Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report KeePassium iOS Apps & Crypto 10.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. D. Bleichenbacher, Dr. N. Kobeissi

Index

- Introduction
- Scope
- Test Methodology
  - Scope Summary
  - Audit Phases and Tools Used
  - Cryptographic Analysis
  - Application Analysis
  - Reporting Methodology
- Identified Vulnerabilities
  - KEE-01-001 WP1: No strength checks on unlock passcode or passphrase (High)
  - KEE-01-002 WP1: Favicon fetch exposes account list to traffic analysis (Low)
  - KEE-01-003 WP1: Self-DoS through unreasonable key derivation settings (Low)
- Miscellaneous Issues
  - KEE-01-004 WP1: Evaluation of AESKDF (Info)
- Conclusions

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“KeePassium combines the security of KeePass with a clean intuitive design. You decide where you store your passwords. KeePassium helps you manage them.”

From https://keepassium.com/

This report describes the results of a penetration test and source code audit against the KeePassium mobile application for iOS, with additional focus placed on its cryptographic implementations, data storage, and network communications.

To give some context regarding the assignment’s origination and composition, KeePassium Labs contacted Cure53 in July 2024. The test execution was scheduled for late September / early October 2024, namely in CW40. A total of ten days were invested to reach the coverage expected for this project, and a team of three senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

The work was split into three separate work packages (WPs), defined as:

- **WP1**: White-box pen.-tests & source code audits against KeePassium iOS app
- **WP2**: White-box pen.-tests & source code audits against KeePassium sync
- **WP3**: White-box pen.-tests & source code audits against KeePassium crypto

All preparations were completed in September 2024, specifically during CW39, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of KeePassium and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-prepared and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates and shared their findings through the aforementioned Slack channel. Live reporting was not specifically requested for this audit.

The Cure53 team achieved good coverage over the scope items, and identified a total of four findings. Of the four security-related findings, three were classified as security vulnerabilities, and one was categorized as a general weakness with lower exploitation potential.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The small number of findings made in this report speaks for itself when considering the security of the KeePassium iOS application and its cryptography. However, despite the app and components clearly having been designed with security in mind, Cure53 was still able to uncover a few issues, and it is advised that these need to be addressed before an excellent level of security can be achieved. Specifically, one of these vulnerabilities related to missing strength checks on the unlock passcode or passphrase, and was ranked with a *High* severity (KEE-01-001). It is recommended that this vulnerability in particular should be resolved as soon as possible. Cure53 is certain that by addressing and resolving all of the issues detailed herein, the KeePassium team will further strengthen the KeePassium iOS application and further improve its already quite positive level of security.

The report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. This details which areas of the software in scope have been covered and what tests have been executed, despite only limited findings having been made during testing. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities*, and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the KeePassium mobile application for iOS, including its cryptographic implementation and features.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Pen.-tests & code audits against KeePassium iOS app & crypto**
  - **WP1:** White-box pen.-tests & source code audits against KeePassium iOS app
    - **Key focus:**
      - Password generation
      - Database storage, secure unlock
      - Passphrase setting, vault creation, vault security via keys, physical hardware YubiKey
      - Premium features (Have I Been Pwned, Favicon fetching, etc.)
      - Implementation of KeePass spec
    - **Source code:**
      - https://github.com/keepassium/KeePassium-internal/releases/tag/dev-1.53.154
  - **WP2:** White-box pen.-tests & source code audits against KeePassium sync
    - **Key focus:**
      - Two-way synchronization features via cloud storage
      - Importing and exporting vaults
  - **WP3:** White-box pen.-tests & source code audits against KeePassium crypto
    - **Key focus:**
      - Determine if cryptographic primitives are appropriate for the given purpose and follow common practices
      - Determine if the use of the cryptographic primitives is following the restrictions of the protocols
      - Check if the inclusion of the legacy encryption mode allows any fallback or key confusion attacks
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

This security audit of the KeePassium password manager, which implements the KeePass standard for iOS devices, aimed to thoroughly evaluate the application's security posture, focusing on cryptographic implementations, data storage, network communications, and overall resilience against common attack vectors. The audit was conducted by examining both the application's source code and its runtime behavior on iOS devices. This methodology section outlines the approach taken during the audit.

Scope Summary

The audit covered the following key areas:

- **Cryptographic primitives and protocols:** Evaluation of cryptographic algorithms, password hashing functions, key derivation functions, hash functions, encryption ciphers, and cryptographically secure pseudorandom number generators.
- **Data encryption and storage:** Analysis of how sensitive data - including vault contents and user credentials - are encrypted and stored on the device.
- **Authentication mechanisms:** Examination of password strength enforcement, unlock passcode security, and challenge-response mechanisms with hardware tokens (e.g., YubiKey).
- **Network communications:** Inspection of all network interactions, integration with external services (Have I Been Pwned, favicon fetching, etc.), and potential exposure through traffic analysis.
- **User interface and experience:** Assessment of the application's UX for potential security implications, including user guidance on security settings and feedback mechanisms.

Audit Phases and Tools Used

The audit was comprised of the following phases:

- **Initial code review**: A static, manual analysis of the source code was undertaken to pinpoint potential vulnerabilities in cryptographic implementations, data handling, and network code.
- **Threat modeling**: Potential attack vectors were identified by considering user expectations of password manager applications, the application's architecture, and adherence to external standards (KeePass).
- **Penetration testing**: Simulated attacks were conducted to assess the application's resilience against potential threats.
- **Reporting**: A comprehensive report was prepared, documenting findings, including PoC exploits, an impact assessment, and recommendations.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Charles Proxy[1] was the sole instrument utilized for the purpose of capturing and examining the network communications occurring between the application and external entities.

Cryptographic Analysis

In the following section, Cure53 will delve into a detailed examination of the cryptographic analysis that was performed. This analysis was systematically partitioned into three distinct phases.

- **Primitives evaluated:**
  - **AES:** KDBX3 uses AES-CBC with PKCS #7 padding. Since the integrity is checked after decryption, this in principle allows padding oracle attacks. However, the given use case makes chosen ciphertext attacks difficult to perform. Hence chosen ciphertext attacks are not a significant threat.
  - **ChaCha20:** ChaCha20 is a stream cipher. One important observation is that using the authentication method of the KDBX3 format would be weakened by the use of a stream cipher. Hence, an important check was to verify that ChaCha20 cannot be accidentally used with the KDBX3 format. It has to be KDBX4 only (as designed and implemented). ChaCha20 is frequently used together with Poly1305 for authentication. Using HMAC instead of Poly1305 makes the protocols slightly more robust against implementation errors, since a nonce reuse for Poly1305 leads to a key leakage, whereas the same is not the case for HMAC.
  - **Twofish:** Since Twofish is a block cipher with similar properties as AES it can be used in all places of the protocols where AES is acceptable.
  - **HMAC:** HMAC is generally a very robust cryptographic primitive that leaves very little space for errors. One such small pitfall is that HMAC computes the hash of its key when the key is larger than the block size of the underlying hash algorithm. This can lead to key leakages when an application uses key hashes, since in the worst case a key hash can expose the effective HMAC key. An evaluation of the code and protocols did not uncover any such issues.
- **Key derivation functions:**
  - **AesKdf:** This key derivation method was used in KDBX3. As pointed out in the documentation this function is not memory-hard. On modern CPUs it does not use the AES instruction pipelines optimally, meaning that an attacker can already gain a speed advantage from evaluating multiple key derivation functions concurrently. One property that was analyzed is the observation that two halves of the input can be derived independently of each other. The input of AesKdf is fortunately a hash value, so this property does not negatively impact the protocol.
  - **Argon2:** KDBX4 uses Argon2 instead of AesKdf. This is a significant improvement over the previous encryption format. The use of Argon2 in the protocol is appropriate and no issues were found.

[1] https://www.charlesproxy.com/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

- **KDBX3 vs. KDBX4:** The KDBX4 format has significant advantages over the KDBX3 format. Whenever an implementation contains multiple different modes it is important to ensure that the legacy modes do not negatively impact the newer modes. During the audit, it was checked that the two modes cannot be mixed, and in particular that no algorithms deprecated in KDBX4 can be used for said mode. Weaknesses in the KDBX3 format are sometimes countered by making restrictive assumptions on the capabilities of a potential adversary. For example padding oracle attacks are considered impractical, since they require a significant number of queries[2]. Because of the improved design of KDBX4, no such restrictive assumptions are necessary in order to analyze the protocol and its implementation.

[2] https://github.com/keepassxreboot/keepassxc/issues/147

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Application Analysis

The audit began with an examination of KeePassium's authentication mechanisms. Cure53 tested the application's ability to enforce password strength, by attempting to set weak master passphrases and unlock passcodes, including single-character passwords. It was observed that the application permits such weak credentials without any warnings or restrictions. This lack of enforcement on password complexity poses significant security risks, as it allows users to secure their vaults with easily-guessable passwords, making them vulnerable to unauthorized access through brute-force or guessing attacks.

Further analysis was conducted on the implementation of challenge-response mechanisms, particularly the integration with hardware security modules (HSMs) like YubiKey. By reviewing the source code files *ChallengeResponseManager.swift* and *YubiKeyUSB.swift*, Cure35 assessed how the application interfaces with these devices. The focus was on identifying potential vulnerabilities or misuses in the implementation that could compromise the security benefits provided by hardware tokens. The assessment ensured that the challenge-response protocol was correctly implemented, and that the cryptographic operations were securely executed.

In addition, the application's handling of Argon2id parameter settings was explored. By attempting to set extreme values for the memory cost, iterations, and parallelism parameters through the user interface, Cure35 observed the application's behavior under unreasonable settings. It was noted that the application allows users to input excessively high values without validation or warning. This can lead to self-inflicted denial-of-service (DoS), where the application becomes unresponsive or consumes excessive resources during the key derivation process, potentially locking users out of their own vaults.

The network communication aspects of KeePassium were scrutinized to assess potential privacy risks. Specifically, the favicon fetching feature was examined by monitoring network requests made during bulk favicon downloads. It was identified that the application initiates simultaneous requests to various domains associated with the user's saved login entries. This behavior can inadvertently expose the user's list of accounts through traffic analysis, as a passive network observer could compile a profile of the services the user has accounts with, based on the outbound requests.

Cure35 also evaluated the application's integration with external services, such as the Have I Been Pwned (HIBP) API. The assessment focused on ensuring that user privacy is maintained during breach checks. Cure35 verified that the application implements the HIBP API in a manner that does not disclose sensitive information, such as full passwords or email addresses, and that it follows best practices for anonymity and data minimization.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Furthermore, Cure35 inspected the use of secure connections (HTTPS) for all network communications, to assess the potential for eavesdropping or manipulator-in-the-middle (MitM) attacks. The analysis confirmed that the application employs proper encryption protocols to protect data in transit. Cure35 also evaluated the certificate validation processes to ensure that the application is resistant to SSL stripping or certificate spoofing attacks.

The user interface and overall user experience were examined with a focus on security-related configurations and feedback mechanisms. Cure35 reviewed the UI for setting key derivation function (KDF) parameters, password generation options, and security settings. The assessment checked whether the application provides users with guidance, warnings, or constraints when configuring security-sensitive parameters. It was found that the application lacks sufficient user guidance and does not enforce reasonable limits on critical settings like Argon2id parameters, which can impact both security and usability.

In terms of error handling and feedback, the application was tested to check responses to various error conditions, such as invalid inputs or failed decryption attempts. It was important to ensure that error messages are informative enough to aid the user without revealing sensitive information that could be exploited by an attacker. The application was evaluated on whether it handles errors gracefully, maintains application stability, and protects against information leakage through error messages.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., KEE-01-001) to facilitate any future follow-up correspondence.

KEE-01-001 WP1: No strength checks on unlock passcode or passphrase *(High)*

KeePassium allows users to set a master passphrase for their vaults and an unlock passcode for quickly accessing the app. However, the application does not enforce any strength checks or minimum requirements on these credentials. Users can set extremely weak passphrases or passcodes - such as a single character, e.g. “1” - without receiving any warnings or strength indicators.

This lack of validation poses a significant security risk. Weak passphrases and passcodes are more susceptible to brute-force attacks, potentially allowing unauthorized individuals to access sensitive data stored within the vault. Given that password managers often contain a user’s entire suite of credentials, the compromise of a vault can lead to widespread unauthorized access to the user’s accounts and personal information.

It is recommended that password strength validation checks be implemented both for the master passphrase and unlock passcode. This includes enforcing minimum length requirements as well as using the *zxcvbn-iOS* password strength checking library that KeePassium already employs across the rest of the application for measuring the strength of password entries for login items.

KEE-01-002 WP1: Favicon fetch exposes account list to traffic analysis *(Low)*

KeePassium includes a feature that fetches favicons for entries stored in a user’s vault. When this feature is activated, the application attempts to download all favicons associated with the login entries at once. This mass retrieval process can inadvertently expose sensitive information. Since the favicon fetches result in a number of outbound network requests to the respective websites of stored vault entries, in quick succession, a passive network observer could monitor the request and compile a list of all the websites for which the user has stored credentials.

While the data transmitted does not include actual credentials or sensitive content, the mere knowledge of which services a user has accounts with can be considered to be sensitive information, and could lead to the profiling of the user’s online activities.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

A proof of concept of this issue was verified by setting up a MitM proxy on a test iOS device and profiling network requests when the KeePassium application was made to request favicons.

While the impact is considered low because the issue requires network monitoring capabilities and does not expose actual credentials, it nonetheless undermines user privacy and could facilitate future attacks.

The following recommendations could be implemented in order to help mitigate this issue:

- **Improve user awareness:** warn users about the potential for network profiling before activating the mass collection of favicons.
- **Individual fetching:** fetch favicons individually, if at all, when the user accesses a specific entry.

KEE-01-003 WP1: Self-DoS through unreasonable key derivation settings *(Low)*

KeePassium allows users to configure the parameters of the Argon2id password hashing function when creating or accessing vaults. Argon2id is a memory-hard function designed to resist brute-force attacks by requiring significant computational resources for password hashing.

While flexibility in setting these parameters can enhance security, the application currently lacks reasonable upper and lower limits on these settings, leading to users potentially inadvertently setting Argon2id parameters - such as memory cost, iterations and parallelism - to excessively high values. This can cause the application to become unresponsive, crash, or take an unreasonably long time to open a vault.

This issue poses a self-inflicted DoS risk. Users may lock themselves out of their vaults. Additionally, when attempting to open existing vaults with overly-aggressive Argon2id settings, the app (especially given smartphone hardware) may fail to complete the key derivation process, rendering the vault inaccessible.

The following recommendations could be implemented in order to help mitigate this issue:

- **Implement upper limits:** Cap memory cost at a reasonable fraction of the device's available RAM, limit iterations to values that can complete within less than a minute, and restrict parallelism to the number of logical CPU cores or a similarly sensible maximum.
- **Implement lower limits:** Similarly, enforce minimum values to maintain adequate security.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit, but which may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

KEE-01-004 WP1: Evaluation of AESKDF *(Info)*

The KeePassium application uses AESKDF as a key derivation function for the KDBX3 format. Since this format has been deprecated, it is only used when the user explicitly enabled this feature for compatibility. A stronger key derivation (Argon2) is used by default.

Cure53 noticed that this key derivation function has some properties which may not have been published before. Notably, the key derivation function computes two halves of the result independently from the two halves of its input. This property makes it essential that the input of the key derivation mixes its components (password, local keys, etc.), before applying AESKDF. It has been verified that such a mixing step is indeed present in the application.

Another weakness of the key derivation function is that it not only is not memory-hard, but that it does not use the AES instruction pipeline optimally. This means that legitimate users suffer a time penalty that an attacker does not have.

A final note is that attackers can always try to attack the weakest link to try to find a password. Hence in the presence of KDBX3 and KDBX4-encrypted volumes with the same password, everything is only as secure as the KDBX3-encrypted volumes. Thus, exporting a volume to the old format without changing the password may be a potential security risk.

On the positive side Cure53 found that there are no additional issues that go beyond the support of a legacy feature. It is recommended that all users of the application transition to the new format.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

As noted in the *Introduction*, this late September / early October 2024 penetration test and source code audit was conducted by Cure53 against the KeePassium mobile application for iOS, with additional focus placed on its cryptographic implementations, data storage, and network communications.

From a contextual perspective, ten working days were allocated to reach the coverage expected for this project. The methodology used conformed to a white-box strategy, and a team of three senior testers was assigned to the project's preparation, execution, and finalization.

This security audit of KeePassium comprehensively evaluated the application's cryptographic implementations, authentication mechanisms, data storage practices, network communications, and user interface, for potential vulnerabilities.

During the auditing process, the team compared the encryption modes for KDBX3 and KDBX4. While KDBX3 does have some shortcomings, it has been confirmed that KDBX4 solves these issues, and hence provides a more robust solution. The cryptographic primitives used in the newer format of the application were found to be appropriate. Care has clearly been taken to ensure that the transition from one format to another does not introduce vulnerabilities such as key confusion attacks.

Nevertheless, this audit still identified several areas where security and privacy could be enhanced in order to better protect users. Overall, while KeePassium employs strong cryptographic primitives, certain implementation choices and lack of user guidance present risks that it is advisable to address, in order to improve the application's security posture.

KEE-01-001 showed that KeePassium does not enforce strength checks on master passphrases or unlock passcodes, allowing users to set extremely weak credentials (such as single-character passwords). This lack of validation poses a significant security risk, as weak passwords can be easily guessed or brute-forced, potentially compromising the entire vault. It is recommended that password strength validation should be implemented, and that real-time feedback should be provided to users. This would mitigate this *High* severity issue and enhance overall security.

KEE-01-002 showed that the application's bulk favicon fetching feature exposes the user's list of accounts to potential traffic analysis. By sending simultaneous requests to all associated domains, a passive network observer could identify the websites for which the user has saved credentials, thereby compromising user privacy. In order to address this issue, it is recommended that the favicon fetching process should be redesigned. This might include fetching icons on-demand or through a proxy, in order to minimize the risk of exposing sensitive information.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

KEE-01-003 showed that KeePassium allows users to set Argon2id key derivation parameters without reasonable limits, enabling configurations that could cause excessive resource consumption or application unresponsiveness. Users might inadvertently lock themselves out of their vaults, by setting parameters that exceed their device's capabilities, leading to a self-inflicted DoS. It is recommended that sensible upper and lower limits should be imposed on Argon2id parameters, and that user guidance should be provided. This would prevent the DoS scenario described above, improving both security and usability.

All in all, the KeePassium iOS application was found to incorporate a good level of security at the time of testing. However, this audit found issues in a few areas, which it is advised require attention and should be resolved.

Cure53 would like to thank Andrei Popleteev from the KeePassium Labs team for his excellent project coordination, support and assistance, both before and during this assignment.