allows:

- To get rates of tokens
- To receive BTC transaction to resend to node

Results of analysis

As a result of analysis we've found the issues including several critical vulnerabilities.
In the latest version of the code the critical vulnerabilities found during the first analysis have been fixed.
The concentration of issues is low for the industry. Found issues are split into two types. The first type of issues can potentially lead to malfunction of the application and to unauthorized access to user's confidential information. The second type of issues can't lead to these problems in the current configuration but they can be transformed from potential issues to real ones during a certain code extension. Hence the issues marked by our experts as "potential" or as "recommendations" should be eliminated in the future.
Successful attacks with an exploitation of found vulnerabilities can lead to the theft of confidential data, the loss of confidentiality of transmitted information, and the substitution of data for fraud.
All found vulnerabilities are described below in details with the information about severity, with the hacking example, and with the recommendations of how to eliminate them.

Critical issues

Critical issues seriously endanger security of the application. We highly recommend fixing them. We've found two critical vulnerabilities in the first version of the code, which have been fixed by the developers in the latest version of the code.

Medium severity issues

Medium severity issues can influence security of the application in current implementation. We highly recommend addressing them.

Unencrypted storage

The application uses unencrypted data storage.
In file **app/libs/db/DatabaseHelper.js:425**

```
// open the Realm with the latest schema
export default realm = new Realm(schemas[schemas.length - 1]);
```

We've found sensitive data in the local storage:

- Addresses of user's wallets. The attacker can get the list of user's addresses and thereby deanonymize it.
- Addresses of wallets, which the user has saved in the contact list. The attacker could replace these addresses in the data storage. The user may not notice the substitution and can send funds to the attacker's wallet.

Example of hacking: the attacker uses malicious software or social engineering methods to get unencrypted databases of the application from devices. After that the attacker can deanonymize wallets of users, steal confidential data or substitute addresses in contact lists.
Recommendations: use the encrypted storage.

Unlimited password entry attempts

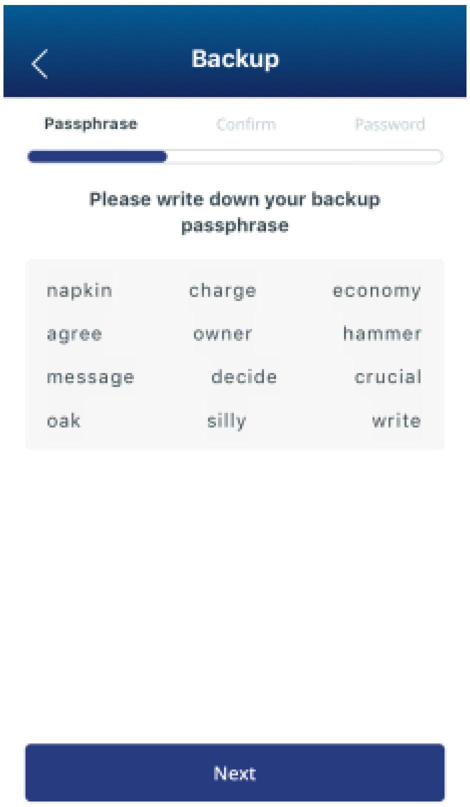
There is no limitation on password entry attempts in the application. Thus the application is vulnerable to brute-force attack.
Example of hacking: if the device is stolen by attacker, it's potentially possible to get into the user's wallet by password brute-force.
Recommendations: implement the limit on authentication attempts by password. When the limit is reached the application should use timeout for the next password try or CAPTCHA or both combined.

Lack of protection against unauthorized access to mobile device

The developer has included in Terms & Conditions the section in which described that the implementation of the application on the mobile device with unauthorized privileged access is under the end-user's responsibilities. We still recommend fixing the issue to increase the application security and physically restrict the implementation on the mobile device with unauthorized privileged access.
There is no protection against unauthorized privileged access to a mobile device in the application. A mobile device with changed security policies (like Jailbreak) can be modified by attacker. Such a mobile device should be considered as an insecure environment for a mobile application execution. The analyzed mobile application is potentially vulnerable due to lack of detection of unauthorized access to a mobile device.
Example of hacking: the attacker gets an access to the home directory of the mobile application which leads to sensitive data leaks.
Recommendations: implement the detection of unauthorized access to a mobile device. If unauthorized access is detected the application should exit or limit a set of provided services. We recommend using the unique combination of detection algorithms for each mobile application.

iOS background screen caching

The developer has included in Terms & Conditions the section in which described that the implementation of the application on the mobile device with unauthorized privileged access is under the end-user's responsibilities. We still recommend fixing the issue to increase the application security and physically restrict the implementation on the mobile device with unauthorized privileged access.
When the application goes to background mode, iOS makes an application screenshot. This screenshot is located in the home directory of the application in **Library/Caches/Snapshots/.** The screenshot can contain confidential data, which can lead to sensitive data leaks. The example of such a screenshot is shown below.



Please write down your backup passphrase

napkin	charge	economy
agree	owner	hammer
message	decide	crucial
oak	silly	write

Example of hacking: the attacker can get screenshots using a malware on mobile devices or personal computers.
Recommendation: disable this feature or hide sensitive data before going background.
Example of hacking: the source package can be arbitrarily modified by attacker. Then modified application can be distributed under the name of customer's application, or can be installed on user's device in case of unauthorized access to mobile device (like Jailbreak) and in case of physical access to device.
Recommendations: implement verification algorithms of source package integrity. Apply advanced methods of code obfuscation for such verification algorithms.

iOS application failure

When the user creates a new wallet, the application proposes to make a backup of a passphrase. Now if the user closes the application and shuts it down, the next time the application will be turned on, it will be in the disabled state. The only possible action for the user is to reinstall the application.
Recommendations: fix the shutdown of the application on the step of passphrase backup. This action can be found in file
app/components/create-restore-wallet/restore_input_passpharse.ios.js

Lack of authentication after background mode

If the iOS version of application is switched into background mode, there is no need to enter the password after recovery. Wherein, on Android platform user needs to do this.
Example of hacking: the user can turn of the application by “Home” button and expect the application to be logged out. If the attacker gets physical access to the device this time, it’s possible to enter the user’s wallet without authentication.
Recommendations: require a password after recovery from background mode or to warn user about the current behavior (for example, during the configuration of the application).

Low severity issues

Low severity issues can influence security of the application in future versions of code. We recommend taking them into account.

Vulnerabilities in third-party components

During the analysis of third-party components used in the application we’ve found the libraries with known vulnerabilities. In the current implementation all of these issues are unexploited, but we recommend paying attention on them in case of further development. The list of libraries is shown below.

Name	Version	Link to more info & fixes
fresh	0.3.0	https://nodesecurity.io/advisories/526
negotiator	0.5.3	https://nodesecurity.io/advisories/106
mime	1.3.4	https://nodesecurity.io/advisories/535
ws	2.3.1	https://nodesecurity.io/advisories/550
parsejson	0.0.3	https://nodesecurity.io/advisories/528
debug	2.2.0	https://nodesecurity.io/advisories/534

Recommendations

Self-developed keyboard implementation for Android platform

The standard keyboard of Android mobile platform is used in the application. Such an application is potentially vulnerable for Cloak & Dagger attack. These attacks abuse one or both of the SYSTEM_ALERT_WINDOW ("draw on top") and BIND_ACCESSIBILITY_SERVICE ("a11y"). If "draw on top" is permitted for the application, this permission is enough to lure the user into unknowingly enable a11y (through clickjacking).
Example of hacking: the attacker can make use of malware to record how the user presses the keyboard. More information about this vulnerability: http://cloak-and-dagger.org/
Recommendations: implement self-developed keyboard.

Codestyle issues

Codestyle issues influence code conciseness and readability and in some cases may lead to bugs in future. We recommend taking them into account. During the analysis of codestyle problems we've found JS style guide violations and errors: missed semicolons, unexpected compression type coercion, wrong JSDOC.
Recommendations: use any linter, e.g. Eslint with the default configuration for React native or some custom configuration - https://www.npmjs.com/package/eslint-config-react-native .

Analysis of the latest version

We've found a new functionality in the present version of the application.
We strongly recommend fully analyzing the latest version of the application

SSL report

During the analysis of interworking we've checked the security of SSL-connections. The report is shown below:

```
SSL Report: x.x.x (139.162.60.245) - A
SSL Report: api.etherscan.io (13.64.154.8) - B
      This server accepts RC4 cipher, but only with older
protocols. Grade capped to B.
SSL Report: x.x.x -  A+
SSL Report: x.x.x - A
```

The only potentially insecure connection calls to Etherscan service to take the courses of cryptocurrencies that does not threaten the application security.

Conclusion

In this report we have considered the security of Infinito Wallet mobile application, API, and backend. We performed our audit according to the procedure described above.
The audit showed critical issues which were fixed by the developer and several medium and low severity issues. We highly recommend addressing them. Part of the medium severity issues are conducted with usage of the application on the mobile device with unauthorized privileged access - the developer has included in Terms & Conditions the section in which described that the implementation of the application on the mobile device with unauthorized privileged access is under the end-user's responsibilities. Also, we recommend making a full audit of the latest version of the application.

This analysis was performed by SmartDec

COO Sergey Pavlin [illegible]

June 28, 2018