TARSKI TECHNOLOGIES

REVEALING TRUTH WITH DATA-DRIVEN EXPERIENCES™

WALLAWALLET SECURITY REPORT

PREPARED FOR
WALLAWALLET

SEP 15, 2020

WWW.TARSKI.TECH

WALLAWALLET SECURITY REPORT

This document provides a secure code review of the Wallawallet mobile H-Bar wallet, as defined below in the "Scope" section. The results of the audit are covered in this document.

I. INTRODUCTION

Aidan Noel, contracting through Tarski Technologies LLC, has conducted a secure code review of the Wallawallet mobile H-Bar wallet, as defined below in "Scope". The results of the audit are covered in this document. The code audit began the 28th of August 2020 and lasted until the 7th of September 2020. The objective of this assessment was to determine security weaknesses in the mobile application, with a focus on secure key management. Additionally, special attention was given to React Native vulnerabilities relevant to the application, as well as cryptographic standards and the implementations of cryptographic functions.

It should be noted that, while the review process will be as thorough as possible in discovering and reporting security vulnerabilities, it is not always guaranteed to find all possible security weaknesses. If no issues are found, this review does not certify that the application is 100% invulnerable to attacks. A third-party secure code review should be considered a strong addition to a company's overarching risk mitigation program, but not a "silver bullet" in the continuous risk mitigation process.

II. BACKGROUND

Wallawallet is a mobile cryptocurrency wallet developed by Ledgerama, LLC. The application is designed to manage accounts storing H-Bar, the cryptocurrency utilized in Hedera Hashgraph transactions. As such, the security of the application is key to guaranteeing the safety of users' personal financial assets. The Android version of the application is currently available in the Google Play store as an open beta release. After being granted access to the codebase for a few days, an initial meeting was held to discuss a general overview of the codebase focusing on the most relevant files to application security. We discussed vulnerabilities which could theoretically exist in the application, as well as some specific uses of cryptographic functions, to help focus the direction of the audit.

III. DISCLAIMER

This audit is not a security warrant, investment advice, or an endorsement of the Wallawallet application. The audit does not provide a security of correctness guarantee of the audited code. Securing cryptocurrency wallets is a multistep process, therefore running a bug bounty program in addition to this audit is always recommended.

IV. SCOPE

A secure code review was performed on the Wallawallet code base on Gitlab. The review included only mobile application code for the Android version of the application. This included two main com- ponents, with the second component being the primary focus of the assessment:

1. The React Native component handling user interaction.
2. The Java and Kotlin component handling cryptography, key management, and Hedera Hash- graph SDK interaction, all encapsulated as a React Native module.

V. ASSESSMENT

I. CRYPTOGRAPHY AND KEY MANAGEMENT

As the client's primary concern was secure key management in their application, this was the initial focus while reviewing the codebase. Symmetric encryption and decryption operations are performed with AES Galois/Counter Mode (GCM) and 16-byte initialization vectors (IVs). IVs are generated utilizing Java's built-in SecureRandom method, providing cryptographically strong pseudo-random number generation. This method is also used to securely generate the 32-byte application key. Asymmetric operations are performed with RSA SHA-256 with MGF1 padding. The cryptographic ciphers utilized in the application adhere to common security best practices. All sensitive material is encrypted and stored in the Android Keystore for retrieval according to best practices noted by F-Secure.[1]

Application keys are cached in memory until an authentication timeout limit is reached, as specified by the user in their settings. Data stored in memory is often vulnerable to several attacks. However, on mobile devices, this requires a malicious actor to either have physical access to the unlocked and authenticated device, for a malicious application to be installed on the device, or for malicious code to be executed on the device. The Wallawallet application largely protects from physical access, even in the case that the attacker has access to the user's fingerprint, with an optional 6-digit pin as well as an adjustable timeout on the cached application key. Physical attacks, such as RowHammer, could theoretically expose the application key, however this would require the attacker to gain physical access to the device and execute the attack successfully before the application key timeout is reached, which is unlikely.

RECOMMENDATIONS

I would recommend strict limitations on the range of timeouts for the application key, and strong insistence that the user utilize the 6-digit pin feature provided. It is the opinion of the auditor that the default timeout of 15 minutes should be considered an absolute maximum to mitigate the security risks. It would also be best to ask the user to disable Javascript auto-execution in their email client and browsers, especially if they are utilizing a client which does not disable it by default or while visiting potentially malicious websites on the device. This is very uncommon in email clients, but not unheard of. User education (or security awareness)) is extremely important in preventing attacks that could take advantage of the application key caching, however users of cryptocurrency

wallets are often more technically educated than the average user. Finally, I would recommend clearing the cached application key if the Wallawallet application is no longer in focus on the device to help protect from any malware seeking to dump the Wallawallet memory sectors while it is running in the background.

The application utilizes 16-byte IVs for AES-GCM cryptographic operations. However, NIST Standards recommend the use of 96-bit (12-byte) IVs when initializing an AES-GCM cipher.[2] Reducing the size of the IVs would increase performance and reduce the likelihood of a collision in the AES pre-counter block (J_0), as explained below:

IV Length is 96 Bits:

$J_0 = IV \| 031 \| 1$

IV Length is not 96 Bits:

let s = 128

len(IV)/128

-len(IV)

$J_0 = GHASH(IV \| 0^{s+64} \| [len(IV)]_{64})$ (2)

By using an IV with greater or less than 96 bits, AES-GCM utilized the GHASH function to instantiate the pre-counter block. This additional operation decreases performance and allows for the possibility of a collision in the pre-counter block where there was none with securely generated 96-bit IVs. Note that the severity of this collision is Low, and this recommendation is largely made with cryptographic standards and application efficiency in mind.

II. REACT NATIVE SECURITY

The React Native application components separate from the HederaUtil module were investigated for vulnerabilities with a focus on the OWASP Top Ten Application Security Risks.[3] This assessment found no vulnerabilities in the general React Native mobile application, with the exception of dependency vulnerabilities noted in the following section.

III. DEPENDENCY VULNERABILITY

Scans were performed with 'npm audit', retire.js, and Snyk to determine the security of third-party packages, as "Using Components with Known Vulnerabilities" is currently number 9 on the OWASP Top Ten Application Security Risks.[3] Such vulnerabilities have the potential to present additional attack vectors to malicious actors targeting users of the Wallawallet application with malware. Vulnerabilities were found within dependencies for 'react-native-level-fs@3.0.1', specifically 'bl@0.8.2' and 'semver@2.3.2'. Vulnerabilities reported range from Medium to High severity, and as they are vulnerabilities in a third-party package, the extent of the attack surface is difficult to determine within Wallawallet's implementation.

RECOMMENDATIONS

Snyk reports that both modules are sub-dependencies of the 'levelup@0.18.6' module, and both vulnerabilities can be resolved by upgrading the 'levelup' module to a more current version.[4] If upgrading this module breaks dependencies, it is recommended to contact the owner of the affected module and request that they update their packages.

V. CONCLUSION

Except for dependency vulnerabilities, attack vectors discovered in the Wallawallet application should be considered Low severity. A range of physical attacks are possible on the application; however, they require the device to be acquired and attacked while the user is authenticated in the application. This opportunity can be reduced by decreasing the timeout limit on the application key and utilizing the additional 6-digit pin feature. Malware based attacks on the application require a malicious application to be installed on the device, or some form of email
or browser-based script injection or remote code execution. Modern browsers and email clients largely prevent attacks of this kind, and mitigation is entirely dependent on a user's security awareness.

VI. ADDENDUM

As of September 15th, 2020, the client has appropriately addressed concerns listed in this Secure Code Review. This report has been discussed with the client and code changes have been reviewed and verified by the auditor.

VII. FOOTNOTE

1. https://labs.f-secure.com/blog/how-secure-is-your-android-keystore-authentication/
2. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
3. https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities
4. https://snyk.io/test/npm/react-native-level-fs