HACKEN

WHITEBIT SECURITY ASSESSMENT

Intro

Name	WhiteBIT
Website	https://whitebit.com/wbt
Repository	https://github.com/whitebit-exchange/wbt
Commit	f8d2d285df13dfc3c933266ff36905a5b0a8975e
Platform	L1
Network	Ethereum
Languages	Go
Methodology	Blockchain Protocol and Security Analysis Methodology
Lead Auditor	y.bratashchuk@hacken.io
Auditor	g.pakharenko@hacken.io
Approver	l.ciattaglia@hacken.io
Timeline	05.06.2023 - 26.06.2023
Changelog	26.06.2023 (Preliminary Report)
Changelog	14.07.2023 (Final Report)

Table of contents

Summary

WhiteBIT, a prominent player in the peer-to-peer cryptocurrency exchange space, has developed WB Network, a blockchain that is integral to the functioning of the WhiteBIT within the decentralized finance (DeFi) ecosystem.

As a derivative of Geth, WB Network node has been diligently crafted by WhiteBIT engineers, focusing on token minting, state migrations, and modifications to block sealing in support of Proof of Authority (PoA) consensus.

At Hacken, we have conducted a comprehensive audit of the amendments introduced by the WhiteBIT team, and analyzed potential vulnerabilities exposed in subsequent versions of Geth.

Documentation quality

The WB Network node codebase stands out for its extensive documentation across all components encompassed by our audit. The README section aligns seamlessly with the requirements. WhiteBIT engineers furnished detailed justifications throughout the audit process, specifically addressing their token minting approach.

While the rationale behind the changes to Clique block sealing was initially unclear, WhiteBIT's engineering team was able to satisfactorily elucidate their motivation and the challenge they were addressing.

The total Documentation Quality score is **10** out of 10.

Code quality

WB Network node is a testament to best practices in Go programming. The project successfully passed a linter run with default configurations, yielding no warnings for any glaring idiomatic Go issues. This stringent adherence to Go's best practices underlines the team's commitment to maintainability, readability, and stability within the codebase.

The token minting logic is robustly supported by unit tests, making it effortless to manipulate certain test cases for exploring various behaviors.

However, we encountered difficulties verifying the MintState contract runtime bytecode and its hash, owing to the absence of adequate instructions.

While the initial code showed a lack of robust error handling and logging in the minting logic, WhiteBIT made requisite updates during the audit. They also provided Hacken's team with the compiler config which greatly eased bytecode reproduction and verification. Also, the issue related to error handling during token minting (WBT-102) was promptly addressed by WhiteBIT engineers.

In regards to block sealing changes, the lack of testing raised concerns regarding their viability and potential deadlock conditions, given their reliance on blocking and concurrency primitives. After conducting our own tests, we gained a clear understanding of the block sealing changes, which were subsequently reverted by WhiteBIT as they didn't resolve block reorganization issues (See WBT-103 issue) as anticipated.

The total Code Quality score is **10** out of 10.

Architecture quality

Built on the foundation of Geth, WB Network node boasts of a well-structured architecture.

Initially, the token minting architecture needed to be clarified, appearing as a workaround rather than an organic part of the overall architecture. However, post-audit WhiteBIT updated documentation to leave no questions regarding architecture and also refactored the code to harmonize better with the existing codebase. They opted not to pursue our suggestion of native contracts due to the increased implementation complexity it would entail.

State migrations have been implemented flexibly, with the capacity to perform diverse migrations at different block heights.

It's important to note that block sealing modifications disrupted the miner worker's and sealing subroutines' concurrent functioning. Despite not causing any deadlock conditions, it was evident this didn't solve the block reorganization problem, leading to its subsequent reversion by WhiteBIT.

The architecture quality score is **10** out of 10.

Security score

Our extensive analysis of WB Network node did not reveal any issues of high severity.

The majority of the issues raised were addressed in the course of the audit.

The prime area of concern was issue WBT-100, which we strongly advised addressing and the WhiteBIT team took this into account after the audit.

A secondary concern was that token minting is controlled manually by the owner of the MintState contract. This approach exposes the process to human error and we strongly advised against this manual operation. In response, WhiteBIT team added sufficient clarifications to the minting documentation and the white paper.

The final security score is **10** out of 10.

Total score

Considering all metrics, the total score of the report is **10.0** out of 10.

Findings count and definitions

Severity	Findings	Severity Definition
Critical	0	Vulnerabilities that can lead to a complete breakdown of the blockchain network's security, privacy, integrity, or availability fall under this category. They can disrupt the consensus mechanism, enabling a malicious entity to take control of the majority of nodes or facilitate 51% attacks. In addition, issues that could lead to widespread crashing of nodes, leading to a complete breakdown or significant halt of the network, are also considered critical along with issues that can lead to a massive theft of assets. Immediate attention and mitigation are required.
High	0	High severity vulnerabilities are those that do not immediately risk the complete security or integrity of the network but can cause substantial harm. These are issues that could cause the crashing of several nodes, leading to temporary disruption of the network, or could manipulate the consensus mechanism to a certain extent, but not enough to execute a 51% attack. Partial breaches of privacy, unauthorized but limited access to sensitive information, and affecting the reliable execution of smart contracts also fall under this category.
Medium	1	Medium severity vulnerabilities could negatively affect the blockchain protocol but are usually not capable of causing catastrophic damage. These could include vulnerabilities that allow minor breaches of user

		privacy, can slow down transaction processing, or can lead to relatively small financial losses. It may be possible to exploit these vulnerabilities under specific circumstances, or they may require a high level of access to exploit effectively.
Low	3	Low severity vulnerabilities are minor flaws in the blockchain protocol that might not have a direct impact on security but could cause minor inefficiencies in transaction processing or slight delays in block propagation. They might include vulnerabilities that allow attackers to cause nuisance-level disruptions or are only exploitable under extremely rare and specific conditions. These vulnerabilities should be corrected but do not represent an immediate threat to the system.
Total	4	

Scope of the audit

Protocol Audit

WB Network node implementation (fork of Geth v1.10.26)

- Genesis, docs, consensus, fork mechanism

Geth (changes after v1.10.26)

- Bugs and vulnerabilities introduced in later versions of Geth

Code Quality

- Static Code Analysis
- Tests coverage

Protocol Tests

Node Tests

- Environment Setup
- E2E sync tests

Issues

Missing backports for vulnerable dependencies and DoS mitigations

Important changes and bug fixes were not backported from later Geth versions (after 1.10.26).

ID	WBT-100
Scope	Security, Code Quality
Severity	**MEDIUM**
Vulnerability Type	Missing backports for vulnerable dependencies and DoS mitigations.
Status	Fixed (5bbc6d60)

Description

Several important changes and bug fixes were implemented in later versions of Geth (after 1.10.26). They include the following key pull requests (PRs) from the Geth git repository:

PR 27038

https://github.com/ethereum/go-ethereum/pull/27038

It cleans the `commitTransaction` function in the miner/worker. Whenever the error occurs after running the transaction, this transaction should be regarded as invalid and all following transactions from the same sender won't be executed because of the nonce restriction.

Also, cleanup of garbage improves DoS resilience (other PRs in this finding explicitly mention this risk).

PR 26907

https://github.com/ethereum/go-ethereum/pull/26907

It's an alternative solution for checking if the transaction is executable quickly. The quotation: ..."Notably, the code iterates the queue list from the head and aborts if there is any nonce gap between pending and queue. It's indeed expensive to iterate the list and in worst case can iterate the entire list if all txs in the queue are executable. But it's impossible for attackers to leverage this point seems they need to actually send executable transactions which are costly."...

Even if comments to this PR consider the attack to have a low exploitability, this feature should be backported.

PR 26920

https://github.com/ethereum/go-ethereum/pull/26920

A previous audit raised an issue related to a crash when parsing Solidity anonymous events (resulting in the LOG0 opcode). Quotation: ..."We had an audit done of a fork of this repo and the auditors raised this issue. I was split on whether to explain why this wasn't an issue (which it is not in the way that it's used here) or to add the suggested check. I decided that adding the check would make the correctness of this code much more clear at very minimal cost and since it's used as a library as well, that seemed like a good tradeoff."...

Though the finding was doubtful, the check was included in the Geth repository, so the diligent strategy is to merge it as well.

PR 26845

https://github.com/ethereum/go-ethereum/pull/26845

This PR fixes a stack overflow when tracing is enabled. Bugs like this must be fixed.

PR 26926

https://github.com/ethereum/go-ethereum/pull/26926

This is an upgrade of the net library which contains the LOW severity issue CVE-2022-41723. A maliciously crafted HTTP/2 stream could cause excessive CPU consumption in the HPACK decoder, sufficient to cause a denial of service from a small number of small requests. Additional details on the fix can be found here: https://github.com/ethereum/go-ethereum/pull/26960.

PR 26928

https://github.com/ethereum/go-ethereum/pull/26928

This is an upgrade of the influxdb dependency, which contains a disputed LOW severity CVE-2022-36640.

PR 26912

https://github.com/ethereum/go-ethereum/pull/26912

This is a fix of permissions when creating a chainspec file, so it is not globally writable. It presents a kind of a minor weakness that should be fixed.

PR 26862

https://github.com/ethereum/go-ethereum/pull/26862

This is a crash fix that is wise to merge from the availability perspective. Related issues and additional details are here:

- https://github.com/ethereum/go-ethereum/issues/26830
- https://github.com/ethereum/go-ethereum/pull/26699

PR 26671

https://github.com/ethereum/go-ethereum/pull/26671

Implement fixes for memorizable errors in the trie db. At least the comment tells that there is a possibility for malicious activity. Quote: ..."Lookups caused by (potentially malicious) eth/snap trie requests must not trigger memorised error...". Additional details are here:

- https://github.com/ethereum/go-ethereum/pull/26670

PR 26648

https://github.com/ethereum/go-ethereum/pull/26648

This is DoS fix. Additional details are in the research papers and related PR:

- https://github.com/ethereum/go-ethereum/pull/26311
- https://tristartom.github.io/docs/ccs21.pdf
- https://dl.acm.org/doi/pdf/10.1145/3460120.3485369

PR 26773

https://github.com/ethereum/go-ethereum/pull/26773)

The issue seems to affect only the hardware Ethereum wallets. This opens a potential to sign accidentally non EIP-155 transactions. Quote: "Because Ledger's RLP deserialization code does not validate the length of the RLP list received, it may prematurely enter the signing flow when APDU chunk boundary falls immediately before the EIP-155". Another comment states that user funds are not affected.

Related issue:

- https://github.com/LedgerHQ/app-ethereum/issues/409

PR 26791

https://github.com/ethereum/go-ethereum/pull/26791

Fixes the gas allowance, which was wasted by invalid transactions, so fewer transactions were included in the block. It can be escalated to DoS; therefore, it should be fixed.

Related issue:

- https://github.com/ethereum/go-ethereum/pull/26799

PR 26633

https://github.com/ethereum/go-ethereum/pull/26663

Update of the abandoned file locking dependency. Backport will reduce the risk in the future.

In general, releases after 1.10.26 contain the following important from security perspective changes:

- Migration from the LevelDB backend to Pebble. The LevelDB backend has poor support from only one maintainer who is no longer interested in investing his time into the project. This is a source of bugs in the future.
- Protection against transaction censorship, when the block is assembled by one set of validators and signed by another.

Impact

Unaddressed PRs can expose vulnerabilities, inviting potential denial of service attacks. Their effects largely depend on the customer's business model. However, for entities like real-time cryptocurrency exchanges or DApp platforms, even minor disruptions can erode customer trust, potentially causing lasting financial harm.

Recommendation

We recommend to backport all PRs listed above, as well as implement monitoring of nodes availability and network health.

Changes to Block Sealing Logic Ineffective in Mitigating Block Reorgs

ID	WBT-103
Scope	Code Quality and Performance
Severity	LOW
Status	Fixed (2da668)

Description

Upon a thorough review and testing of the modified block sealing logic, we've found that the changes intended to mitigate block reorgs are ineffective. The changes disrupt the concurrency, causing block reorgs to persist even when a new block sealing task arrives and cancels an unfinished previous task. As a result, rather than preventing block reorgs, the changes merely reject previously submitted but unfinished sealing tasks.

Further, our tests verified that there are no deadlock conditions resulting from these changes. However, the absence of comprehensive comments explaining these changes creates ambiguity around the block sealing function's rationale and expected improvements.

Recommendation

Considering these findings and their adverse impact on the network's performance and reliability, we recommend reverting the changes made to the block sealing function. Once reverted, consider raising an issue with the go-ethereum project, outlining the identified inefficiencies and the inability of the recent changes to effectively mitigate block reorgs.

Should there be a need for improvements in the block sealing function, they should be carefully tested to prevent the introduction of new issues or exacerbation of block reorgs. Any significant modifications ought to be submitted to the go-ethereum project for review and inclusion, to contribute to the overall improvement of the platform.

Architectural Concerns in Mint Instructions Implementation

ID	WBT-102
Scope	Code Structure and Design
Severity	**LOW**
Status	Fixed (6ca9e)

Description

Upon review of the Mint Instructions introduced in `core/mint/instruction.go`, several areas of concern have been identified that necessitate attention:

1. **Transaction Data Size Inconsistency**: The current tx data size of 65 bytes is unachievable through the combination of different ABI datatypes but can be reached without them. The nearest achievable size using ABI datatypes is 68 bytes (32 - mint amount, 32 - burn tx hash, 4 - burn tx network).

2. **Inconsistency in code comments**: The `ParseInstruction` function is inconsistent with its comments. The `mint contract address contains expected mint contract code` condition check is not implemented.

3. **Missing Verification of Burn Transaction**: The Mint Instruction mandates the presence of a burn transaction hash and network. However, the current code lacks a mechanism to verify that the provided values exist on the specified network.

4. **Error Handling Discrepancy**: The `ParseInstruction` function handles errors in a manner inconsistent with the existing codebase. Rather than returning to the block, as events in the contracts do, errors are redirected to the Node log. For effective detection of attackers and prevention of DoS, it's best to maintain logs in the block and charge a substantial amount of gas.

5. **Inconsistent Minting Implementation**: The current approach of minting, established at the very early stage prior to EVM processing and carrying a gas usage of "0" in the `Apply` function, doesn't align with the rest of the codebase. More background information and justifications are required for this implementation, or alternatively, minting could be implemented as part of `core/vm/contracts.go`. This would place it alongside other native Go contracts with deterministic gas usage.

Recommendation

We recommend a thorough re-evaluation of the above issues. Primarily, a consistent strategy for transaction data size should be established, possibly involving ABI datatypes. Error handling should be uniform with the rest of the codebase, ensuring errors are returned to the block. Finally, consider rethinking the minting implementation in line with current codebase conventions or provide justifications for the existing approach.

For guidance on implementing native contracts, refer to this article: Dissecting EVM Using Go Ethereum (ETH) Client Implementation Part II: EVM

Inconsistencies Detected in MintState Contract Bytecode

The Bytecode of the MintState contract is not up to date with `MintState.sol`.

ID	WBT-101
Scope	Code Quality
Severity	**LOW**
Status	Fixed (6ca9efb)

Description

A discrepancy has been identified between the MintState contract's bytecode (hardcoded into the `core/mint/contract.go` file) and the `MintState.sol`. Notably, the string literal "Available only for owner" is absent in the decompiled bytecode of the MintState contract.

Moreover, there appears to be a typo in the existing string literal, with it currently reading "Zero address is not a valid owne", where the letter "r" is evidently missing from the word "owner".

Recommendation

We recommend defining and documenting a standardized approach for the compilation of the `MintState.sol`. It would be ideal to establish a method that allows for the regeneration of the file containing the hardcoded MintState contract runtime bytecode. This would ensure consistency and accuracy between the source code and the compiled bytecode.

Disclaimers

Hacken disclaimer

The code base provided for audit has been analyzed according to the latest industry code quality, software processes and cybersecurity practices at the date of this report, with discovered security vulnerabilities and issues the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functional specifications). The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code (branch/tag/commit hash) submitted to and reviewed, so it may not be relevant to any other branch. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits, public bug bounty program and CI/CD process to ensure security and code quality. English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical disclaimer

Protocol Level Systems are deployed and executed on hardware and software underlying platforms and platform dependencies (Operating System, System Libraries, Runtime Virtual Machines, linked libraries, etc.). The platform, programming languages, and other software related to the Protocol Level System may have vulnerabilities that can lead to security issues and exploits. Thus, Consultant cannot guarantee the explicit security of the Protocol system in full execution environment stack (hardware, OS, libraries, etc.)