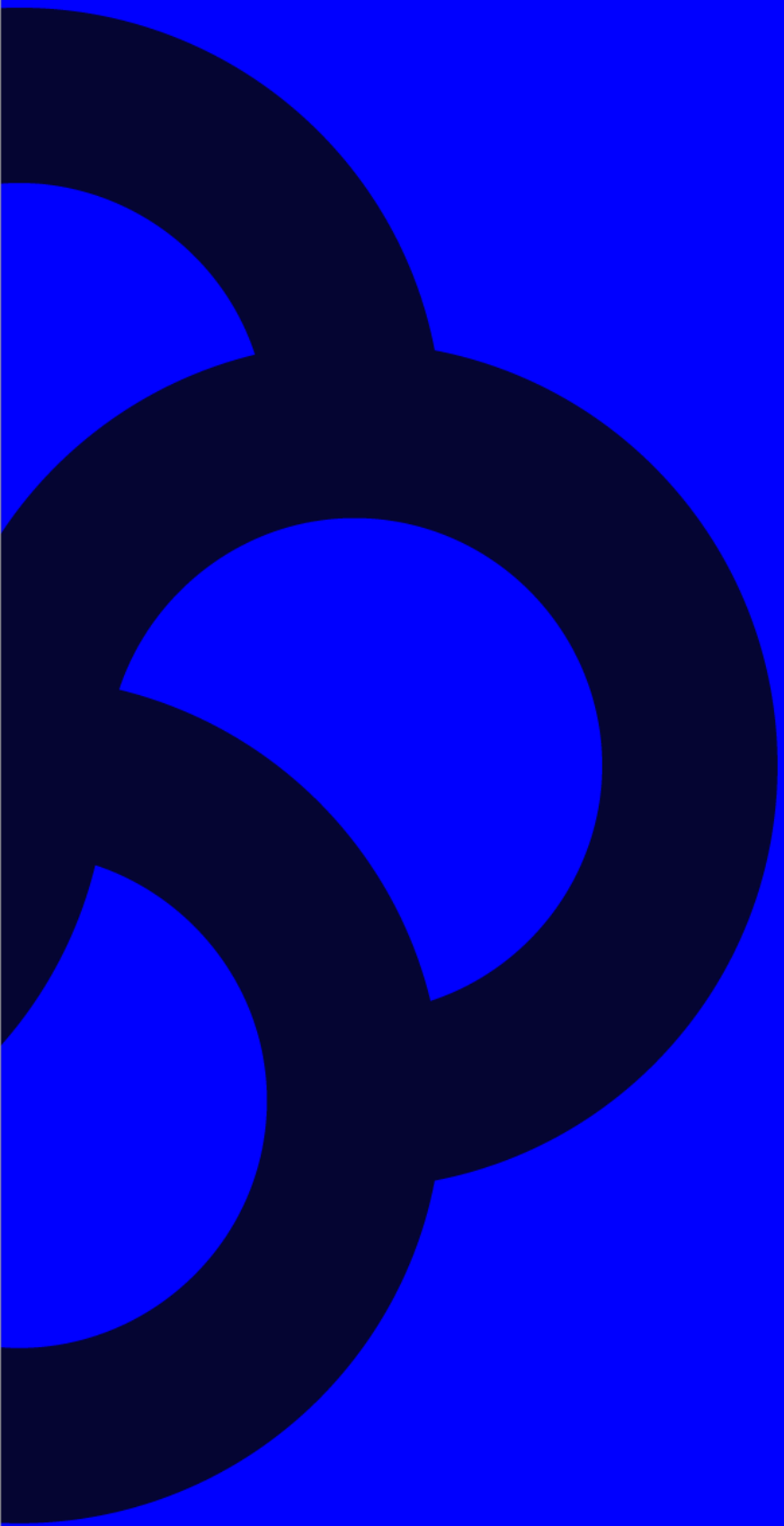
ORU

INFORMATION SECURITY

SECURITY REVIEW: ENJIN CRYPTO SMART WALLET



September 2018

FOR PUBLIC RELEASE

This document has been cleared for public release by:

Maxim Blagov, CEO of:
Enjin
16 Raffles Quay #33-03
Hong Leong Building
Singapore 048581

The purpose of penetration testing is to identify potential system vulnerabilities and generate a report on the detected vulnerabilities that will enable Enjin to rectify any defects in the security of the product and thereby increase the security level of the application.

Enjin understands that information security is by its nature continuously subject to change, and that the penetration testing performed by the consultants, and recommendations aimed at rectifying detected defects shall not imply that the application and the information system of Enjin is completely secure. Enjin understands that it is not possible to run all existing tests, and that it is impossible to test vulnerabilities in the software or hardware which were not known and/or available at the time of testing, i.e. it is not possible, in terms of time and practical considerations, to test each component of the system for all input/output values.

The public release of this document is only to state that security testing is performed against a specified testing object and to be transparent against Enjin's internal security practices.

This document is redacted in relation to the original, classified report. Only operational security info was removed (full request dumps, etc.), all vulnerabilities and issues are retained and no found vulnerabilities or issues were removed from this report.

Unauthorized disclosure, duplication, modification or use of this document without Enjin's permission is strictly forbidden and will be prosecuted to the fullest extent of the law.

Table of Content:

Report Summary

The purpose of this report is to present the findings that were discovered during the penetration testing engagement between Enjin and Oru, where the goal of this penetration test was to perform a penetration test against:

1. Enjin Crypto Smart Wallet for Android
2. Backend API that is used by Enjin Crypto Smart Wallet

The penetration test was performed as the second in succession external independent penetration test against Enjin's Android Smart Wallet, to verify implemented controls that were set up since the last third party external penetration testing and private internal security testing.

During our testing, we only found a few low risk issues that do not compromise the security of the wallet software or the signing keys, but should be implemented to aid in the defense in depth principle of the software.

The backend source code we analyzed is very well written, readable and uses current best practices. Although the API is critical to the operation of the wallet no critical security decisions are made on the API since it only serves as a connector to other systems and networks (Bitcoin, Ethereum, Litecoin…) and it relays the transactions that were generated on the mobile device. We found minor issues in the API, which mostly are tied to defensive programming practices.

Retesting of the testing objects after the penetration test showed that the Enjin team has addressed all low risk issues as stated in Appendix 1.

Overall Posture:

Our finding is that the overall security posture of the wallet application and the backend API is solid for the required risk profile and that the software is developed according to current security best practices which are required for a product that warrants a high level of security.

We found no issues that could be used to compromise the wallet or the cryptographic keys stored in the system in the scope of our penetration test.

Recommendation Summary:

We recommend that the Enjin team addresses the following low risk issues in the following order:

For the mobile wallet:

1. Missing tampering and debugging detection
2. Insufficient obfuscation in the APK
3. App logs information to Logcat
4. Insufficient Device Binding (Optional)

For the wallet backend API:

1. Implement specific whitelisting and validation for deterministic variables
2. No authentication between the wallet and the backend API (Optional)

Scope of testing

Penetration testing was performed by two consultants:

1. doc.dr.sc. Tonimir Kišasondi - Principal security researcher, Oru
2. Tomislav Turek, mag.inf. - Security researcher, Oru

From the Client side, the engagement was authorized by:

1. Maxim Blagov, CEO of Enjin
2. Witek Radomski, CTO of Enjin

The testing objects that were submitted by Enjin for the penetration test are:

1. Source code of the backend API service for the Enjin Crypto Smart Wallet with the commit hash `aee0eaa9940d6daaef095290a5de90cf284f8263`.
2. Active testing version of the API that was connected to the testing version of the Android APK (black box test).
3. Testing version 1.3.0-audit of the Android APK with certificate pinning and obfuscation removed to help with testing of specific attacks (black box test) with SHA256 hash:
    - `5b7843f818a06e75d413a566a210c53a768c5670b109a9d2eb7cd49f1e19efc5`

4. Release version 1.2.3 of the Android APK (available from: https://enjinwallet.io/apk.html) that was used to estimate and check and evaluate anti-debugging, anti-reverse engineering and anti-tampering controls (black box test) with SHA256 hash:
    - `e8e411eff11c3e24b9a6b08c82c78159bed9f56d8c21cbb5462fa45f417921b3`

Retesting of the testing objects after the penetration test showed that the Enjin team has addressed all low risk issues as stated in Appendix 1.

The objectives of this test were to simulate the following scenario and test the following hypothesis:

1. Simulate an attacker that can install the APK on his wallet and:
    a. Use the obtained knowledge from the installed APK to glean enough information to compromise a third party wallet on an unrooted, modern Android device with software based attacks against the wallet software.
    b. Use the obtained knowledge to attack the backend API and compromise the backend.
    c. Check if the there is any benefit to the attacker if he compromises the backend API or controls the backend API.
    d. Check for possible backdoors in the code or any other method that can be used as a backdoor.

Out of scope

- We did not focus on hardware level attacks like side channel attacks or attacks on hardware platforms.
- Our tests were carried out with an assumption that the underlying Android OS is running on a modern, hardware backed android keystore with an unrooted Android OS.
- We did not cover any attacks that assume the Android OS or the hardware/firmware of the platform is compromised.
- ARM assembly code and the source code for the wallet were not in scope.
- Connected APIs and services since the wallet API serves as a connector to other production APIs from third parties (Ethereum network, Bitcoin network, etc…)

Retest information:

Retesting the testing objects after the penetration test was done, shows that the Enjin team has addressed all found issues as stated in Appendix 1.

Penetration Testing Methodology

Oru's penetration testing approach is focused on optimizing the conducted tests and the time that is available to uncover security vulnerabilities in the testing object. Since it is not possible to run all possible tests, nor it is possible to test for vulnerabilities that are not known and/or available at the time of testing, nor it is possible in terms of time and practical considerations to test every component of the system for all possible input/output values and scenarios that could occur to test the entire attack surface, we tailor our testing to discover vulnerabilities that could have the most potential risk and impact, regarding a possible threat scenario, optimized for the ease of discovery by a third party. This enables us to discover vulnerabilities that have the biggest possible impact on the system, but are practical and realistic according to the threat model of the testing object.

Oru's penetration testing methodology relies upon following industry best practice standards:

Mobile Application penetration testing:

OWASP Mobile Top 10 - The Open Web Application Security Mobile Top 10 outlines the top 10 vulnerability classes on mobile devices, that were collected by the OWASP project's Mobile Top 10 team and graded to find the most prevalent security issues in mobile devices. This document helps us focus on the most pervasive, prevalent and detectable vulnerabilities in mobile applications. To read more about the OWASP Mobile Top 10 project, please refer to:

- https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10

OWASP Mobile Testing Guide - The OWASP Mobile Testing Guide defines the industry best practice mobile penetration testing methodology. The Testing Guide outlines both the Android and iOS specific testing procedures, with a highlight on:

- Resiliency against reverse engineering
- Architecture, design and threat modelling
- Data Storage and Privacy
- Cryptography
- Authentication and Session Management
- Network Communication
- Platform Interaction
- Code Quality and Build Settings

Depending on the available time for the test, we optimize to run tests that would have the most significant impact and could be discovered easily by an attacker. To read more about the OWASP Mobile Testing Guide and the mobile penetration testing methodology, please refer to:

- https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

Web Application penetration testing:

OWASP Top 10 - The Open Web Application Security Project's Top 10 document outlines the most critical web application security flaws for a specific year the document was created. According to the latest OWASP Top 10 document: "The OWASP Top 10 - 2017 is based primarily on 40+ data submissions from firms that specialize in application security and an industry survey that was completed by over 500 individuals. This data spans vulnerabilities gathered from hundreds of organizations and over 100,000 real-world applications and APIs. The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact". The value in using this document as a reference on the possible prevalence, exploitability, impact and detectability of a particular class of vulnerabilities, helps us focus on finding the most prevalent and detectable vulnerabilities that could have the most significant security impact. Uncovering the classes of vulnerabilities that are defined by the OWASP Top 10 set is our highest priority. To read more about the OWASP Top 10 project, please refer to:

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Testing Guide - The Open Web Application Security Project's Testing Guide document defines the industry best practice web application penetration testing methodology. The methodology is very detailed and outlines 91 tests in a total of 11 categories:

1. Information Gathering
2. Configuration and Deployment Management Testing
3. Identity Management Testing
4. Authentication Testing
5. Authorization Testing
6. Session Management Testing
7. Input Validation Testing
8. Error Handling
9. Cryptography
10. Business Logic Testing
11. Client Side Testing

Depending on the available time for the test, we optimize to run tests that would have the most significant impact and could be discovered easily by an attacker. To read more about the OWASP Testing guide and the application penetration testing methodology, please refer to:

- https://www.owasp.org/index.php/OWASP_Testing_Project

Vulnerability classification and ranking:

This report will use the following scheme for vulnerability marking:

- Description of the vulnerability (Risk Level)

The report is using the following risk ratings, marked with their respective colors:

High - High risk vulnerabilities enable attackers unauthorized access to the application as a privileged user and can have severe effects on the confidentiality, availability or integrity of data and the entire system.

Medium - Medium risk vulnerabilities enable attackers unauthorized access to the application as a normal user and can have negative effects on the confidentiality, availability or integrity of data in the system.

Low - Low risk vulnerabilities give attackers additional information or a good starting point to develop an attack in combination with other vulnerabilities

Informational - Informational statement, not a risk but might be interesting to consider and implement since it will improve the security posture.

Approval - If a statement is highlighted green, this means that the implementation follows best practices and is properly implemented.

Findings - Enjin Mobile Wallet Android application

Found issues:

Missing tampering and debugging detection (Low / Informational)

Both production APK and the provided testing object were missing APK modification detection measures thus enabling us to change the APK to a debuggable state, resign it and install it on the device. Along with the inability to detect that the APK has been changed, further debugging of the changed APK via debugging tools showed us that debugging detection is also missing. This does not compromise the security of the wallet in any way since an attacker cannot extract the signing keys, but we nevertheless suggest implementing this mitigation in order to make the wallet maximally resistant to reverse engineering and tampering.

Suggestions for mitigation

There is no complete security when tampering and debugging detection is implemented as it can still be bypassed by a very well versed adversary and those methods serve as a deterrent. We suggest implementing at least basic detection such as verifying the APK signature on the client side and implement checks if the APK is in debuggable state or a debugger is connected or if the application is being ran in an emulator to add more defense in depth. Upon detection, different actions can be taken such as alerting the backend service or not executing the app at all.

Insufficient obfuscation in the APK (Informational)

Both the provided testing object and the production APK did not implement advanced obfuscation techniques. Since a lot of logic is contained in native platform assembly language, an attacker needs to have specific, advanced skills in order to fully reverse engineer the application. Although, a small portion of logic is still contained in the APK and the implementation has little to no obfuscation.

Suggestions for mitigation

To achieve an additional level of obfuscation in the APK itself, use custom Proguard rules. An alternative can be using some commercial obfuscators instead of Proguard such as Dexguard.

Insufficient Device Binding (Informational)

There is no device-wallet binding implemented in the application, which means it is possible to copy data from one device to another device and change app data (specifically wallet data) with root level access. This can be used as a wallet lock bypass method which will only expose the owner's private information such as coins, balance and transactions. In case of a lock bypass, private key is still not exposed and cannot be used maliciously since the private key is stored in the Android Keystore and cryptographically tied to the old password which was swapped with a different one.

Replacing the app data poses no risk to the wallet private keys and only has an impact on the wallet's basic data, which might be a privacy issue, if for instance a third party wants to obtain info about the balance on a specific device, but since this is only a problem if a device is rooted, we consider it below low risk.

Note: Device binding would allow Enjin to correlate which installed wallet has access to what addresses. If Enjin decides to implement such measures, make sure to make deanonymization attacks harder.

App logs information to Logcat (Informational)

When the APK is in a debuggable state, some non-sensitive information gets logged into Logcat. Please keep in mind that it is not possible to view logs in Logcat if APK is not in debuggable state but the logs may still be written to the device. Also, we did not notice any sensitive information logged into Logcat.

Suggestions for mitigation

Before releasing the APK, remove logging calls. This can be done manually or, the simplest solution, use a custom Proguard rule which removes logging calls from code in production builds:

```
-assumenosideeffects class android.util.Log {
    public static boolean isLoggable(java.lang.String, int);
    public static int v(...);
    public static int i(...);
    public static int w(...);
    public static int d(...);
```

```
    public static int e(...);
    public static int wtf(...);
}
```

This way, the logging capability will stay in the debug builds but will be removed in release builds, hence no logs will be seen even if the release build is modified to a debuggable state.

Approvals:

In addition with found issues, we want to specifically outline some attacks and methods we tested, where we found that adequate security measures have been implemented:

TLS Interception Detection (Approval)

We've conducted tests in order to confirm that messages sent between client and server cannot be easily intercepted through a intercepting proxy and that certificate pinning is properly implemented. The production APK correctly implements certificate pinning.

Application is not debuggable and does not backup app data (Approval)

Several tests confirmed that the application is not in a debuggable state and does not backup app data, which means no sensitive information will be saved in the app data backups. To make sure this stays this way, fix the low risk issue "Missing tampering and debugging detection" since this can be modified by an attacker.

No information leakage in the app switcher (Approval)

During production APK use, we've observed that there is no possibility to view the contents of the wallet (total balance, list of coins and their balance) via the app switcher.

Root level access detection (Approval)

The production APK was installed on a rooted device. The application correctly detected that a device was rooted and showed a prompt with a notice to the user that it is not recommended to use the wallet on a rooted device.

Sensitive input cannot be cached or copied to clipboard (Approval)

Production APK showed that all sensitive inputs use the Enjin Secure Keyboard. All manual tests we performed for caching of input or copying input to the clipboard failed.

Cryptographic material is stored in Android Keystore (Approval)

Reverse engineered source code showed that all cryptographic key material is stored in the Android Keystore. This means that cryptographic operations are never done through the application process and keys may be bound to a Trusted Execution Environment of the device. We've tried to fetch the keys by using a custom piece of malware we developed and moving it to a system app after discovering the alias of the key in the Android Keystore. None of the apps (user or system) were able to fetch the cryptographic material.

No sensitive functionality exposed through IPC (Approval)

Through several tests, we've found that only the main activity is exported and cannot be launched or used by other applications. Other activities, services, providers or receivers are not exported. Also, no browseable activities were found which can be invoked from the browser.

No sensitive information found in memory (Approval)

We've conducted tests by dumping the application memory when performing sensitive actions (wallet unlock, wallet backup, ...) and searching for sensitive information such as passwords, cryptographic keys, etc. via the Android Profiler. We found no occurrences of sensitive information in memory dumps.

App overlay detection (Approval)

We've wrote a custom user and system app which simulates a malicious app that overlays the screen and collects user taps on the screen, which would enable an attacker to collect a wallet password. When opening the wallet, Enjin Wallet app detected a screen overlay and did not proceed until the screen overlay was shut down.

No ability to take screenshots when making backup (Approval)

On the production APK we've tried both taking screen captures and recordings. Regular user screenshots are blocked with a notification, but captures that run from an ADB shell are not blocked. Upon inspection of the made recordings, no app content was seen in the recordings, instead, only a black screen was shown on both image and video.

No severe issues found by automated scanners or manual review (Approval)

In order to find the most obvious issues and vulnerabilities, we have scanned the app with several automated scanners that are a part of our in-house solution for automatic source code and binary analysis and did a manual review of the application.

None of the scanners or our manual review found any severe issues that were identified as true positives, the scan results are available as an addition to this report.

Findings - Backend API

Found issues:

Implement specific whitelisting and validation for deterministic variables (Low/Informational)

Most of the parameters like addresses, amounts, transactions etc. used in the backend API are very deterministic. The current codebase does not perform whitelisting or validation on deterministic variables in the requests to the backend APIs and services. We suggest that at least the deterministic parts of the requests that can be easily validated or type checked should be verified before they are processed as a good defense in depth technique, to protect other connected API's. This is not a vulnerability, but we suggest implementing this for additional security.

No authentication between the wallet and the backend API (Low/Informational)

The current architecture of this solution has no coupling between the mobile wallet and the backend API. The wallet depends on the backend API to operate correctly, but the backend API does not "authenticate" a specific mobile wallet and can be used by almost anyone since there is no authentication towards the backend.

We suggest that at least a specific wallet identifies itself to the backend API with an API key that will be generated and enrolled on a specific wallet install. This will help track API usage patterns and ensure the fair use of the API.

Keep in mind such measures would allow Enjin to correlate which installed wallet has access to what addresses. If Enjin decides to implement such measures, make sure to make deanonymization attacks harder. Note that this is not a vulnerability.

Approvals:

In addition with found issues, we want to specifically outline some attacks and methods we tested, where we found that adequate security measures have been implemented:

No additional content or endpoints were discovered (Approval)

We attempted to discover additional services or applications that might help us expand the attack surface, but found no additional services or systems or misconfigurations on the test server that would help us gain more access.

TLS is enforced (Approval)

TLS use on the API is enforced by forcing access only via TLS (port 443) and blocking any access to port 80 or plain http.

No injection vulnerabilities found (Approval)

During our review, we found no vulnerabilities that stem from combining untrusted input with specific commands or queries.

No deserialization vulnerabilities found (Approval)

Application does unserialize untrusted input. JSON parsing is done with functions that do not support evaluation of arbitrary objects or code.

No XXE vulnerabilities found (Approval)

Application does not parse XML or use any XML specific parsing / including.

Note on XSS and CSRF (Approval)

The application is a public, unauthenticated backend API that's being consumed by a mobile application, where security and data centric decisions (transaction signing) are made on a the Android application. Since there is no storage layer that whose output be presented to the user in a browser, there in no risk of XSS vulnerabilities. Since the API is just a public connector to various blockchain based APIs, there is no CSRF issues either.

Note on access control, authorization and direct object references (Approval)

The API is public, and doesn't have any access control measures implemented by design, so there is no access control to test. Authorization is created on the mobile wallet, where a specific transaction will be signed. There are no direct object references, as the API is just a relay for other specific API's.

No severe issues found by automated scanners or manual review (Approval)

In order to find the most obvious issues and vulnerabilities, we have scanned the app with several automated scanners that are a part of our in-house solution for automatic source code and binary analysis and did a manual review of the application.

None of the scanners or our manual review found any severe issues that were identified as true positives, the scan results are available as an addition to this report.

About Oru & Contact information

This penetration test was conducted by the Oru Security team: doc.dr.sc. Tonimir Kišasondi and Tomislav Turek, mag.inf.

Tonimir Kišasondi is the Principal Security Researcher at Oru, he finished his PhD in the area of cryptanalysis at the University of Zagreb. From his industrial cooperation side, for the last 10 years he specializes in helping software, IoT and blockchain companies from the EU and US build secure products from the design to the production stage. His professional and research area of interest is security architecture, application security, security testing & analysis and applied cryptography.

Tomislav Turek is a Researcher at Oru, his masters specialization is in the area of security design patterns in applications, and has done his professional training and specialization in the Netherlands. His professional and research focus is in the area of secure architecture design, source code assessment and security reviews. When he is not testing code written by others, he likes to keep his Java, Go and Python dev skills current.

Contact Oru:

E-mail: tony@oru.hr ; tom@oru.hr
Web: www.oru.hr
GnuPG Key fingerprint: CC88 9E42 192A CE54 37DF 0B81 2D17 70B3 0F40 907D

Appendix 1.

Retest of the production API and testing/production APK shows that the following low risk issues are mitigated:

- Missing tampering and debugging detection (Low/Informational)
- Insufficient Device Binding (Informational)
- Insufficient obfuscation in the APK (Informational)
- Implement specific whitelisting and validation for deterministic variables