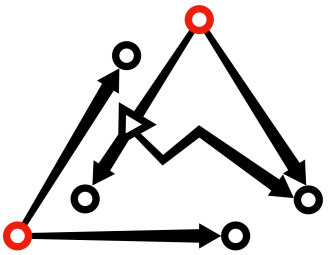

Least Authority

PRIVACY MATTERS

Blank Wallet Browser Extension

Security Audit Report

Blank

Final Audit Report: 22 September 2021

Table of Contents

Overview

Background

Blank requested that Least Authority perform a security audit of the Blank Wallet browser extension for the Chrome browser. The Blank Wallet aims to provide a privacy-preserving, non-custodial wallet for Ethereum.

Project Dates

- **July 5 - August 6**: Code review *(Completed)*
- **August 11**: Delivery of Initial Audit Report *(Completed)*
- **September 20 - 21**: Verification Review *(Completed)*
- **September 22**: Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- May-Lee Sia, Security Researcher and Engineer
- Rai Yang, Security Researcher and Engineer
- Suyash Bagad, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Blank Wallet browser extension followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Extension-background: https://github.com/Blank-Wallet/extension-background
- Extension: https://github.com/Blank-Wallet/extension
- Extension-provider: https://github.com/Blank-Wallet/extension-provider
- Extension-ui: https://github.com/Blank-Wallet/extension-ui
- Blank-make: https://github.com/Blank-Wallet/blank-make

Specifically, we examined the Git revisions for our initial review:

Extension-background: `55715318df52ce1a6003aa7fc6d7cc30147d2ad8`

Extension: `124891606ad3337f9d9c1791ceac844f8719d336`

Extension-provider: `7c7d737e44f30cc8b129c8dd08941381399f018d`

Extension-ui: `54ed29faa6be9748bf7dbe9b88838639e0acdac0`

Blank-make: `ec77c5b81ab42fac28642db8429a0fedb7a3852c`

For the verification, we examined the Git revision:

Extension-background: `8af3e41d1b778693cfc388fa97d74047526c8bf0`

Extension: `0919e3b1941657e4f8a6f7efb0cd6e6b9889da0e`

Extension-provider: `8d76daf79ee07e371565358445dc78a45c508c9e`

Extension-ui: `b384d43569ca75b3fa0a51328489e0fafca058bc`

Blank-make: `ec77c5b81ab42fac28642db8429a0fedb7a3852c`

For the review, these repositories were cloned for use during the audit and for reference in this report:

Extension-background:
https://github.com/LeastAuthority/Blank-Wallet-extension-background/tree/la/audit

Extension: https://github.com/LeastAuthority/Blank-Wallet-extension

Extension-provider: https://github.com/LeastAuthority/Blank-Wallet-extension-provider

Extension-ui: https://github.com/LeastAuthority/Blank-Wallet-extension-ui

Blank-make: https://github.com/LeastAuthority/Blank-Wallet-blank-make

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Blank Wallet Extension Documentation:
  https://github.com/Blank-Wallet/blank-extension-documentation
- Blank Medium: https://blankwallet.medium.com/
- Blank Whitepaper:
  https://blank-wallet.github.io/extension-whitepaper/book/overview/overview.html

In addition, this audit report references the following documents:

- F. B´eres, I. Seres, A. Bencz´ur, M. Quintyne-Collins, 2020, "Blockchain is Watching You: Profiling and Deanonymizing Ethereum Use." *arXiv:2005.14051* [BSB+20]

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and adherence to best practices;
- Exposure of any critical information during user interactions with the blockchain and external libraries, including authentication mechanisms;
- Adversarial actions and other attacks that impact funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Vulnerabilities in the code, as well as secure interaction between the related and network components;
- Proper management of encryption and storage of private keys, including the key derivation process;
- Inappropriate permissions and excess authority;

- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Blank Wallet browser extension allows users to generate a key pair for holding and exchanging ETH and ERC-20 tokens. The distinguishing feature of the Blank Wallet browser extension is that it is designed to allow users to send and receive tokens anonymously. This privacy-enhancing feature utilizes Tornado Cash, a non-custodial mixer protocol for private transactions on Ethereum, to obfuscate the link between the user's initial ETH deposit address and the subsequent new withdrawal address.

System Design

Our team performed a broad and comprehensive review of the system design and examined all security critical components for vulnerabilities and implementation errors. The Blank Wallet browser extension provides anonymity of transactions by creating a Tornado Cash deposit note, which contains a secret key and a nullifier. In order to withdraw the Tornado deposit note, the browser extension wallet generates a proof of the validity of the deposit note (i.e. that the secret key is correct and the nullifier is unspent). The validity proof is generated using the circuit and prove key, which are provided by Tornado Cash. The generated proof is then verified by the Tornado Cash smart contract. If it is validated, the smart contract releases the deposited funds to a newly generated withdrawal address, breaking the link between the depositing address and the withdrawal address.

Note Import & Derivation Implementation Issues

The import of deposit notes from seed phrase implemented in the Blank Wallet browser extension contains issues that may disable the intended functionality of the browser extension wallet or result in risk to users. In the event that the user changes the network used by the browser extension wallet in order to interact with Tornado, user deposit notes would fail to import, preventing the user from withdrawing deposits made on the previous network. We recommend setting the import flag `isImported` for all supported networks to enable the import of deposit notes (Issue C). Similarly, upon importing an existing wallet using the seed phrase, the retrieval of the network always returns an incorrect default, causing the existing deposit notes import to fail for any network, preventing the user from making withdrawals (Issue E).

Furthermore, a deposit note's derivation path is derived from a `chainID` and currency amount pair. If two blockchains have the same `chainID`, the deposit notes will be identical, facilitating the possibility for a double-spend attack using one deposit note on two different chains, which will result in a loss of user funds. To mitigate against this possibility, we recommend generating a unique derivation path to prevent double-spending attacks (Issue F).

Use of Cryptography

The Blank Wallet browser extension uses a SHA512 hash of the `derivedKey` to compute the secret and nullifier used in generating proofs of valid deposit notes. We suggest using an alternative hash function that could increase the efficiency of the proof generation, resulting in an improved user experience (Suggestion 6).

Additionally, the Blank Wallet browser extension implementation uses `getCryptoRandom` as a pseudo-random number generator to select an unspent deposit note for a withdrawal. Although this

algorithm generates sufficient randomness up to 32 bits, we caution against using this algorithm to create random numbers greater than 32 bits, as this could lead to collisions (Suggestion 5).

Browser Extension Security

In the `GenericVault` implementation, the password and seed phrase are kept in cleartext in the memory when the browser extension wallet is unlocked, which can be captured in the memory dump by the browser's DevTools and stolen by an attacker that gains control of the browser extension wallet. This can result in the browser extension wallet being compromised and a total loss of user funds. As a result, we recommend removing the password from memory after the browser extension wallet is unlocked and storing the root key together with the seed phrase in the vault (Issue A).

In addition, the browser extension wallet currently allows users to set insecure passwords, which reduces its security. Weak passwords facilitate an easier compromise of the browser extension wallet by attackers and may result in the total loss of user funds. We recommend that acceptable passwords be constrained according to NIST Guidelines (Issue D).

Finally, the validation of the deposit note proof that is required for withdrawal is generated using Tornado Cash's fork of `snarkjs`, which introduces the use of `unsafe-eval` to the browser extension wallet's Content Security Policy (CSP). Given that `unsafe-eval` can run untrusted code, it is considered a known security vulnerability. During the security audit, we suggested that the Blank team explore using the SRI hash of the circuit function on CSP, which did not mitigate the vulnerability. Subsequently, the Blank team came up with a solution, which forked the Tornado version of `snark.js` that was using the `unsafe-eval` and created a hardcoded version to specifically use only the `tornado.json` circuit. This change moved all the functions on the file that were evaled to a `.js` file, calling them directly instead.

De-anonymization Risks of Dependencies on Tornado Cash

The privacy features that are proclaimed by the Blank Wallet browser extension are critically dependent on Tornado Cash. The browser extension wallet makes assumptions about Tornado Cash that must not be broken for the privacy-preserving features to work. In addition, although the Tornado Cash system has undergone security audits, Tornado Cash transactions can be de-anonymized in various ways [BSB+20]. Users should be aware that a short time interval between deposit and withdrawal, reuse of a withdrawal address, a centralized relayer node, or an exposed IP of the browser extension wallet could de-anonymize a browser extension wallet using the Tornado Cash system.

We suggest that the Blank team make users aware of potential vulnerabilities of Tornado Cash in the user documentation of the browser extension wallet and by providing resources such as how to stay anonymous (Suggestion 12). Additionally, we suggest that a feature be implemented to inform the users that a withdrawal attempt soon after a deposit could result in the deanonymization of the withdrawal transaction (Suggestion 10). Finally, we recommend that the Blank team conduct ongoing research on improving the privacy of the Blank Wallet browser extension with respect to Tornado Cash.

Code Quality

The Blank Wallet browser extension code base is very well organized and adheres to best practices. However, there are some instances where a high level of complexity is present in the coded implementation (i.e. the transaction flow in `TransactionController.ts`). Excessive complexity increases the risk of security vulnerabilities and implementation errors going unnoticed by maintainers and security reviewers of the implementation. As a result, we recommend that complexity in the implementation be reduced where possible (Suggestion 14).

In addition, an implementation error is present in `PermissionsController.ts`, which could cause the system to behave unexpectedly. Unexpected behavior can impact both the user experience or disrupt the

functionality of specific features, with unknown consequences. As a result, we recommend that the error be fixed and that the code base be scanned for similar errors (Issue B).

Finally, the implementation would better adhere to security best practices if several instances of redundant code are removed to reduce complexity and increase the readability of the code (Suggestion 1; Suggestion 7). Furthermore, implementing a correctly configured linter would improve the overall code quality (Suggestion 9).

Tests

The Blank Wallet browser extension has sufficient test coverage for success and failure cases, helping to protect against errors and bugs and identifying potential edge cases, which may lead to vulnerabilities or exploits. We commend the Blank team for their diligence in implementing a robust test suite, which improves the overall security of the implementation.

Documentation

The documentation provided by the Blank team in the repository `README` files was accurate and helpful in describing the intended functionality of the system. However, the documentation could be further improved by adding more detailed descriptions for security critical components, including `extension-background` and `extension-provider`. Similarly, a detailed and comprehensive description of the Blank Wallet browser extension transaction flow would facilitate an easier understanding of the expected behavior of the system, which would better allow maintainers and security reviewers to identify potential instances of unintended behavior and errors that may lead to security vulnerabilities (Suggestion 12).

Code Comments

The documentation contained within the Blank Wallet browser extension code base in the form of code comments sufficiently explains most of the intended functionality of the system components. However, some functions are not described in the code comments and we suggest that they also be commented (Suggestion 13). In addition, there are a considerable number of outstanding TODO items in the code comments. We suggest that the TODO items be resolved to prevent confusion about the completion of the implementation (Suggestion 8). Security audits of code and documentation that are actively being developed may result in missed security issues and an inefficient security review. We suggest that all future independent security audits be conducted during milestones in the development roadmap in which all development and documentation for the specific in-scope audit target is complete.

Scope

While the Blank Wallet browser extension implementation is significant in size, the scope of the security audit was well-defined. The Blank team identified specific areas of concern, which helped guide our team in identifying, understanding, and thoroughly reviewing the security critical components of the system. As a result, we found the scope of the audit to be sufficient in that it covered all security critical aspects of the system design and implementation.

Dependencies

Outside of the dependency concerns previously noted as it relates to Tornado Cash, we did not identify any dependency issues.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Password and Seed Phrase in Cleartext in Memory When Browser Extension Wallet Unlocked	Partially Resolved
Issue B: Check for Pending Permission Error May Cause Unintended Flow	Resolved
Issue C: Import Deposit Note Does Not Work on Network Change	Resolved
Issue D: Low Entropy Passwords Allowed	Resolved
Issue E: Network Name Error Causes Wallet Import Failure	Resolved
Issue F: Double-Spend of Deposit Note With Same ChainId on Different Blockchains	Resolved
Issue G: Browser Extension Wallet Allows for Message Signing Without Explicit User Confirmation	Resolved
Suggestion 1: Remove Redundant Network Support Check	Resolved
Suggestion 2: Research Batch Updates to Merkle Trees	Resolved
Suggestion 3: Check Spent Note Status Before Merkle Tree Update	Resolved
Suggestion 4: Check if the rootPath is Set at the Start of getNextUnderivedDeposit	Resolved
Suggestion 5: Caution on Using getCryptoRandom	Resolved
Suggestion 6: Research Using Blake2b/3b instead of SHA-512	Resolved
Suggestion 7: Remove Redundant Vault Unlock in ImportNotes	Resolved
Suggestion 8: Resolve TODOs in the Code Base	Resolved
Suggestion 9: Improve Linting Rules to Improve Code Quality	Resolved
Suggestion 10: Alert Users on Withdrawal Attempt Soon After Deposit	Resolved
Suggestion 11: Use estimateGas to Estimate Deposit and Withdrawal Gas Costs	Resolved
Suggestion 12: Improve Documentation	Unresolved

Suggestion 13: Increase Code Comments	Unresolved
Suggestion 14: Reduce Implementation Complexity	Unresolved

Issue A: Password and Seed Phrase in Cleartext in Memory When Browser Extension Wallet Unlocked

Location

blank-deposit/infrastructure/GenericVault.ts#L74

src/controllers/AppStateController.ts#L93

Synopsis

When the Blank Wallet browser extension is unlocked, the password and seed phrase are kept in cleartext in the memory, which can be captured in the memory dump by the browser's DevTools. The password can then be captured by an attacker that gains control of the browser extension wallet.

Impact

An attacker captures the password and seed phrase, resulting in gaining control of the browser extension wallet, which may result in a total loss of user funds.

Preconditions

The attacker gains physical control of the browser extension wallet when unlocked and dumps the memory from the browser.

Technical Details

The password is required to unlock the browser extension wallet. Once it has been unlocked, the password is stored in the memory and the seed phrase is decrypted using the password from the vault. The password and seed phrase are then saved in the memory to use later to derive the root key. The password and seed phrase are both stored in cleartext. Once the memory is dumped by the browser's DevTools, they are exposed. The memory dump enncompasses the following steps: open *Chrome Developer Tools,* select *Memory*, click *Take Snapshot*, and then *Save*. With the downloaded file, run the strings command and, for quick results, grep for a word in the password and seed phrase to view in clear text.

Mitigation

We recommend removing the password from the memory after the browser extension wallet is unlocked. The password is not required after unlocking the browser extension wallet to make transactions. When the password is required again for other operations (e.g. import notes, network change, and others), the password can be re-entered by the user.

In addition, we recommend storing the root key together with the seed phrase in the vault so that when the browser extension wallet key (derived from the root key) is required to sign a transaction, it is read from the decrypted vault and the seed phrase is not required to derive it, which exposes the cleartext seed phrase to the memory. A root key is more difficult to identify in a memory dump than a seed phrase.

Status

The Blank team has decided against requiring that the user re-enter their password when vault unlocks are needed, due to the significant user experience implications. To slightly decrease the impact of the

cleartext password, they have decided to hash the password (with browser extension wallet ID as salt) to secure other programs against dictionary attacks, for which the user might use the same password. Hashed passwords are more difficult to identify than cleartext passwords once memory is leaked, however, they are still vulnerable to brute-force attack on weak passwords (Issue C). For root key derivation, the Blank team has decided against making changes due to other security concerns and are instead adhering to MetaMask's eth-keyring controller approach. We acknowledge that the current resolution has slightly decreased the vulnerability of cleartext passwords in memory and has the minimum impact on the user experience. However, the changes implemented by the Blank team only partially resolve the issue, since the hashed password is still in the memory and the aforementioned brute-force attack of a weak password is still feasible.

Verification

Partially Resolved.

Issue B: Check for Pending Permission Error May Cause Unintended Flow

Location

src/controllers/PermissionsController.ts#L295-L302

Synopsis

The code segment below checks for pending permission requests coming from a specific origin, in which case it throws an error and stops the process.

```
if (
  !Object.values(requests).every((request) => {
    request.origin !== origin;
  })
```

Since the statement `request.origin !== origin;` is surrounded by brackets with no return statement, its value is ignored which renders the whole check useless.

Impact

It is difficult to determine the extent of the impact of bypassing this check, however, it could affect user experience or disrupt the functionality of specific features in the browser extension wallet, resulting in unforeseen security critical consequences.

Remediation

We recommend implementing one of the following remediations:

1. Remove the brackets surrounding the conditional statement; or
2. Add a return statement to fix the issue.

Status

The Blank team has implemented the suggested remediation by replacing the `every` method from the Array prototype with a `for...in` loop.

Verification

Resolved.

Issue C: Import Deposit Note Does Not Work on Network Change

Location

blank-deposit/tornado/TornadoService.ts#L210

Synopsis

When changing the browser extension wallet connection to a supported network such as Goerli, the imported deposit note must be reconstructed for the target network. In the current implementation, the import flag `isImported` is not set for other networks other than the default (mainnet). As a result, `importNotes` does not work.

Impact

The deposit note imported will not be reconstructed for the target network when changing the network, preventing the release of previously deposited user funds from Tornado Cash.

Preconditions

The network is changed from the default network to another supported network when the browser extension wallet is imported using the seed phrase.

Technical Details

```
  const { isImported, isInitialized } =vault.deposits[name as
AvailableNetworks];

     if (isImported && !isInitialized) {

        this.importNotes();

     }
```

`isImported` is not set in the deposit vault state for the non-default network (mainnet) and `importNotes` will not run when the network is set to an alternative target network.

Mitigation

We recommend adding the `isImported` flag to the deposit vault state for all networks.

Status

The Blank team has added the `isImported` flag to the deposit vault state.

Verification

Resolved.

Issue D: Low Entropy Passwords Allowed

Location

routes/setup/PasswordSetupPage.tsx#L16

Synopsis

When setting up the Blank Wallet browser extension, the password validation in the user interface (UI) allows for a password length minimum of only eight characters. Additionally, password validation allows

for a number of weak forms of passwords, such as dictionary words, repetitive letters, and sequential characters.

Although browser extension wallet users are provided a password strength estimate in the UI, this feature is only informational to encourage users to create secure passwords. However, this bar does not enforce any particular level of password strength and users have the ability to choose insufficiently secure passwords.

Preconditions

A machine that the browser extension wallet is installed on is compromised.

Impact

An attacker gains full access to a user's account by brute-forcing an insecure password, which may result in total loss of user funds.

Feasibility

Easy. Password attacks are increasingly common. Furthermore, it may be that Blank's anonymous withdrawals feature is an additional incentive to conduct such an attack.

Technical Details

Password entropy is given by the following formula:

$$E = L * log2(R)$$

where:

R - Size of the pool of unique characters from which we build the password (in this case, a-z, A-Z, and 0-9)

L - Password length (i.e. the number of characters in the password; in this case, 8)

An 8-character mixed case password that includes a numeral contains approximately 48 bits of entropy and would take under a day to compromise, at a very conservative rate of 10 billion guesses/second (1 trillion password guesses per second speeds or more are not uncommon, depending on cores and machines used). A dictionary word, prior breached password, or sequential/repeating password would crack nearly instantaneously.

According to `zxcvbn` documentation, a password should have a minimum of approximately 1e10 bits of entropy to be considered "safely unguessable" from an offline attack with user-unique salting but a fast hash function like SHA-1, SHA-256, or MD5.

Mitigation

We recommend increasing the requirement of password entropy and disallowing passwords that are easy to guess.

We recommend following the NIST Guidelines, which are industry standard rules and best practices for handling passwords when building memorized secret authenticators.

Remediation

We recommend using the `dropbox/zxcvbn` library for a password composition policy intended to prevent the use of passwords obtained from previous breaches, common passwords, and passwords containing repetitive or sequential characters, as recommended by the NIST guidelines. We suggest requiring that all passwords meet `zxcvbn`'s strength rating of 4.

Status

The Blank team is using a library which implements the `zxcvbn` package and has added a validation step to the UI that requires the user to choose a password strength of at least 3, which is categorized as "safely unguessable".

Verification

Resolved.

Issue E: Network Name Error Causes Wallet Import Failure

Location

`blank-deposit/tornado/TornadoNotesService.ts#L474`

Synopsis

In `getNextUnderivedDeposit`, the network name is always incorrectly set to the default network (Homestead) by retrieving the network provider.

Impact

The wrong network name prevents the imported deposit note from being reconstructed, causing the import of deposit notes to fail. This prevents the user from retrieving unspent deposit notes and to perform withdrawals.

Technical Details

In `getNextUnderivedDeposit`:

```
const { name: network, chainId } =
this._networkController.getProvider().network;
```

`network` will always be the default mainnet name (Homestead) instead of the correct network name

```
const depEv = await this._tornadoEventsDb.getDepositEventByCommitment(

network as AvailableNetworks,

currencyAmountPair,

deposit.commitmentHex,

);

} catch (error) {

console.error('Unable to check if deposit has been spent');

spent = undefined;

}
```

The deposit event search will fail with the wrong network name, which will cause the import of the deposit note to fail.

Mitigation

We recommend deriving the network name by

```
const { chainId } = this._networkController.getProvider().network;

network = getNetworkFromChainId(chainId)
```

Status

The Blank team has implemented the suggested mitigation, which has resolved the network name error resulting in wallet import failure.

Verification

Resolved.

Issue F: Double-Spend of Deposit Note With Same ChainId on Different Blockchains

Location

blank-deposit/tornado/config/paths.ts#L66

utils/constants/networks.ts

Synopsis

The deposit note's derivation path is derived from a `chainId` and currency amount pair. The note could be identical if there are two blockchains which have the same `chainId`, resulting in the possibility for double-spending attacks on different chains.

Impact

A deposit note could be double spent on different blockchains resulting in a loss of user funds.

Preconditions

Different blockchains with the same `chainId` would be required for this to occur.

Technical Details

In `getDerivationPath`, the deposit note's derivation path is generated by

```
`${BASE_PATH}/${chainId}'/${derivation}`

const derivation = BLANK_DEPOSITS_DERIVATION_PATHS[currency][amount];
```

Supported `network(chainId)` is added explicitly in a configure file. In the case of networks on different blockchains that have the same `chainId`, the derivation path would be the same , as a result the note would be the same in case of `depositIndex` is also the same across different chains.

Mitigation

We recommend adding a blockchain ID (e.g. hash of blockchain name) to the derivation path of the deposit note, preventing `chainId` conflicts between different blockchains.

Status

The Blank team has indicated that no action has been taken and responded with the following:

"Given that currently, adding a new network requires manual changes and that the blockchains that will likely add in the feature are EVM based, their `chainId` will be different responding to what is specified in EIP-155. We can later on, if needed, change the derivation to use an arbitrary integer and use the same as the `chainId` for the already supported networks."

We acknowledge that manual network addition and EVM based chains can currently guarantee a unique `chainId`. Since the Blank team has indicated that they will implement a new derivation method in the future as needed, we consider this issue to be resolved.

Verification

Resolved.

Issue G: Browser Extension Wallet Allows for Message Signing Without Explicit User Confirmation

Location

`extension-background/src/controllers/BlankProviderController.ts#L233`

`extension-background/src/controllers/BlankProviderController.ts#L628`

Synopsis

Two related vulnerabilities have been identified where explicit user confirmation is not needed to sign a message:

Part 1: The Blank team identified an issue in `BlankProviderController`, where an external site can call the JSON-RPC method `eth_sign` through the `provider`, which would immediately trigger the browser extension wallet signing a message, without prompting the user for confirmation.

Part 2: The Blank team brought this issue to our attention, after which we identified the following additional vulnerability:

In the window management method called `stateWatch` in the `BlankProviderController`, `SIGNING` is declared as a watch-state but the state update does nothing while "signing" a message. This implies that the user is not prompted before signing a message.

Impact

These vulnerabilities are critical as they imply that an external provider can sign any message on behalf of a user without the user explicitly confirming.

Technical Details

Both of these are similar, as they relate to signing an EIP-712 message without user confirmation.

Part 1: Call the JSON-RPC method `eth_sign` through the `provider`, which would immediately trigger the browser extension wallet signing a message, without prompting the user for confirmation.

Part 2: `SIGNING` is a watch-state but the state update does nothing while "signing" a message.

Mitigation

For Part 1, the Blank team has prevented the ability of `BlankProvider` to sign messages by removing the `eth_sign` method from the `BlankProvider` as a fix. Alternatively, they can ensure that the user is asked for approval before signing by a provider.

Remediation

The Blank Team has planned on properly implementing message signing, with user confirmations, in order to address Part 1.

For Part 2, we recommend changing the `watchState` called `SIGNING` to check if the user has approved signing of a message and then in `checkWindows` it can be confirmed that the signing was correctly approved.

Status

The Blank team has removed the `eth_sign` method since it may be used to sign a transaction. Additionally, they have added support for other signature methods (`eth_signTypedData`, eth_signTypedData_v1, `eth_signTypedData_v3`, `eth_signTypedData_v4` as per EIP-712, and eth.personal.sign) with proper user confirmation implemented.

Verification

Resolved.

Suggestions

Suggestion 1: Remove Redundant Network Support Check

Location

`blank-deposit/tornado/TornadoService.ts#L1153`

`blank-deposit/tornado/TornadoService.ts#L190`

Synopsis

In the `importNote` function, the network support check for a deposit note is redundant, since the check is performed upon network change subscription.

Mitigation

We recommend removing the network support check in `importNote`.

Status

The Blank team has responded that after further review by their team, this method is being triggered in other places apart from the network change callback, so removal of the check is not required. We verified this in the code and agree with the Blank team's decision to keep the network support check.

Verification

Resolved.

Suggestion 2: Research Batch Updates to Merkle Trees

Synopsis

The Blank Wallet browser extension's backend keeps track of the deposits using a Merkle tree to store deposit notes. The deposit notes are added to the Merkle tree sequentially (i.e. every new note is added to the next empty leaf in the Merkle tree). The sequential addition of data into the tree allows for batch updates (i.e. adding multiple leaves at once). The current implementation of the Blank Wallet browser extension inserts new notes individually. This could be improved to allow batch updates of the deposit notes' tree. We explain in Appendix A how batch updates to a Merkle tree could work. Additionally, we refer to the documentation of how subtree updates work in Zkopru.

Mitigation

We recommend conducting research on supporting batch updates to the deposit note tree. Batch updates require a shorter Merkle path for verification and could reduce the gas costs associated with the verification of withdrawals.

It is important to note that batch updates to the Merkle tree could work only if the batch size is a power of two. If and when the Blank Wallet browser extension supports simultaneous multiple deposits for users, it would be worthwhile to use batch updates to the Merkle tree.

Status

The Blank team has conducted the necessary research on using batch updates to Merkle trees. Further updates will be considered in future implementations in order to achieve better performance.

Verification

Resolved.

Suggestion 3: Check Spent Note Status Before Merkle Tree Update

Location

`blank-deposit/tornado/TornadoNotesService.ts`

Synopsis

In the `generateMerkleProof` function, a Merkle hash path (i.e Merkle proof) is generated for a given deposit. The flow of the function (in the given order) is as follows:

1. Update the Merkle tree to ensure that the latest deposits have been added to the tree;
2. Check if the deposit is spent; and
3. Generate a Merkle proof using the root of the tree and the leaf index.

The order of execution should be changed to (2, 1, 3) instead of (1, 2, 3). This would ensure that when the function is invoked for spent deposits, it could exit right away after checking the spent status instead of first updating the tree. Although this function is unlikely to be invoked with spent deposits, it would ensure a better execution of the logic.

Mitigation

We recommend changing the `generateMerkleProof` function to check if a deposit is spent right at the beginning of the function, as detailed below:

1. Check if the deposit is spent;
2. Update the Merkle tree to ensure that the latest deposits have been added to the tree

3. Generate a Merkle proof using the root of the tree and the leaf index

Status

The Blank team has implemented the recommended mitigation to check if a deposit is spent right at the beginning of the function.

Verification

Resolved.

Suggestion 4: Check if the rootPath is Set at the Start of getNextUnderivedDeposit

Location

`blank-deposit/tornado/TornadoNotesService.ts`

Synopsis

The `getNextUnderivedDeposit` function yields the next underived deposit for a given currency-amount pair. At the start of this function, the network and the contract corresponding to the currency-amount pair is fetched. This is followed by checking if the root path is set (i.e. the browser extension wallet is initialized). This check should be present right at the start of this function so that if the browser extension wallet is locked or is not initialized, the function could exit immediately with an error.

Mitigation

We recommend changing the `getNextUnderivedDeposit` function to check if the root path is set right at the beginning of the function.

Status

The Blank team has implemented the `getNextUnderivedDeposit` function to check if the root path is set right at the beginning of the function.

Verification

Resolved.

Suggestion 5: Caution on Using getCryptoRandom

Location

`blank-deposit/utils/getCryptoRandom.ts`

Synopsis

The function `getCryptoRandom` is used to randomly pick the index of an unspent deposit on a request for withdrawal. It generates a pseudo random 32-bit (or less) number and ensures that an adversary cannot link the withdrawal to unspent deposits from the past. This works since the deposit tree depth is less than 32. However, if this function is used to generate numbers greater than 32 bits, the algorithm is not likely to preserve pseudo randomness. The way this function generates a random number given a `maxValue`:

```
⌊maxValue * rand32() / 2^{32}⌋
```

where `rand32()` generates a random 32-bit number. Ideally, the probability of a number generated in the range `[0, maxValue]` should be `(1 / maxValue)`. Using this algorithm, this probability is constant at `(1 / 2^{32})`, which can lead to collisions when `maxValue > 2^{32}`.

Mitigation

We recommend avoiding the use of this function to generate numbers greater than 32 bits. Currently, this function is only used in one place to generate random numbers less than 32 bits and we caution against its use elsewhere in the implementation.

Additionally, we suggest changing the name of this function to `getCryptoRandom32()` in order to avoid its unintentional use for larger numbers.

Status

The Blank team has changed the function name to `getCryptoRandom32()` and it is used in a single place to generate a random 32-bit number.

Verification

Resolved.

Suggestion 6: Research Using Blake2b/3b Instead of SHA-512

Location

https://www.blake2.net/

https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf

Synopsis

To compute the `secret` and `nullifier`, the derive key is hashed:

```
hashedKey := SHA512(derivedKey)

secret := hashedKey[0:31], nullifier := hashedKey[33:64].
```

SHA-512 is used so that the `secret` and `nullifier` (each 31 bytes long) could both be fetched from the 64-byte hash output. Instead, a faster hash function such as Blake2b or Blake3b could be used to speed up the computation. Blake2b is about 2 times faster than SHA-512 while Blake3b is 10 times faster. Since every deposit needs the `secret` and `nullifier` to be computed on the client-side, a faster hash function could improve the user experience.

Mitigation

We recommend investigating the use of a faster hash function like Blake3b instead of SHA-512.

Status

The Blank team has replaced the SHA512 hash with a Blake3b hash for faster computation of the nullifier and the secret, and are using an open-source npm package of blake3 authored by the Blake3 creators.

Verification

Resolved.

Suggestion 7: Remove Redundant Vault Unlock in ImportNotes

Location

src/controllers/BlankController.ts#L1788

blank-deposit/tornado/TornadoService.ts#L201

blank-deposit/tornado/TornadoService.ts#L1151

Synopsis

In deriving the browser extension wallet from the key phrase or changing the browser extension wallet network, the vault is unlocked. It is unlocked again in `importNotes`.

Mitigation

We recommend removing the vault unlock in `importNotes`.

Status

The Blank team refactored the vault implementation and created two new functionalities, initialize and retrieve:

- Initialize: When the browser extension wallet is initialized for the first time, it initializes the vault with the password that the user provided. To unlock the vault the client must provide the initialization password.
- Retrieve: The unlock feature has been split from the retrieve feature. The vault must be unlocked by the client before the data can be retrieved. After the vault is unlocked, the client can call the retrieve method to get its data.

In the case of `ImportNotes` called by network change, if the `unlockPhrase` is not passed to `importNotes()`, the `unlock` method is not called again.

Verification

Resolved.

Suggestion 8: Resolve TODOs in the Code Base

Location

Examples (non-exhaustive):

blank-deposit/tornado/TornadoService.ts#L1658

blank-deposit/infrastructure/GenericVault.ts#L17

blank-deposit/tornado/config/config.ts#L1

blank-deposit/tornado/TornadoService.ts#L1571

Synopsis

There are many unresolved TODO items in the code comments of the in-scope repositories, which may lead to a lack of clarity and cause confusion about the completion of the implementation. In addition, TODOs should be resolved prior to a complete security audit of the code.

Mitigation

We recommend resolving TODO items in the code comments.

Status

The Blank team has resolved all the TODO items in the code base. All TODO items were reviewed and some were fixed while the remainder were added to an internal task list.

Verification

Resolved.

Suggestion 9: Improve Linting Rules to Improve Code Quality

Location

Examples (non-exhaustive):

`src/util/fetchWebsiteIcon.ts#L4`

`src/controllers/EnsController.ts#L21`

Synopsis

There are several instances where improved linting rules can improve the general code quality and avoid minor bugs if configured correctly.

Mitigation

We recommend extending the ESlint linter rules to all in-scope repositories.

Status

The Blank Team has enabled ESLint on the `extension-background` repository, which covers most of the code. It has been configured according to best practices guidelines for TypeScript and has been adapted for the Blank Wallet browser extension's use case. The Blank team has also noted that the errors reported have been examined and have implemented fixes where applicable.

Verification

Resolved.

Suggestion 10: Alert Users on Withdrawal Attempt Soon After Deposit

Synopsis

If a user deposits funds in the Blank Wallet browser extension and immediately tries to withdraw them to a fresh address, it is possible for an attacker to link the withdrawal address to the deposit address. Once a user's deposit is processed, it is recommended to wait until at least a few deposits are made thereafter, before withdrawing the note. The users should be alerted if they try withdrawing their notes immediately following a successful deposit.

Mitigation

We recommend incorporating alerts in the UI to caution the user before withdrawing their note if the withdrawal is initiated immediately after a deposit. Moreover, Tornado Cash recommends several steps for users to preserve anonymity and privacy. Some of those recommendations could be incorporated in the Blank Wallet browser extension for better protection against heuristic and other attacks.

Status

The Blank team has implemented changes to the withdrawal page to warn users about deanonymization risks resulting from withdrawing their latest deposit within 24 hours.

Verification

Resolved.

Suggestion 11: Use estimateGas to Estimate Deposit and Withdrawal Gas Costs

Location

`blank-deposit/tornado/TornadoService.ts#L1558`

`blank-deposit/tornado/TornadoService.ts#L1628`

Synopsis

In the Blank Wallet browser extension implementation, the deposit and withdrawal gas costs are hardcoded based on current empirical values. Hardcoded gas costs are difficult to update once the value changes and are not as accurate as genuine transaction gas costs. We suggest using `estimateGas` for gas estimation.

Mitigation

We recommend using `web3.eth.estimateGas` for gas cost estimation. In the case of the provider node breaking privacy by linking the deposit and withdrawal data, users can use a trusted Ethereum provider node or retrieve gas costs from a relayer.

Status

The Blank team has decided to keep the hardcoded values due to withdrawal and deposit gas limits being almost static. They have slightly adjusted the withdrawal gas limit to be in line with the `tornado-relayer config` file, and also assigned a separate variable to the withdrawal gas limit, in order to simplify changing it in the future as needed.

Verification

Resolved.

Suggestion 12: Improve Documentation

Location

`Blank-Wallet-extension-background/blob/master/README.md`

`Blank-Wallet-extension/blob/master/README.md`

`Blank-Wallet-extension-provider/blob/master/README.md`

`Blank-Wallet-extension-ui/blob/master/README.md`

`Blank-Wallet-blank-make/blob/main/README.md`

Synopsis

The provided documentation can be further improved by including detailed descriptions of security critical components, including the `extension-background` and `extension-provider` components. Similarly, a detailed and comprehensive description of the Blank Wallet browser extension transaction flow would facilitate an easier understanding of the expected behavior of the system. Finally, there is no documentation available informing users of concerns regarding Tornado Cash.

Mitigation

We recommend the following improvements to the project documentation:

- Document detailed descriptions of security critical components including the `extension-background` and `extension-provider` components;
- Document detailed and comprehensive description of the Blank Wallet browser extension transaction flow; and
- Expand the browser extension wallet user documentation to include a warning of potential vulnerabilities of Tornado Cash, along with links to resources such as how to stay anonymous.

Status

The Blank team has acknowledged this suggestion and are allocating developer time to improve the documentation in the future. Since the documentation has not been improved at the time of verification, this suggestion remains unresolved.

Verification

Unresolved.

Suggestion 13: Increase Code Comments

Location

Examples (non-exhaustive):

`blank-deposit/tornado/TornadoNotesService.ts#L619`

`src/controllers/blank-deposit/tornado/TornadoNotesService.ts#L651`

`src/controllers/blank-deposit/tornado/TornadoNotesService.ts#L424`

`blank-deposit/tornado/TornadoService.ts#L1140`

`blank-deposit/tornado/TornadoService.ts#L1704`

`src/controllers/IncomingTransactionController.ts#L80`

Synopsis

We found code comment coverage to be generally sufficient, however, some functions were found without descriptive code comments. Documentation contained within the code should be comprehensive and explain the intended functionality of each of the components (e.g. functions or entry points).

Mitigation

We recommend adding detailed descriptive code comments to each function in the code base.

Status

The Blank team has acknowledged this suggestion and are allocating developer time to increase code comments in the future. Since the documentation has not been improved at the time of verification, this suggestion remains unresolved.

Verification

Unresolved.

Suggestion 14: Reduce Implementation Complexity

Location

Example (non-exhaustive):

`src/controllers/blank-deposit/tornado/TornadoService.ts`

Synopsis

Our team found there are some instances where a high level of complexity is present in the code base. For example, the transaction flow of deposits to Tornado Cash can be summarized as:

```
blankController.approveDepositTransaction->blankDepositController.approveDepos
itTransaction ->TornadoService.approveDepositTransaction ->
SignedTransaction.approveTransaction ->
TransactionController.approveTransaction -> sendTransaction
```

Simple errors are more difficult to identify in a complex system and may go unnoticed by maintainers and security reviewers of the implementation. This may lead to the increased risk for severe security vulnerabilities that have devastating outcomes in wallets containing significant value.

Mitigation

We recommend that complexity in the implementation be reduced where possible.

Status

The Blank team has acknowledged this suggestion and agree that “reducing complexity and simplifying code should always be a goal and will work on achieving this in the future”. Since the complexity has not been changed at the time of verification, this suggestion remains unresolved.

Verification

Unresolved.

Appendix A

Suppose a Merkle Tree with leaves marked with yellow are the filled ones while the grey leaves are empty. Insert 4 new leaves at the 4 indices shown with arrows. The following figures show the state changes on insertion of 4 new leaves. Note that batch updates with Merkle tree work only when the number of leaves to be inserted is a power of two.

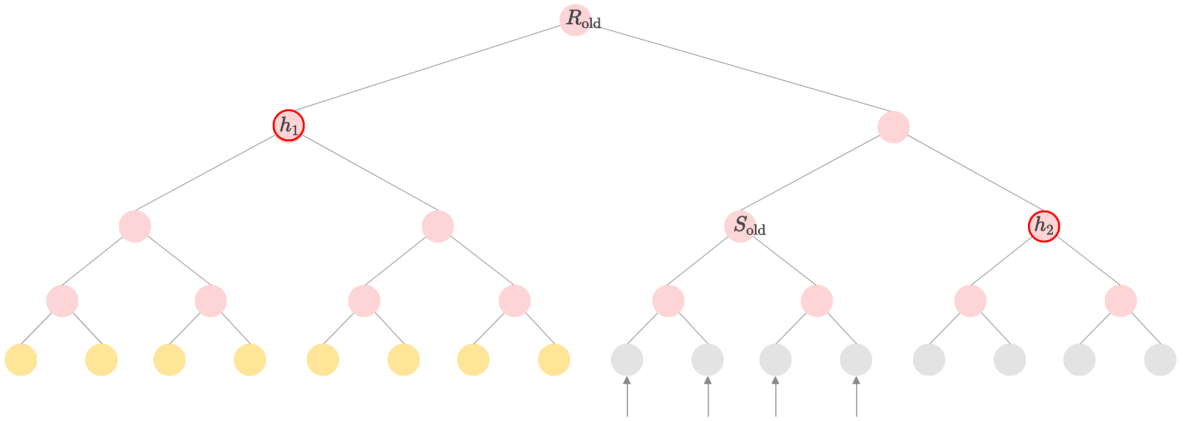

Figure 1: State before inserting 4 new leaves

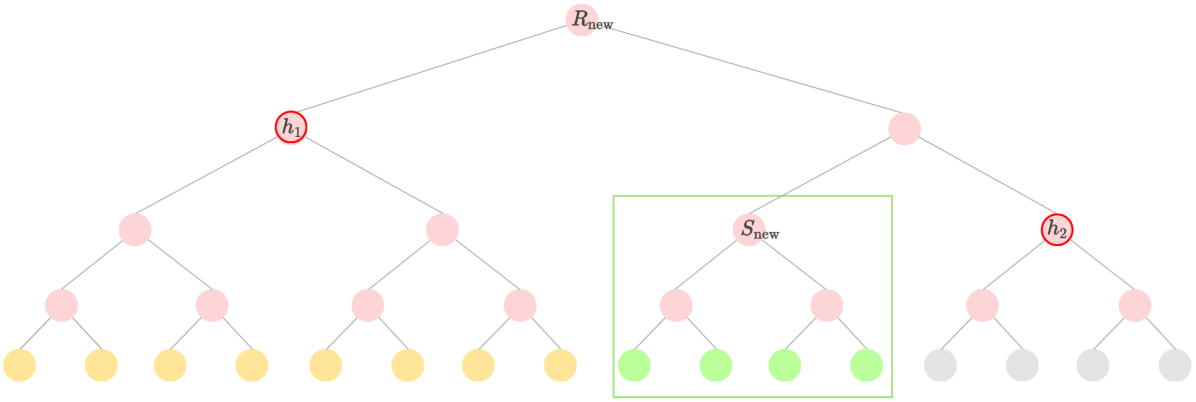

Figure 2: State after inserting 4 new leaves

Now, given the new leaves L = $\{l_1, l_2, l_3, l_4\}$, compute the sub-tree root $S_{new} = H(\{l_1, l_2, l_3, l_4\})$. To verify if the correct batch L was inserted from index 8, first compute the old sub-tree root $S_{old} = H(\{\varphi, \varphi, \varphi, \varphi\})$ and the old Merkle root R_{old}. From the old state of the tree, the partial Merkle path is known: $\{h_1, h_2\}$. All the verifier (circuit) needs to check is the following:

$$R_{new} == H(h_2, H(S_{new}, h_1))$$

$$R_{old} == H(h_2, H(S_{old}, h_1))$$

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.