Ywallet Security and Privacy Analysis

Taylor Hornby
zecsec@defuse.ca
ZecSec

Delivered: October 28, 2022
Updated: January 3, 2023
FINAL-v3

Contents

1 Introduction

This report documents a 13-day security review of Ywallet [2] that was performed for the Zcash Ecosystem Security grant [4].

Ywallet is a cryptocurrency wallet that supports both the Zcash and Ycash blockchains. It is known for its faster sync times using the "warp sync" algorithm as well as having a broader set of features. Ywallet is mainly written in Rust and Dart, and supports both iOS and Android using Flutter.

Unlike ZecWallet-lite, Nighthawk, and the wallets developed by Electric Coin Co, Ywallet is not based on Electric Coin Co's SDKs and implements its own syncing, scanning, and state management algorithms. A major focus of this audit was to find security and privacy vulnerabilities in this new codebase.

This review found no critical-severity issues, one high-severity issue, which is a problem in the memo-based contact storage system which allows an attacker to intercept users' messages, two medium-severity issues and six low-severity issues, described below.

Our main recommendation for future Ywallet development is to implement integration tests of the wallet's state management using "darksidewalletd". We also recommend that a ZIP standard for authenticated memos should be produced within the Zcash community.

1.1 Scope

The following GitHub repositories were included in the scope of this audit. Next to each repository name is the commit hash that was reviewed.

- `hhanh00/zwallet` - `25462bbf48e5635364b40fc01140e98f11ea1907`
- `hhanh00/zcash-params` - `0f1975b0d8799d852fe62e717dad8885eec1f106`
- `hhanh00/zcash-sync` - `3d5becd20e2a8e7167ffe909ee939ee25f0fc1c2`
- `hhanh00/jubjub` - `d8abaa3124ac7344a72b32dc20e946b4736ed50d`
- `hhanh00/librustzcash` - `625a06128659d011881698ec13edb66c078a6aa2`

While each repository listed above was reviewed, an emphasis was placed on the main body of code in the `zwallet` repository, the syncing and state-management algorithms in `zcash-sync`, the changes made to `librustzcash`, and the one-commit difference between `hhanh00/jubjub` and the upstream `jubjub` repo.

Primarily, this audit looked for the following kinds of issues:

1. Bugs that could result in loss-of-funds or theft-of-funds.

2. Privacy leaks that are not already documented in the Zcash Wallet App Threat Model [1].
3. The security and privacy of other wallet features, like memo-based messaging.
4. Usable security bugs, i.e. design issues that may confuse users about their level of security or privacy or APIs that are prone to misuse.

Some Ywallet features and related code were **not included in the scope**, specifically:

- The CI/CD practices used to build, sign, and deploy Ywallet.
- The security of the public infrastructure (`lightwalletd`) Ywallet connects to.
- `hhanh00/zwallet`'s GitHub Actions configuration.
- `hhanh00/k_chart`.
- `hhanh00/flutter_barcode_scanner`.
- The integration with hardware wallets.
- The GPU implementations of trial decryption.
- Dependencies, unless otherwise stated above.

1.2 Threat Model

A detailed threat model for Zcash shielded wallet apps is available online [1]. The security and privacy issues documented in that threat model also apply to Ywallet; we do not repeat them in this report. Any security and privacy issues not already documented there are considered bugs and are reported in this document.

2 Security & Privacy Findings

This section describes the security and privacy issues that were found.

The severity of each issue is graded to aid with prioritization. A rating of "*Critical*" means a critical vulnerability that can definitely be exploited to impact many users. "*High*" means a vulnerability that may have a severe impact for many users. "*Medium*" means a vulnerability of lesser impact or one that may only be exploitable in special circumstances. "*Low*" means a vulnerability whose exploitation would have very little impact on any user or which is unlikely to ever be exploited in practice.

2.1 Contacts stored in memos can be manipulated to intercept private messages

Severity: High

Ywallet has a feature which maps Zcash addresses to human-readable contact names. Initially, the address-to-name mapping is stored locally in the wallet, but it can be committed to the blockchain, encoded across several memos.

The contact list that gets stored in the wallet's memos is not cryptographically authenticated, so an adversary can replace its contents and change the user's address-to-name contact mapping without their knowledge or consent. All the attacker needs to know to carry out this attack is the vulnerable wallet's address, in order to send specially-crafted memos to modify the contact list.

An attacker can take advantage of this weakness to intercept messages. For example, if Alice and Bob are communicating using Ywallet's messenger, the attacker can update Alice's mapping for the name "Bob" to their own address, and update Bob's mapping for the name "Alice" to their own address. Then, as Alice and Bob are chatting with each other, their messages will actually go to the attacker, who has a chance to see and modify them before relaying them to their proper destination. To carry out this attack, the attacker needs to know both Alice's and Bob's addresses, and to know which human-readable names Alice and Bob use for each other in their contact lists.

To fix this, saved contact information should be cryptographically authenticated.

We recommend that for a long-term fix, the Zcash community should standardize an approach to memo signing (this will also address the next security issue). With an approach to memo signing, the contact storage memos could be required to be signed by the wallet itself, eliminating this vulnerability[1].

In the short term, Ywallet could derive an additional signing key off of the seed phrase or spending key. This needs to be done carefully to avoid conflicts with the rest of the Zcash protocol. We will work with the Ywallet developers to design a solution.

Update 2023-01-03: This issue was mitigated by having the wallet only update the contact list using memos in transactions that also spent funds from the same account. This way, the spending key holder must sign the memo. A payment made to a payment URI with the wallet's own address, containing a malicious memo, could still generate a contact-updating memo, but this would be visible to the user.

[1] Care needs to be taken to ensure memos set through payment request URIs (ZIP 321) can never modify the contact list.

2.2 Chat messages are unauthenticated, not reflected in the UI

Severity: Medium

Ywallet uses the "`Reply-To:`" convention to identify the sender of a memo. This information is not authenticated, i.e. anyone can add `Reply-To: <X>` for any address X, even one they do not own. Because it is not authenticated, users may be tricked into thinking a memo came from an address or contact that it really did not. This might enable phishing attacks, e.g. if one user uses the memo field to request actions from another user, those requests for action could be forged by an attacker (if they know both users' addresses). Note that this issue is not specific to Ywallet and is common among Zcash wallets.

One way to address this issue would be to add some symbol nearby the unauthenticated addresses and contact names, perhaps a broken lock, which lets the user click it for more information about the security status of the memo. By displaying the authentication status of messages in the app, users will better understand the memo field's security properties and can push for future support of authenticated memos. We recommend to the Zcash community that a ZIP standard should be introduced for authenticated memos.

Update 2023-01-03: This issue remains; users are expected to be aware that memos are not authenticated unless indicated otherwise. We recommend implementing optionally-signed memos as soon as a standard is available.

2.3 `zcash-sync` localhost API feature is unauthenticated

Severity: Medium

`zcash-sync` implements a feature that serves its API over HTTP via connections to localhost. Requests made to that interface are not authenticated, so unprivileged or different users of the same system can make requests to the API and access keys and send transactions. This allows theft-of-funds whenever the HTTP API is used on multi-user systems with untrusted users.

To fix this, the HTTP API should be marked as a development-only feature (e.g. requiring the use of a command-line flag like `--unsafe-http-api`), or it should be modified to require at least HTTP Basic Auth authentication using a randomly-generated secret.

Update 2023-01-03: This is to be mitigated with a disclaimer in the documentation.

2.4 Low maximum note value limits will cause an information leak

Severity: Low

Ywallet implements a feature that allows users to limit the maximum value of all notes that will be created when performing a spend. Since the transaction arity (number of inputs

and outputs in Sapling) is publicly visible, setting low values for this maximum will leak information about the amount of value being sent.

For example, if a user sends 10 ZEC, but sets the max-amount-per-note setting to 1 ZEC, a transaction will be created with 10 outputs, revealing the approximate amount that was sent. The sender and recipient of the shielded transaction are still private, but this information leakage may affect some users for certain use cases. In order to approximate the value this way, the attacker would need to have knowledge of, or guess, the user's maximum-note-value setting. If the attacker does not know the user's max-note-value setting, but the user always uses the same setting, then information is still leaked, since the number of outputs will be directly proportional to the value sent.

To fix this issue, the user should be informed of the potential for information leakage while using this feature. To reduce the amount of information leaked, Ywallet could pad transactions affected by this feature to a fixed number of outputs (e.g. 10 outputs). A disadvantage of the latter approach is that it would make these transactions (and the fact the user is using Ywallet) more easily recognizable on the blockchain. As long as users are appropriately informed of the information leak, it is acceptable to retain the current functionality.

Update 2023-01-03: We agreed that the risk here is low enough that it is safe to de-prioritize this issue relative to other work.

2.5 Seeds/spending keys are stored outside of secure storage

Severity: Low

Spending keys are stored in the wallet's SQLite database, rather than in the phone's secure storage. This exposes them to greater risk of theft in case a user loses their phone or has their phone stolen while the wallet app is not running. Using secure storage protects the keys, requiring the user to authenticate to their phone in order to allow the application to access them.

On iOS, secrets can be stored more securely in the keychain:

`https://developer.apple.com/documentation/security/keychain_services/keychain_items/using_the_keychain_to_manage_user_secrets`

On Android, there is no equivalent to Apple's keychain, but one can be approximated using `secure-storage-android`:

`https://github.com/adorsys/secure-storage-android`

Since the spending keys and seed phrases end up in the application process anyways, they can likely be stolen from the process's memory if the wallet is running when the phone is stolen, so fixing this issue is only of marginal benefit.

Update 2023-01-03: We agreed that the risk here is low enough that it is safe to de-prioritize this issue relative to other work.

2.6 lightwalletd connections allow use of an expired CA certificate

Severity: Low

In `zcash-sync` there is a `ca.pem` file containing an expired Let's Encrypt CA certificate. This certificate is loaded for use in authenticating connections to the lightwalletd server. It is unlikely that the key corresponding to this certificate was ever compromised, so there is little risk of users' connections being compromised.

To fix this, delete the `ca.pem` file and use the system's default certificate authorities list. Or, for added security, assuming all lightwalletds will use Let's Encrypt, load Let's Encrypt's up-to-date certificate and refuse certificates from all other certificate authorities.

Update 2023-01-03: The wallet will be tested without this `ca.pem` file and it will be deleted if it is unneeded.

2.7 Backup encryption API is error-prone, may lead to nonce re-use

Severity: Low

In `zcash-sync`, backup encryption is implemented which uses ChaCha20Poly1305 with a constant nonce:

```
const NONCE: &[u8; 12] = b"unique nonce";
...
pub fn encrypt_backup(accounts: &[AccountBackup], key: &str) -> anyhow::Result<String> {
...
        let cipher = ChaCha20Poly1305::new(key);
        // nonce is constant because we always use a different key!
        let cipher_text = cipher
            .encrypt(Nonce::from_slice(NONCE), &*accounts_bin)
            .map_err(|_e| anyhow::anyhow!("Failed to encrypt backup"))?;
        base64::encode(cipher_text)
```

The comment is correct that a fresh, random key is used for each encryption, so the current implementation is not at risk. However, if a key were to be reused, it may become possible to decrypt the ciphertexts without any knowledge of the key.

This API should be changed so that the `encrypt_backup` method itself generates a new encryption key and returns it along with the ciphertext. This will make it obvious that the key is never re-used, without having to review any code in the `zwallet` repository or the code of any other downstream users of the `zcash-sync` library.

Update 2023-01-03: The error-prone API is still in place, but there are plans to improve it in the future.

2.8 Some privacy is lost by fetching price data from CoinGecko

Severity: Low

Ywallet fetches coin price information from CoinGecko, a third-party cryptocurrency market information company. This means that CoinGecko could potentially track Ywallet users' approximate location (based on IP address) and the times they are actively using their wallet. Users are appropriately warned about this issue in the app's About page.

Fiat price conversions will be an important feature of all Zcash wallets, so we recommend that the Zcash community funds and builds a price API into lightwalletd itself. This can be done in a way that reveals little to no information to third-parties.

Update 2023-01-03: Since users are properly warned about the privacy leak, it is OK to keep the implementation as-is until price APIs are built into lightwalletd.

2.9 Dependency updates

Severity: Low

Several dependencies of the `zwallet` app are out of date. These can be discovered by running the `dart pub outdated` command.

We ran cursory searches to check for known vulnerabilities in some dependencies, but our search was not exhaustive. We recommend a general practice of keeping dependencies up to date with each release of the app. This is good practice under the assumption that any bugfixes made to dependencies may actually be fixing security bugs, whether that is known by the dependencies' authors or not.

Several of `zcash-sync`'s Rust dependencies have known security vulnerabilities (none of which appear to affect Ywallet at the present time). These can be discovered using `cargo audit` in that repository.

We also noted that the `hhanh00/k_chart` fork is a number of commits behind its upstream repository. Consider bringing it up to date with the latest commits on `OpenFlutter/k_chart`. The `zcash-sync` and `zcash-params` submodules in `zcash-sync` are also slightly out of date relative to the upstream repositories.

Update 2023-01-03: Several dependencies are still in need of an update, but there is no security risk that we are aware of.

3 Recommendations

3.1 Use darksidewalletd for integration testing, test reorg edge cases

One of the trickiest parts of writing a Zcash wallet is state management in the face of edge cases like reorgs. If old state is leftover after a reorg, for example, a user may think they have a spendable note when in fact they do not.

Test cases for such scenarios have been written for Electric Coin Co's wallets, using the "darksidewalletd" feature of lightwalletd. Darksidewalletd allows the test-writer to produce arbitrary blocks and trigger reorgs arbitrarily in order to test scenarios that are possible but only occur rarely on mainnet and testnet.

We recommend implementing the same style of tests in Ywallet as a defense against state-management bugs. This should be a high priority for future funding and development.

3.2 Publish security contact information

Community members and security auditors should have a way to report bugs to Ywallet's author. We recommend posting security contact information on the Ywallet homepage and in the READMEs of the `zwallet` and `zcash-sync` repos.

3.3 Sign releases

Ywallet implements the code signing that is required for publishing the app in the Google Play app store and the iOS App Store. These signatures are not made easily available to the public; we recommend Ywallet also sign its GitHub releases using GPG.

3.4 Avoid the use of `ON CONFLICT DO NOTHING` in database queries

Several of Ywallet's `INSERT` queries in `zcash-sync` make use of `ON CONFLICT DO NOTHING`, which will cause the record-to-be-added to be dropped if there is already conflicting data. This has the potential to hide bugs and may lead to state-management problems in the future. We recommend dropping these pragmas and instead investigating occurrences of insertion conflicts as bugs.

3.5 Standardize the `MSG` memo encoding for messages in a ZIP

Ywallet's messaging feature encodes messages into a memo in a certain format. For compatibility with other wallets that wish to implement messaging, it would be a good idea to specify this encoding in a ZIP.

3.6 Make `from_c_str` return an error if information would be lost

In `zcash-sync` FFI, the `from_c_str` function is used to convert from a C-style null-terminated string into a Rust `String`. `to_string_lossy` is used to do this; we recommend instead generating and handling an error in case information would be lost. (We could find no cases where important information would actually be lost.)

3.7 Integrate `cargo audit` into CI checks for `zcash-sync`

The `cargo audit` tool can be used to automatically search for known vulnerabilities in Rust dependencies. We recommend integrating `cargo audit` checks into the CI for `zcash-sync`.

It may also be desirable to run `dart pub outdated` in `zwallet`'s CI to notice updates to Dart dependencies.

4 Good Things

4.1 Code organization and clarity

Ywallet's `zwallet` and `zcash-sync` codebase is well-organized and it is easy to find the implementations of all features without much help. Generally the code is written in a clear style, and being mainly Rust, is written so that all edge-cases and errors are handled.

4.2 Secure UX improvements

Ywallet's scanning algorithm increases its usability (especially in the face of high network load) without sacrificing privacy beyond information leaks that are already documented in the threat model for Zcash wallet apps. The wallet also experiments with novel privacy-enhancing features like balance auto-hide and does a good job of communicating its privacy limitations to users.

It is a good thing for the Zcash community to have an independent wallet implementation which can be used to explore different user experiences and implementation directions. A recommendation we have would be for Ywallet to explore privacy enhancements to the light wallet protocol, to close off some of the known privacy leaks that are currently present in all wallets.

4.3 Randomized testing of cryptography algorithms

Ywallet has its own implementation of Sapling's Pedersen hashes and note commitment Merkle tree. This is necessary for its implementation of warp sync. Ywallet has implemented randomized tests of these algorithms, comparing them against the implementations

in `librustzcash`. This greatly reduces the chances of a bug in Ywallet's implementations.

4.4 Fast response time with quick bug fixes

Ywallet's developers responded to this audit report and had already fixed several of the issues within days of receiving the report. This is impressive prioritization and speed!

5 Future Work

The following areas should be considered in future audits of Ywallet.

5.1 Security analysis of GPU code

A future audit should analyze the CUDA and Metal implementations of trial decryption, as this was out of scope for this audit. According to Ywallet's authors, this code has not been published and is not present in any production release, so at this time there is no risk to users from bugs in the GPU code.

5.2 Review to-be-written integration tests for completeness

In a recommendation above, we suggested writing integration tests that use darksidewalletd to trigger edge-cases like reorgs. A future review should review these tests for completeness, to ensure that all edge cases have tests.

5.3 Analysis against a malicious light wallet server

Currently, Zcash wallets operate in a model where the light wallet server is assumed to be honest. In the future, wallets should remove this assumption by validating block headers and hashes of the information they receive (such as notes and nullifiers). If and when Zcash wallets move to a model where the light wallet server is assumed to be malicious, more security analysis should be done on Ywallet to make sure it is secure in the face of a malicious server, for example making sure that the adversary cannot overcome the integrity checks, brick the wallet, or otherwise attack the user.

6 Conclusion

In conclusion, Ywallet's code was found to be clearly organized and written. One high-severity issue was found that allowed messages sent through the app's messaging feature to be intercepted. Several medium- and low-severity issues were also found and are documented above. No critical-severity issues were found. We recommend emphasizing integration testing

in Ywallet's future development roadmap, making use of darksidewalletd to test the wallet's state management in the face of reorgs and other edge cases. We also recommend that a standard for memo signing be developed within the Zcash community.

7 Acknowledgements

This security analysis was performed as part of the Zcash Ecosystem Security Grant [4] funded by Zcash Community Grants [3]. Thanks to the Zcash Grants Committee and the broader Zcash community for supporting the security of privacy-enhancing open-source software.

References

[1] Zcash wallet app threat model.
`https://zcash.readthedocs.io/en/latest/rtd_pages/wallet_threat_model.html`.

[2] Ywallet website.
`https://ywallet.app/`.

[3] Zcash community grants.
`https://zcashcommunitygrants.org`.

[4] Zcash ecosystem security grant.
`https://forum.zcashcommunity.com/t/zcash-ecosystem-security-lead/42090`
`https://zecsec.com`.

[5] Zwallet github repo.
`https://github.com/hhanh00/zwallet`.