Least Authority

PRIVACY MATTERS

WB Network Protocol

Security Audit Report

Whitebit

Final Audit Report: 15 August 2023

Table of Contents

Overview

Background

Whitebit has requested that Least Authority perform a security audit of their WB Network Protocol.

Project Dates

- **June 19, 2023 - July 19, 2023:** Initial Code Review *(Completed)*
- **July 21, 2023:** Delivery of Initial Audit Report *(Completed)*
- **August 15, 2023:** Verification Review *(Completed)*
- **August 15, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Shareef Maher Dweikat, Security Research and Engineer
- DK, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the WB Network Protocol followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

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

- `whitebit-exchange`: https://github.com/whitebit-exchange/wbt
  - In particular:
    - `core/vm/evm.go` (new method Mint with all required checks)
    - `core/state_transition.go` (changes for running Mint instructions)
    - `core/state_processor.go` and `miner/worker.go` (changes for executing state migrations)
    - `core/state/migrations.go` (state migrations implementation)
    - `core/state/migrations/mint_contract.go` (specific state migration that adds mint state contract bytecode with a predefined storage to a predefined address)
    - `core/mint/contract.go` (representation of predefined Mint state contract)
    - `core/mint/contract/MintState.sol` (Mint state contract source code)
    - `params/config`

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

- 6ca9efb2965a6604e44a37d7f97b1ddac2aa1116

For the verification, we examined the Git revision:

- bb50fbfe813e8fae9e014ec6709f737c8f0a7f58

For the review, this repository was cloned for use during the audit and for reference in this report:

- `whitebit-network-protocol-audit`: https://github.com/LeastAuthority/whitebit-network-protocol-audit

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:

- Website: https://whitebit.com
- WB Network Changes and Concept.pdf *(shared with Least Authority via Slack on 21 June 2023)*

In addition, this audit report references the following documents and links:

- Proof-of-Authority (PoA): https://eips.ethereum.org/EIPS/eip-225

Areas of Concern

Our investigation focused on the following areas:

- Correctness and security of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Security best practices implemented in the development of the system;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the protocol or disrupt its execution;
- Protection against malicious attacks and other ways to exploit the protocol;
- 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

Our team conducted a security audit of the Whitebit Protocol, a `go-ethereum` fork, which aims to provide a platform for transferring Whitebit tokens (WBT) between Ethereum, Tron, and the Whitebit networks. The Whitebit team made modifications to the `go-ethereum`, opting for a Proof-of-Authority (PoA) mechanism to achieve faster transactions with lower cost, in addition to implementing a method for minting new coins on the Whitebit Network.

During our review, our team examined the newly introduced changes implemented in the Whitebit Network Protocol to assess whether any vulnerabilities were introduced as a result of these changes. Specifically, our team reviewed the minting functionality to determine whether the changes are accessible to a set of authorized users only. Additionally, we checked the implementation of the set of addresses that is exclusively authorized to approve transactions. We found that security has been taken into consideration, as demonstrated by the proper use of access controls restricting unauthorized access to critical functions and contracts.

The implemented minting method relies on providing proof of WBT burning on Ethereum or other networks, which is then used to mint new coins on the Whitebit Network, and intends to allow only designated contract owners to perform such transactions until the `mintLimit` in the contract is reached.

Our team investigated the contract's susceptibility to spam attacks, in addition to the probability of the state migrations being subject to hacks or exploits, but did not identify any issues. Overall, our team found that the introduced changes are consistent with the blockchain implementation generally, and the consensus specifically, and did not find any issues relating to the state migration mechanism being utilized to update the state on the nodes in the network.

Code Quality

We performed a manual review of the Whitebit target code and found the code to be generally well-organized. During our review, we identified several areas of improvement in the quality of the code, including improvement of error handling (Suggestion 1) and the function and variable naming convention used (Suggestion 2). We also made suggestions on the use of hard coded constants (Suggestion 3) and the use of a switch clause (Suggestion 4). However, it is noted that these findings were identified in the original `go-ethereum` code, which was out of the scope of this audit.

Tests

The codebase implements sufficient test coverage.

Documentation

Our team found that the project documentation provided by the Whitebit team is sufficient and accurately describes the changes made to the `go-ethereum` fork.

Code Comments

We found sufficient code comments describing the intended behavior of security-critical functions and components.

Scope

The scope of this review was limited to the modifications made to the `go-ethereum` fork. As a result, our team assumed that out-of-scope components function as intended. In this context, we found the scope of the review to be sufficient.

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
Suggestion 1: Improve Handling of Exceptional Conditions and Limit and Avoid Using Panics (Out of Scope)	Unresolved
Suggestion 2: ImBookmarkprove Variable and Function Naming Convention	Unresolved

(Out of Scope)	
Suggestion 3: Do Not Hard Code Strings (Addresses)	Resolved
Suggestion 4: Combine Switch Cases (Out of Scope)	Unresolved
Suggestion 5: Refactor Code To Avoid Duplicate Functions (Out of Scope)	Unresolved

Suggestions

Suggestion 1: Improve Handling of Exceptional Conditions and Limit and Avoid Using Panics (Out of Scope)

Location

`worker.go#L1147`

`worker.go#L1138`

`worker.go#L1109`

`worker.go#L764`

`worker.go#L602`

`worker.go#L567`

`passphrase.go#L143`

Synopsis

Our team found some errors that are returned by the `w.commit` function but not handled. Such errors will be lost, which might create confusion or result in an unfavorable user experience. We also identified multiple instances in the code that would trigger a panic in case of an error. Functions that can cause the code to panic at runtime may lead to denial of service.

Mitigation

We recommend that all errors be caught and handled appropriately. Additionally, we recommend providing the user with helpful error descriptions as well as refactoring the code and removing panics where possible. One of the possible improvements is to propagate errors to the caller and handle them on the upper layers. Note that error handling does not exclude using panics. Moreover, if a caller can return an error, the callee function may not panic but, instead, propagate an error to the caller.

Status

The Whitebit development team acknowledged the finding but stated that it was identified in the original `go-ethereum` code and is therefore out of the scope of this audit. The Whitebit team stated that they decided not to refactor any `go-ethereum` code. Instead, they plan to follow the original `go-ethereum` repository to avoid any possible conflicts in the future.

Verification

Unresolved.

Suggestion 2: Improve Variable and Function Naming Convention (Out of Scope)

Location

core/vm/evm.go#L512

Synopsis

There are two functions with the same name, in the same file, but with different behavior (`Create` and `create`). This will create confusion for future maintainers of the codebase.

Mitigation

We recommend giving the functions more expressive names to make them easier to distinguish.

Status

The Whitebit development team acknowledged the finding but stated that it was identified in the original `go-ethereum` code and is therefore out of the scope of this audit. The Whitebit team stated that they decided not to refactor any `go-ethereum` code. Instead, they plan to follow the original `go-ethereum` repository to avoid any possible conflicts in the future.

Verification

Unresolved.

Suggestion 3: Do Not Hard Code Strings (Addresses)

Location

core/mint/contract.go#L45

core/state/migrations_test.go#L55

Synopsis

Some constants are hard coded, such as the hashes and add address. Constants that are hard coded in multiple locations can lead to mistakes during development, leading to different values throughout the codebase.

Mitigation

Important strings should be accessed from one source of truth in the codebase. We recommend creating a `strings` file to export strings for usage.

Status

The Whitebit team stated that they rewrote `state_migrations_test.go` to make it use already defined values. Furthermore, the team moved the `mint` contract components strings into the `strings` file to improve the readability of the code.

Verification

Resolved.

Suggestion 4: Combine Switch Cases (Out of Scope)

Location

[core/blockchain_test.go#L1441]

Synopsis

The aforementioned switch clause has multiple branches that are identical.

Mitigation

We recommend combining the identical branches in order to improve code quality.

Status

The Whitebit development team acknowledged the finding but stated that it was identified in the original `go-ethereum` code and is therefore out of the scope of this audit. The Whitebit team stated that they decided not to refactor any `go-ethereum` code. Instead, they plan to follow the original `go-ethereum` repository to avoid any possible conflicts in the future.

Verification

Unresolved.

Suggestion 5: Refactor Code To Avoid Duplicate Functions (Out of Scope)

Location

[core/blockchain_test.go#L392]

[core/vm/gas_table.go#L388]

[core/vm/memory_table.go#L73]

Synopsis

There are multiple occurrences of duplicate, identical functions. The difference between the functions lies in the context of the EVM usage. However, this results in poorer code quality.

Mitigation

We recommend refactoring the code to avoid duplication or explicitly describing the different purposes of each function.

Status

The Whitebit development team acknowledged the finding but stated that it was identified in the original `go-ethereum` code and is therefore out of the scope of this audit. The Whitebit team stated that they decided not to refactor any `go-ethereum` code. Instead, they plan to follow the original `go-ethereum` repository to avoid any possible conflicts in the future.

Verification

Unresolved.

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 multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. 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. We are an international team that believes 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 include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website 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. As 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 and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and 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 before having 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 and comprehensive 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 Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, 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 a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.