XPRESS PROTOCOL SECURITY AUDIT REPORT

March 28, 2025

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

This project implements a novel primitive: an on-chain limit order book exchange for decentralized trading built on an EVM-based blockchain using Solidity. The protocol enables users to execute trades and claim their order results while supporting fees and rebates, batch orders, native token trading, and the option to keep tokens inside the contract to further save gas. The exchange's innovative architecture leverages an efficient radix tree data structure, enabling a logarithmic complexity matching engine that can handle a large volume of orders per trade and deliver significant gas savings compared to linear matching engine DEXs.

1.4 Project Dashboard

Project Summary

Title	Description
Client	XPress Protocol
Project name	OnchainCLOB
Timeline	27.05.2024 - 23.01.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
28.05.2024	09b6188e028650b9c1758010846080c5f8c80f8e	Initial commit for the audit
09.08.2024	70b15ec4d9e757824814160450384371 6a67d875	Commit for the second stage of the audit
18.09.2024	38982d883c1408d19e1b39a47a9ac6327204d1fe	Commit for the reaudit
01.10.2024	f699fdc9d9d74f699757cbdc8872f3d909ad5832	Commit with WatchDog feature
20.01.2025	4c8237da4ec356ad1a96b7308c1a8ae5f41b23a5	Deployed commit

Project Scope

The audit covered the following files:

File name	Link
src/HanjiTrie.sol	HanjiTrie.sol

File name	Link
src/HanjiLOB.sol	HanjiLOB.sol
src/HanjiLOBFactory.sol	HanjiLOBFactory.sol
src/HanjiTrieFactory.sol	HanjiTrieFactory.sol
src/HanjiErrors.sol	HanjiErrors.sol
src/HanjiFP24.sol	HanjiFP24.sol
src/HanjiHelper.sol	HanjiHelper.sol
src/HanjiWatchDog.sol	HanjiWatchDog.sol
src/HanjiWatchDogFactory.sol	HanjiWatchDogFactory.sol

Deployments

File name	Contract deployed on Sonic
WatchDogFactory.sol	0xf0c9f1...965c2833
Proxy.sol	0x14F9C8...722cc387
WatchDog.sol	0x0e9fa0...bc204f53
TrieFactory.sol, Trie.sol	0xA30646...cCCB5fC3
OnchainCLOBFactory.sol, OnchainCLOB.sol	0xB9A10c...0Bda243f
Helper.sol	0x38e577...d753f8c0

1.5 Summary of findings

Severity	# of Findings
Critical	4
High	4
Medium	8
Low	25

ID	Name	Severity	Status
C-1	Several reentracy scenarios	Critical	Fixed
C-2	Incorrect Condition for `post_only` Orders	Critical	Fixed
C-3	Potential Data Corruption in Trie Structure	Critical	Fixed
C-4	Incorrect accounting of commissions	Critical	Fixed
H-1	Incorrect Handling of Fee-on-Transfer Tokens	High	Fixed
H-2	Potential Overflow in Accumulator Variables	High	Fixed
H-3	Potential Read-Only Reentrancy	High	Fixed
H-4	Lack of rebasable tokens support	High	Acknowledged
M-1	Commission rates can be set to 100%	Medium	Fixed
M-2	Lack of market maker address check	Medium	Fixed
M-3	Lack of Upgradeability and Pausable Functionality	Medium	Fixed
M-4	Potential Accumulation of Unused Commissions Due to Rounding	Medium	Fixed

M-5	Lack of The Expiry Parameter	Medium	Fixed
M-6	Centralization risks	Medium	Acknowledged
M-7	Anyone can set the `marketmaker` address	Medium	Fixed
M-8	Potential failure in token permit execution	Medium	Fixed
L-1	Redundant Restricting Modifier in View Function	Low	Fixed
L-2	Redundant Conditions and Operations	Low	Fixed
L-3	Consider Making `public` Functions `external`	Low	Fixed
L-4	Initialization and Non-Zero Address Checks	Low	Fixed
L-5	Event Emission Issues	Low	Fixed
L-6	SafeMath and SafeCast Usage	Low	Fixed
L-7	File Name Mismatch	Low	Fixed
L-8	Consider Defining Commission Scaling Factor as a Constant	Low	Fixed
L-9	Use Ownable-Two-Step Pattern for Administrator Contract	Low	Fixed
L-10	Incomplete Documentation for Function `placeOrder`	Low	Fixed
L-11	Inefficient Commission Accounting for Administrator and Market Maker	Low	Fixed
L-12	Lack of View Functions for Key Data Retrieval	Low	Fixed
L-13	Streamline Error Handling with `require` Statements	Low	Fixed
L-14	Incorrect Initialization of Nonce	Low	Fixed
L-15	Incorrect Naming of Error `ClaimstatusDoesntAllowNonOwnerClaims`	Low	Fixed
L-16	Inefficient encoding of uint56 to uint24 floating point representation	Low	Fixed

L-17	Inefficient initalization in UUPS proxy	Low	Fixed
L-18	Incorrect interface check for market maker contract	Low	Fixed
L-19	Insufficient validation for ETH as trading token	Low	Fixed
L-20	Improper timing of WETH deposit operation	Low	Fixed
L-21	Redundant status variable in `HanjiTrieFactory`	Low	Fixed
L-22	Gas optimization for marketmaker callbacks	Low	Fixed
L-23	Immutable variables can be initialized in the constructor	Low	Acknowledged
L-24	Potential risk of double-free memory	Low	Fixed
L-25	Use of native ETH `transfer` limits gas availability	Low	Fixed

1.6 Conclusion

During the initial phase of our audit, we identified several significant security issues. Many of them emerged partly due to the project's novelty of implemented smart-contracts, which neccessitated substantial changes by the development team increasing the source lines of code (SLOC) by 50%. The subsequent audit of the revised code demonstrated marked improvements in code quality, with no high or critical severity issues detected.

The project under review implements a smart-contract-based order book using a modified Radix-tree data structure. This design ensures that all order book operations are performed with low gas usage, optimizing efficiency and cost-effectiveness.

The scope of the audit consists of the following contracts:

- **Trie**: The core data structure responsible for storing and updating orders.
- **OnchainCLOB**: The user interface that handles user interactions for placing and claiming orders.
- **WatchDog**: A monitoring contract that monitors the network and can pause the system if it is not live.
- **Helper**: A utility contract providing view functions to assist clients in querying order book data.

The users of the project can be categorized into three main types:

- **Active traders**: These users place market orders or limit orders that can be partially executed. They pay fees proportional to the executed amount of tokens to the contract owner, market maker (if present), and passive traders. Active traders drive the trading volume and liquidity in the market.
- **Passive traders**: These users place limit orders without immediate execution, providing liquidity to the market. Depending on deployment settings, they can pay fees or receive "passive payout" fees proportional to the executed amount of tokens. Passive traders are essential for maintaining market depth and stability.
- **Market makers**: These users can act as both active and passive traders. Optionally can be configured to receive callbacks after placing orders depending on deployment parameters, giving them a more advantageous position to control the market. Additionally, market makers earn fees from active traders for their role in enhancing market liquidity. Their presence is crucial for a healthy and dynamic trading environment.

Traders can utilize the `LOB` contract as a centralized exchange. This contract allows balance changes to be accounted for without calling `transfer` after each order placement or claim, thereby saving gas. This feature enhances the system's efficiency, making it more appealing for frequent traders, but requires the traders to trust in the project's security.

The main attack vectors and concerns examined in the audit include:

1. **Architecture of the Project**:

    - The price value is stored in a small type (uint24). If the price exceeds this type's limit, the contracts become ineffective and need redeployment for a new price interval. It is recommended to consider an alternative price model, such as mantissa with exponent or a Uniswap v3-like price formula (1.0001^price). These changes will increase the contract's flexibility but require careful coding into the

tree nodes. As of the revised code, the team has introduced a wider price range encoded in exponent/mantissa format.
- The use of a mutable data structure that can remove subtrees poses additional risks. Local data corruption can affect the entire tree, impacting all users. Mitigating this risk can involve adopting a lazy propagation method instead of deleting nodes, which localizes potential problems. Additionally, it is advised to enhance the project's robustness by increasing the independence of orders from each other during claim operations. To address this, the team will proactively montior tree integrity, pausing the contract if anomaliees are detected. Additionally, the team improved the test coverage to mitigate incorrect tree operations.

2. **Reentrancy**:

   - The protocol may use ERC-777/ERC-667 tokens, yet the functions are not marked as `nonReentrant`. This oversight could lead to reentrancy attacks using ERC-777 hooks for double spending. To mitigate these risks, it is advisable to implement `nonReentrant` and Check-Effects-Interaction patterns in the relevant functions. The revised code includes reentrancy safeguards, reducing these risks.

3. **Support of Non-Standard ERC-20 Tokens**:

   - The current implementation does not support rebasing ERC-20 tokens or tokens with transfer fees. Using such tokens could lead to critical problems and potential losses. To mitigate these risks, it is recommended to add safeguards to validate token behaviour and consider implementing mechanisms to securely support these types tokens. The revised code implements a `_safeTansferFromWithBalanceCheck` function to validate token behavior and prevent unexpected losses.

4. **Frontrunning and Sandwich Attacks**:

   - Users set the maximum or minimum price they are willing to pay when placing an order, preventing frontrunning/backrunning activity beyond the specified price bound. To further enhance security, it is advisable to add an expiry parameter to prevent the placement of outdated orders when transactions remain pending in the blockchain memory pool for extended periods, which has been implemented in the updated contracts.

5. **Centralization Risks**:

   - Owners can configure the contracts during initialization and modify the market maker, resulting in low centralization risks. However, due to the novel architecture, it is recommended to temporarily increase governance capabilities by making the project upgradeable and pausable for all user methods except for claiming orders. In a revised version, pausing also disables claiming due to potential tree-related risks. Once the contract is thoroughly battle-tested, the multi-sig can renounce their governance rights mitigating the centralization risk.

6. **Math Calculations**:

- Several issues related to incorrect fee distribution were identified and described in detail in the second section of this report. Moreover, the tree nodes store the sum accumulator of all values in the subtree using the same number of bytes as individual values, which may lead to potential overflow problems and contract lock-up. To mitigate these risks, it is recommended to increase the type size to enhance the project's reliability and adaptability, even if it requires additional storage slots. In the revised contracts, the relevant storage sizes have been increased to 128 bit to mitigate these concerns further.

Implementing the recommendations and addressing the issues outlined in the second section of this report will significantly enhance the project's robustness and resilience. To further improve the project's integrity and reliability, it is advisable to take the following actions:

- **Write Comprehensive Unit Tests** that cover all possible branches within each function. This includes testing positive scenarios where the function performs as expected and negative scenarios where the function encounters errors or edge cases.
- **Perform Stress Tests and Simulations** that verify the system's stability under extreme scenarios, over prolonged periods and under various market conditions. These tests should mimic real-world usage patterns and transactions. This will help identify potential performance bottlenecks and ensure the system maintains data integrity and correct balances.
- **Implement Fuzzing Tests** to provide random and unexpected inputs to the core algorithms, helping to uncover hidden bugs and vulnerabilities that may not be evident through standard testing.
- **Employ SMT checking** to formally verify the core algorithm against specified invariants and properties. This will help ensure the algorithm behaves correctly under all possible conditions.
- **Integrate Monitoring Mechanisms** to track the project's performance and health in real-time. This includes monitoring transaction processing times, error rates, and other critical metrics. Implementing alerting mechanisms to notify the development team promptly of any anomalies or critical issues enables a timely response to potential problems.
- **Deploy on a Test-Net** to validate its functionality in conditions that closely resemble the main-net. This helps identify and fix issues that may not arise in isolated development environments. Perform extensive testing on the test-net to ensure all features work as expected and that the project can handle real-world scenarios and user interactions.
- **Conduct Periodic Independent Audits** of the project's codebase. These audits will provide an objective assessment of the system's security posture and identify any new vulnerabilities or weaknesses.
- **Create a Bug Bounty Program** to incentivize the broader security community to identify and report vulnerabilities. This can significantly enhance the project's security by leveraging the expertise of individual researchers.

By following these steps diligently, the project can continuously improve its security, reliability, and overall performance. This proactive approach ensures the system remains robust, user-friendly, and trustworthy over the long term.

2.FINDINGS REPORT

2.1 Critical

C-1	Several reentracy scenarios
Severity	Critical
Status	Fixed in 70b15ec4

Description

The issue is identified within the `LOB` contract.

Some ERC-20 tokens implement extensions, such as the ERC-777 standard, that allow an account to gain execution control during token transfers. As the protocol is planned to be deployed on several networks and support various tokens, it may encounter issues related to these types of tokens.

If an attacker gains control during a token transfer, they can launch a "reentrancy" attack against the protocol by following these steps:

1. Set up send/receive hooks.
2. Perform an action that causes the protocol to send or receive tokens.
3. Execute additional calls to the protocol, exploiting the inconsistent state from step 2 to gain profit.

At a minimum, the following scenarios can be exploited:

- Withdraw tokens, reenter the protocol, and withdraw the same tokens again, effectively double-spending.
- Deposit tokens, reenter the protocol with the `placeOrder` function, and double-spend the tokens.

This issue is classified as **critical** severity because it can lead to significant financial loss through double-spending.

Recommendation

We recommend applying the OpenZeppelin `nonReentrant` modifier to all public and external mutable functions to prevent reentrancy attacks. Additionally, consider updating the trader's balance before performing the transfer operations.

Client's commentary

> *Fixed. Followed the check-effects-interaction pattern.*

C-2	Incorrect Condition for `post_only` Orders
Severity	Critical
Status	Fixed in 70b15ec4

Description

The issue is identified within the OnchainLOB.sol#L230 function of the `LOB` contract.

There is a critical vulnerability in the `post_only` order handling logic. The incorrect condition in the `if` statement means that `post_only` bid orders will always be executed, causing the bidder to pay commissions that are not intended for passive orders in this architecture. This can lead to unintended financial costs for users.

The test named `testPostOnly` provided by the developers does not catch this issue because it does not properly check the execution path.

The issue is classified as **critical** severity, because it can lead to significant unintended financial consequences for users placing `post_only` orders.

Recommendation

We recommend flipping the sign in the `if` statement condition to ensure that `post_only` orders are handled correctly:

```
if (price_id <= askTree.best_offer()) {
    return 0x0;
}
```

C-3	Potential Data Corruption in Trie Structure
Severity	Critical
Status	Fixed in 70b15ec4

Description

The issue is identified within theTrieLib.sol#L780-L795 function of the `Trie` contract.

There is a critical vulnerability where the `matchesNode` function checks if the `order_id` potentially exists in the matching node, but it doesn't verify the `found` boolean value before updating the subtree. If the `order_id` does not actually exist in the matching node, it can lead to a violation of the validity and consistency of the trie, resulting in data corruption and potential loss of funds.

The issue is classified as **critical** severity because it can cause severe data corruption and financial loss.

Recommendation

We recommend checking the `found` boolean value before updating the subtree to ensure that the `order_id` actually exists in the matching node. This will prevent inconsistencies and maintain the integrity of the trie.

C-4	Incorrect accounting of commissions
Severity	Critical
Status	Fixed in 70b15ec4

Description

In the OnchainLOB.sol#L261 function, the `total_commission` value is accounted for twice: once as the commission and once as the `token_y` value of the trader. This duplication can lead to incorrect commission accounting and potential contract malfunction due to mismatched accounted and actual balances. Additionally, it may allow the market maker account to exploit the commission system for profit.

This issue is rated as critical severity as it may lead to malfunction due to contract state inconsistency and enable the market maker to gain undesired profit.

Recommendation

We recommend addressing the commission accounting error in the `placeOrder` function to prevent discrepancies in balance accounts and potential exploitation for profit.

Client's commentary

> *A new logic was implemented for calculating fees, which accurately distributes even the "dust" left after rounding.*

2.2 High

H-1	Incorrect Handling of Fee-on-Transfer Tokens
Severity	High
Status	Fixed in 70b15ec4

Description

The issue arises within the `receiveTokens` function of the `LOB` contract .

There is a vulnerability related to the improper handling of fee-on-transfer tokens. The function does not verify that the balances have been adjusted as expected after the `safeTransferFrom` calls. This oversight can result in incorrect token balances, especially when dealing with tokens that impose transfer fees or employ other mechanisms that alter the expected transfer amount.

The issue is classified as **high** severity as it can lead to discrepancies in token balances, potentially causing financial inconsistencies and loss.

Recommendation

We recommend adding a check to ensure that the balances reflect the expected changes after the `safeTransferFrom` calls. If the balances do not match the expected change, the transaction should be reverted to safeguard against financial irregularities.

H-2	Potential Overflow in Accumulator Variables
Severity	High
Status	Fixed in 70b15ec4

Description

The issue is pinpointed within the TrieLib.sol#L10 struct of the `Trie` contract.

There is a vulnerability due to the use of `uint64` variables for `total_shares` and `total_value` accumulators. When the admin sets a small scaling factor or the price of one asset compared to another raises significantly, these values can overflow, leading to a hault in adding new orders. This overflow risk poses a grave threat to the functionality and reliability of the contract.

The issue is classified as **high** severity as it can immobilize the system, preventing new orders from being added and interrupting standard operations. However, it does not lead to permanent losses since the orders can still be claimed/cancelled.

Recommendation

We recommend either storing `total_shares` and `total_value` in larger uint types such as `uint128` to accommodate larger sums and setting explicit requirements for the `scaling_factor` parameters during initialization to avert overflow.

Client's commentary

> *It has been fixed as recommended, switching to uint128. As mentioned earlier, the issue was mitigated by the tree structure. Additional gas costs were reduced by "reusing" utilized storage words through a simple memory manager.*

H-3	Potential Read-Only Reentrancy
Severity	High
Status	Fixed in 70b15ec4

Description

The OnchainLOB.sol#L154 function is vulnerable to read-only reentrancy attacks, especially with ERC-777 tokens. This vulnerability arises because the trader's balance updates in storage occur after token transfers, potentially allowing a reentrancy exploit.

The issue is classified as **high** severity because it causes discrepancies between the actual balance and the returned value from the view function, which could lead to critical issues and vulnerabilities in the project's integrations.

Recommendation

We recommend updating the trader's balance prior to transferring the external tokens to mitigate the read-only reentrancy risk.

Client's commentary

> *Fixed, the Check-Effects-Interaction pattern was used.*

H-4	Lack of rebasable tokens support
Severity	High
Status	Acknowledged

Description

The issue is identified in the OnchainLOB.sol contract.

For rebasable tokens, such as those that adjust their supply over time (e.g., Ampleforth), the LOB contract may keep all rewards in the contract balance. During a rebase (positive or negative), the recorded balances in the contract may not accurately reflect the true balances of the tokens held. This discrepancy can lead to scenarios where, after a negative rebase (slashing), the remaining users attempting to withdraw may not be able to receive their full share of assets because the contract's actual balance is less than the sum of the recorded balances.

Recommendation

To handle rebasable tokens properly, the contract should dynamically adjust internal records to reflect rebases. This can be done by recalculating balances based on the current total supply and the contract's actual balance.

Client's commentary

> *The inclusion of support for rebasable tokens would have unnecessarily complicated the contract's logic and significantly increased gas consumption. Considering that this feature offers little practical benefit, we have decided not to implement it.*

2.3 Medium

M-1	Commission rates can be set to 100%
Severity	Medium
Status	Fixed in 70b15ec4

Description

The issue is identified within the OnchainLOB.sol#L90 of the `LOB` contract.

The commissions (`admin_commission`, `marketmaker_commission`, `passive_order_payout`) can be set to 100%, which presents a centralization risk. Such a configuration could potentially be exploited by the contract administrator.

Recommendation

We recommend implementing checks to ensure that the total commission does not exceed a reasonable percentage of the total value. Additionally, consider decentralizing the control over these parameters.

M-2	Lack of market maker address check
Severity	Medium
Status	Fixed in 70b15ec4

Description

A flaw has been detected within the OnchainLOB.sol#L313 function of the `LOB` contract.

If the market maker is not a smart contract that implements the `onTrade` function and the `should_invoke_on_trade` flag is set to true, calling the `placeOrder` function will result in a revert. This occurs because the contract attempts to call onTrade on the market maker's address, expecting it to be a valid function call.

Recommendation

We recommend adding checks (in both the constructor and the `changeMarketMakerAddress()` function) that the market maker's address is indeed a smart contract and implements the `onTrade` function.

Client's commentary

> *Added with IERC165 support.*

M-3	Lack of Upgradeability and Pausable Functionality
Severity	Medium
Status	Fixed in 70b15ec4

Description

The issue is identified within the OnchainLOB.sol contract.

There is an issue due to the lack of upgradeability and pausable functionality. While adding these features introduces additional centralization risks, they are crucial for handling scenarios where the contract becomes malfunctioned. The pausable function, in particular, allows halting all operations except for order removal, ensuring orderly migration to a new contract version.

The issue is classified as **medium** severity because it impacts the flexibility and maintainability of the contract, potentially leading to prolonged downtime and operational risks in case of malfunctions.

Recommendation

We recommend implementing upgradeability and pausable functionalities using OpenZeppelin's `AccessControl`, `Pausable`, and `Ownable` contracts. To mitigate centralization risks in the future, it is advised to use renounceable deployer and pause roles once the project has been battle-tested.

M-4	Potential Accumulation of Unused Commissions Due to Rounding
Severity	Medium
Status	Fixed in 70b15ec4

Description

The issue is pinpointed within the OnchainLOB.sol#L543 function of the `LOB` contract.

There is an issue related to the accumulation of unused commissions due to rounding operations, which can lead to significant residual amounts, especially over prolonged contract operation scaled by `token_y_scaling_factor`.

The issue is classified as **medium** severity because while the accumulated amounts might seem minor individually, they can sum up to a significant value over time, leading to potential discrepancies and inefficiencies in the commission calculation.

Recommendation

We recommend calculating the `admin_amount` as `total_commission_amount - marketmaker_amount - passive_order_payout` to reduce dust accumulation and remove the unused residual amounts. This adjustment will ensure more accurate commission distribution.

Client's commentary

> *Implemented a new commission calculation mechanism that fully utilizes "dust."*

M-5	Lack of The Expiry Parameter
Severity	Medium
Status	Fixed in 70b15ec4

Description

The issue arises within the OnchainLOB.sol#L199 function of the `LOB` contract.

There is an issue due to the absence of an expiration parameter for orders. Without this parameter, there is no guarantee that an order is added to the blockchain before a specified timestamp, which can be problematic if a transaction remains pending for a long time. This can lead to unintended execution of stale orders, potentially causing financial losses or market disruptions.

The issue is classified as **medium** severity because it affects the timing and validity of order execution, which can impact the reliability and efficiency of the trading platform.

Recommendation

We recommend adding an `expires` parameter to the `placeOrder` function. This parameter should specify a timestamp before which the order must be added to the blockchain. If the transaction is executed after this timestamp, it should be reverted to ensure timely and valid order execution.

M-6	Centralization risks
Severity	Medium
Status	Acknowledged

Description

The issue has been identified within the HanjiLOB.sol#L55 and HanjiWatchDog.sol contracts.

The current design of the contract grants significant centralized control to the administrator and pauser roles. The administrator has extensive powers, including the ability to upgrade the contract to any arbitrary address, pause or unpause the contract, and configure critical settings. The pauser role can also pause and unpause the contract. This concentration of control introduces centralization risks, as a single compromised or malicious administrator or pauser could disrupt or manipulate the contract, leading to potential security breaches or operational failures. Additionally, allowing the pauser role to unpause the contract could be risky if the contract is in an unstable state, as it could inadvertently enable harmful operations.

Moreover, the owner of the HanjiWatchDog contract can upgrade the contract's implementation. If the new implementation is flawed or if the HanjiWatchDog contract is compromised, the creation of new orders in the HanjiLOB contract may be blocked.

The issue is classified as **medium** severity due to the potential impact on the contract's security and operational integrity if these roles are exploited.

Recommendation

We recommend using a multisig wallet for both the administrator and pauser roles. This would require multiple approvals for critical actions like upgrading the contract, pausing, or unpausing, thereby reducing the likelihood of malicious or erroneous actions. Additionally, we advise disabling the ability for the pauser role to unpause the contract, reserving this capability solely for the administrator role under the protection of a multisig wallet. This would help ensure that the contract can only be resumed under secure and verified conditions.

Client's commentary

> *Multisig will be used.*

M-7	Anyone can set the `marketmaker` address
Severity	Medium
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L313 function of the `HanjiLOB` contract.

The function allows the administrator to change the market maker address and includes a flag, `_should_invoke_on_trade`, that determines whether callbacks are invoked after each trade for the `marketmaker` address. However, the function also permits anyone to change the market maker address if the current market maker address is set to zero. This presents a security risk as it could enable any actor to set a malicious `marketmaker` address with the `_should_invoke_on_trade` flag set to `true`. This setup could allow them to engage in sandwich attacks and claim additional fees, thereby manipulating market operations.

The issue is classified as **medium** severity because it introduces a potential vulnerability that could be exploited to interfere with the market's integrity.

Recommendation

We recommend restricting the ability to change the market maker address exclusively to the owner of the contract, even if the current market maker address is set to zero.

M-8	Potential failure in token permit execution
Severity	Medium
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L428, HanjiLOB.sol#L552, and HanjiLOB.sol#L754 functions of the `HanjiLOB` contract.

These functions call the `permit` method on an ERC20 token contract to grant spending approval. However, if the `permit` function has already been executed in a previous transaction, it may revert, causing the entire transaction to fail. This could lead to significant disruptions, particularly in a high-frequency trading environment where consistent operation is critical.

The issue is classified as **medium** severity because it can cause transaction failures, negatively impacting user experience and potentially resulting in missed trading opportunities.

Recommendation

We recommend wrapping the `permit` function call with a `try-catch` block or implementing a conditional check to ensure the transaction does not revert if the `permit` call fails. This approach would allow the `placeOrder` function to continue executing even if the permit has already been granted in a previous transaction, thereby ensuring smoother operation.

```
try token.permit(msg.sender, address(this), value, expires, v, r, s) {
    // Permit executed successfully
} catch {
    // Handle the case where permit execution fails
    // Consider logging or ignoring the failure based on business logic
}
```

Alternatively, consider implementing a mechanism that checks if the permit is already valid by ensuring the required allowance has been met and only executes the `permit` function if necessary.

2.4 Low

L-1	Redundant Restricting Modifier in View Function
Severity	Low
Status	Fixed in 70b15ec4

Description

The issue is identified within the TrieLib.sol#L155 function of the `Trie` contract.

This function has a redundant `onlyLOBPermission` modifier. Since this is a view function, it should not require restrictive permissions. This redundancy complicates the code without adding necessary security or functionality.

Recommendation

We recommend removing the `onlyLOBPermission` modifier from the `getOrderInfo` function to simplify it. This will enhance code readability and maintainability without compromising security.

L-2	Redundant Conditions and Operations
Severity	Low
Status	Fixed in 70b15ec4

Description

Several redundant conditions and operations were spotted across various functions:

- In the TrieLib.sol#L216 function, condition `order_id > node_id` is always false and can be removed.
- In the TrieLib.sol#L836 function, the bitwise OR operation `|` is redundant when casting to `uint64`.
- In the TrieLib.sol#L335 function, the `new_rightmost_map` is always assigned to `rightmost_map` in the usages (TrieLib.sol#L455, TrieLib.sol#L529) of this function. Updating `rightmost_map` directly without returning `new_rightmost_map` is recommended.
- In the TrieLib.sol#L269, TrieLib.sol#L684 and TrieLib.sol#L734 functions, the operations can be simplified by using `nodes[node_id].right = new_right_child` instead of loading the node first.
- The OnchainLOB.sol#L601 function could be optimized to return the nonce before subtraction.
- In the OnchainLOB.sol#L316 function the `return order_id` statement is redudant, as it is already implicitly marked to return.
- In the OnchainLOB.sol#L511 function, setting the new trader_id can be simplified to `trader.trader_id = last_used_trader_id++`
- In the OnchainLOB.sol#L691-L692 function, the `trader` parameter is passed by memory, which is unnecessary as it can be modified directly by reference. Additionally, the `trader_changed` argument is redundant and not needed.
- The calls to `SafeERC20.safeTransfer|From` can be made more readable by using syntax `using SafeERC20 for IERC20;`. This allows the call to be written as `token_x.safeTransfer|From(...)`,
- The while loop in the TrieLib.sol#L833 function uses an inefficient bitwise operation to find the highest bit set. Utlizing OpenZeppelin's `Math.log2` function can perform this operation more efficiently and save gas.
- The usages of the TrieLib.sol#L857 function are equivalent to condition `order_id < node_id`, making the custom function call redundant.

Recommendation

We recommend optimizing the operations enlisted above to improve the readability and reduce gas costs.

L-3	Consider Making `public` Functions `external`
Severity	Low
Status	Fixed in 70b15ec4

Description

Several functions in the OnchainLOB.sol contract that are marked as `public` could be declared as `external` to save gas costs and improve performance. It is particularly relevant for functions that are not called internally within the contract.

Recommendation

We recommend changing the visibility of the following functions from `public` to `external` where appropriate: `getConfig`, `getTraderBalance`, `changeMarketMakerAddress`, `firstLevel`, `setClaimableStatus`, `batchPlaceOrder`, `getOrderInfo`, `batchClaim`, `batchChangeOrder`, `depositTokens` and `withdrawTokens`.

L-4	Initialization and Non-Zero Address Checks
Severity	Low
Status	Fixed in 38982d88

Description

Several critical parameters and addresses are not validated upon initialization. Specifically, there are no validations in the `LOB` OnchainLOB.sol#L62 if the token addresses (`_tokenXAddress`, `_tokenYAddress`), administrator address (`_administrator`), and market maker address (`_marketmaker`) are non-zero before proceeding with contract execution.

Recommendation

We recommend adding checks to ensure that these addresses are non-zero during the contract's initialization to prevent potential issues.

L-5	Event Emission Issues
Severity	Low
Status	Fixed in 70b15ec4

Description

There are several instances of missing and improper event emissions in the `LOB` contract:

- The OnchainLOB.sol#L164 function does not emit an event after changing the market maker address.
- The OnchainLOB.sol#L305 and OnchainLOB.sol#L307 events use `quantity` which might not represent the actual executed quantity.
- The OnchainLOB.sol#L446 and OnchainLOB.sol#L473 functions don't emit events after changing balances

Recommendation

We recommend adding event emissions where necessary and optimizing the redundant code to improve the contract's efficiency and transparency.

Client's commentary

> *Events added.*

L-6	SafeMath and SafeCast Usage
Severity	Low
Status	Fixed in 70b15ec4

Description

The contract does not utilize Solmate's FixedPointMathLib or OpenZeppelin's SafeCast libraries for arithmetic operations, which could streamline the readability of the code.

Recommendation

We recommend using FixedPointMathLib and SafeCast libraries to handle arithmetic operations securely.

L-7	File Name Mismatch
Severity	Low
Status	Fixed in 70b15ec4

Description

Contracts OnchainLOB.sol and TrieLib.sol do not match the file names in which they are implemented. This discrepancy can create confusion for developers and auditors, as it becomes challenging to locate and reference the contract accurately. Consistency between file names and contract names is essential for code readability, maintenance, and ensuring accurate auditing processes.

Recommendation

We recommend aligning the contract names with the file names in which they are implemented. This practice enhances code readability, simplifies maintenance, and facilitates accurate auditing.

L-8	Consider Defining Commission Scaling Factor as a Constant
Severity	Low
Status	Fixed in 70b15ec4

Description

The OnchainLOB.sol#L51 is set as an `immutable` variable, but it is typically a constant value (e.g., `1e6`) in many contracts. Using a constant instead of an immutable variable can optimize gas usage and improve clarity.

Recommendation

We recommend setting the `commission_scaling_factor` as a constant with a value of `1e6`.

Client's commentary

> *The standard factor of 1e18 was chosen.*

L-9	Use Ownable-Two-Step Pattern for Administrator Contract
Severity	Low
Status	Fixed in 70b15ec4

Description

The contract currently uses an immutable OnchainLOB.sol#L84 address initialized through the `Governance` contract. However, adopting the Ownable-Two-Step pattern for managing ownership and administrative tasks offers enhanced flexibility and security benefits.

Recommendation

We recommend implementing the Ownable-Two-Step pattern using the OpenZeppelin library. This pattern allows for smoother management of the administrator role, ensuring better flexibility and security in contract operations.

L-10	Incomplete Documentation for Function `placeOrder`
Severity	Low
Status	Fixed in 70b15ec4

Description

The documentation for the OnchainLOB.sol#L193 function is missing important information regarding the `quantity` parameter. Specifically, it does not mention that the `quantity` will be multiplied by the scaling factor.

Recommendation

We recommend updating the documentation for the `placeOrder` function to explicitly state that the `quantity` parameter provided by users will be multiplied by the scaling factor before execution.

Client's commentary

> *Docstrings have been revised, and additional useful views have been added to the helper for retrieving the actual price.*

L-11	Inefficient Commission Accounting for Administrator and Market Maker
Severity	Low
Status	Fixed in 70b15ec4

Description

The OnchainLOB.sol#L582 function updates commissions by modifying the `Trader` struct for both the administrator and the market maker. A more efficient approach would be to account for these commissions in specific storage variables rather than within the `Trader` entity, simplifying commission management and reducing potential errors.

Recommendation

We recommend creating specific storage variables to account for the administrator's commissions instead of relying on the `Trader` struct.

Client's commentary

> *The commission calculation mechanism has been completely redesigned, as mentioned above.*

L-12	Lack of View Functions for Key Data Retrieval
Severity	Low
Status	Fixed in 70b15ec4

Description

The `LOB` contract lacks view functions for essential data retrieval, such as calculating the actual price taking into account scaling factors and fetching orders by `trader_id`. These functions would enhance the usability and transparency of the contract.

Recommendation

We recommend implementing additional view functions to:

1. Calculate the actual price considering scaling factors.
2. Retrieve orders by `trader_id`.
3. Determine the number of shares and corresponding value that can be executed without modifying the trie and existing orders.

Client's commentary

> *Fixed. Many useful views have been added to the helper contract.*

L-13	Streamline Error Handling with `require` Statements
Severity	Low
Status	Fixed in 70b15ec4

Description

As of Solidity 0.8.26, the preferred method for error handling is using the `require(bool, Error)` statements instead of `if (bool) revert Error`. This modern approach not only streamlines the code but also enhances is readability and gas efficiency

Recommendation

To align with current Solidity best practices, we recommend updating the error handling mechanism in the `LOB` contract to use the `require(bool, Error)` statements instead of `if (bool) revert Error`.

L-14	Incorrect Initialization of Nonce
Severity	Low
Status	Fixed in 70b15ec4

Description

The issue is identified within the OnchainLOB.sol#L57 contract.

The initialization of the `nonce` variable is incorrect. The current implementation `uint64(1) << nonce_length - uint64(1)` sets it to `2^38` instead of `2^39 - 1`.

Recommendation

We recommend initializing `nonce` as `(1 << nonce_length) - 1` to set it correctly to `2^39 - 1`.

L-15	Incorrect Naming of Error `ClaimstatusDoesntAllowNonOwnerClaims`
Severity	Low
Status	Fixed in 70b15ec4

Description

The `ClaimstatusDoesntAllowNonOwnerClaims` error is incorrectly named, which may lead to confusion for developers and users of the contract. The name does not conform to standard naming conventions and does not provide a clear understanding of the error's purpose.

Recommendation

To improve clarity and consistency, we recommend renaming the error to align with standard naming conventions. For example, the error can be renamed to `ClaimNotAllowed`. This change will ensure that the error name clearly reflects the nature of the issue, enhancing readability and comprehension for stakeholders.

Client's commentary

> *Fixed.*

L-16	Inefficient encoding of uint56 to uint24 floating point representation
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L1317 function of the `HanjiLOB` contract.

The current implementation converts a `uint56` number into a `uint24` floating point representation, but the method is not optimized for encoding efficiency. The approach involves multiple loops and checks that could be streamlined. Additionally, the encoding for exponents lower than 6 is implemented inefficiently, leading to suboptimal use of the `uint24` bits. Since negative exponents are not used for prices, the bits allocated for exponents in the range of 1 to 6 could be repurposed. If the mantissa is less than the threshold number, the exponent should be set to 0 instead of adjusting it based on the number of digits. This would increase the acceptable price range by a factor of 5.

The issue is classified as **Low** severity because, while it does not pose a critical risk, it could result in unnecessary gas consumption and reduced performance.

Recommendation

We recommend refactoring the encoding logic to reduce complexity and improve efficiency. The proposed implementation below optimizes the function by using the exponent more effectively, thereby expanding the price range that can be packed:

```
function _packFP24(
    uint72 a
    ) internal pure returns (uint24) {
    require(0 < a &&
            a <= 999999000000000000000,
            HanjiErrors.InvalidPriceRange());

    uint72 threshold_number = 999999;
    uint8 e = 0;

    while (a > threshold_number) {
        require(
            a % 10 == 0,
            HanjiErrors.ExcessiveSignificantFigures());
        a /= 10;
        e++;
    }

    uint24 p = (uint24(e) << 20) | uint24(a);
    return p;
}

function _unPackFP24(
    uint24 p
    ) internal pure returns (uint72) {
    uint72 e = uint72(p >> 20);
    uint72 a = uint72(p & 0xfffff);

    require(0 < a &&
            a <= 999999,
            HanjiErrors.InvalidFloatingPointRepresentation());
    uint72 f = uint72(10 ** e);
    a *= f;

    return a;
}
```

L-17	Inefficient initalization in UUPS proxy
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the dummy HanjiLOB.sol#L225 function of the `HanjiLOB` contract.

The current implementation follows the UUPS proxy pattern, which requires the initialization of the implementation contract to be disabled to prevent its accidental or malicious reinitialization. The current approach involves implementing an additional `initialize` function that sets all parameters to 0. However, a more efficient and straightforward method is to use the `_disableInitializers()` function provided by OpenZeppelin's `Initializable` contract, called directly from the `constructor`. This approach achieves the same goal while simplifying the contract's deployment process and improving clarity.

The issue is classified as **low** severity because it does not pose a critical risk but could improve contract efficiency and clarity.

Recommendation

We recommend refactoring the contract by replacing the `initialize` function with the following constructor:

```
constructor() {
    _disableInitializers();
}
```

L-18	Incorrect interface check for market maker contract
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L1088 function of the `HanjiLOB` contract.

The function changes the market maker address and checks whether the new market maker implements the `ITradeConsumer` interface by using the ERC165 `supportsInterface` function. However, the current implementation uses the `onTrade.selector` to perform this check, which is not the standard approach for ERC165 interface identification. Instead, the correct method is to use the `ITradeConsumer.interfaceId`, which represents the full interface rather than a single function.

The issue is classified as **low** severity because, while it may not cause immediate failures, it deviates from the ERC165 standard and could potentially lead to compatibility issues with other contracts that correctly implement the standard.

Recommendation

We recommend updating the interface check to use `ITradeConsumer.interfaceId` instead of `onTrade.selector`. This change ensures that the check is consistent with the ERC165 standard and accurately verifies that the market maker implements the entire `ITradeConsumer` interface.

```
if (!maker.supportsInterface(type(ITradeConsumer).interfaceId)) {
    revert HanjiErrors.InvalidMarketMaker();
}
```

L-19	Insufficient validation for ETH as trading token
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L942 function of the `HanjiLOB` contract.

The function includes a validation step to ensure that if `msg.value` is not zero, the token being traded must be ETH (as indicated by `supports_native_eth`). However, this check has only been partially implemented. Specifically, the function does not verify whether the `isAsk` flag corresponds with the expected behavior when ETH is involved. This oversight could lead to unexpected outcomes, such as ETH being mistakenly considered the trading token. This could result in either the full return of the transferred ETH to the `msg.sender` or unintentionally storing it on the trader's balance.

The issue is classified as **low** severity because it may not directly cause failures but could lead to logical inconsistencies in how orders are processed, particularly concerning ETH.

Recommendation

We recommend enhancing the validation logic to include a check that ensures `isAsk` and `supports_native_eth` are consistent with the expected token behavior. Specifically, if `msg.value` is non-zero, the function should ensure that ETH is correctly identified as the trading token and that the logic behaves accordingly.

L-20	Improper timing of WETH deposit operation
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L998 function of the `HanjiLOB` contract.

The function processes WETH deposits based on the `msg.value` passed in the transaction. However, the `weth.deposit{value: actual_value}()` operation occurs before the completion of all transfer-related operations. This can lead to a potential redundant `weth.deposit` followed by a subsequent `weth.withdraw` operation within the `_handleTokenTransfer` function if the `msg.value` exceeds the required amount of tokens to be sent to the contract. Such inefficiencies can lead to unnecessary gas consumption and complicate the logic.

The issue is classified as **low** severity because, while it does not immediately threaten contract functionality, it can lead to inefficiencies and potential logical errors in the contract's operation.

Recommendation

We recommend moving the `weth.deposit` operation to the `_handleTokenTransfer` function to ensure that all necessary validations and operations are successfully completed before the deposit is made.

L-21	Redundant status variable in `HanjiTrieFactory`
Severity	Low
Status	Fixed in 38982d88

Description

The issue has been identified within the HanjiLOBFactory.sol#L101 and HanjiTrieFactory.sol#L46 contracts.

These contracts exhibit inefficiencies and redundant operations in the deployment process. In the `HanjiLOBFactory` contract, deploying the `HanjiLOBProxy` involves unnecessary steps, such as toggling the `TrieFactory` status on and off by calling the `TrieFactory.setStatus` function. The status that controls whether the `TrieFactory` can create the trie is redundant, as this ability does not impact the security of the system. The `createTrie` function can be safely marked as callable by anyone, making the status control unnecessary.

The issue is classified as **low** severity because it does not directly threaten the security of the contracts but can lead to inefficiencies and potential errors in the deployment logic.

Recommendation

We recommend refactoring the deployment logic to eliminate the need for toggling the `TrieFactory` status. Specifically:

1. Remove the redundant `setStatus` function from the `HanjiTrieFactory` contract.
2. Streamline the `createHanjiLOB` function to deploy the `HanjiLOBProxy` more effectively without requiring intermediate status changes in the `TrieFactory`.

L-22	Gas optimization for marketmaker callbacks
Severity	Low
Status	Fixed in 38982d88

Description

The issue is identified within the HanjiLOB.sol#L122 contract.

The storage variables `marketmaker` and `should_invoke_on_trade` are frequently accessed together. Gas usage can be optimized by placing these two variables in the same storage slot, reducing the number of storage read and write operations:

```
struct MarketmakerConfig {
    address marketmaker;
    bool should_invoke_on_trade;
}
```

Additionally, the bytecode and code readability can be improved by moving the check and the call to `ITradeConsumer(marketmaker).onTrade(isAsk)` into a separate internal function, thereby reducing the code dublication.

This issue is classified as **low** severity because, while it does not pose a security risk, it can lead to gas savings, especially in contracts that perform frequent operations.

Recommendation

We recommend reworking the marketmaker callbacks as described above by grouping the related variables in a struct and refactoring the callback logic into a separate internal function.

L-23	Immutable variables can be initialized in the constructor
Severity	Low
Status	Acknowledged

Description

The issue is identified within the HanjiLOB.sol#L55 contract.

Variables with the `immutable` modifier do not use storage, which makes them more gas-efficient. In the context of a proxy pattern, it is still acceptable to initialize such variables in the constructor instead of the `initialize` function. Doing so can result in significant gas savings when these variables are accessed.

In this case, the following variables can be marked as `immutable` and initialized in the constructor to optimize gas usage:

- `token_x`
- `token_y`
- `scaling_factor_token_x`
- `scaling_factor_token_y`
- `supports_native_eth`
- `is_token_x_weth`
- `askTrie`
- `bidTrie`

The issue is classified as **low** severity because, while it does not pose a security risk, optimizing these variables can lead to more efficient contract execution and reduced gas costs.

Recommendation

We recommend marking these variables as `immutable` and initializing them in the constructor to achieve better gas efficiency.

Client's commentary

> *Acknowledged. Not implemented due to contract size constraints.*

L-24	Potential risk of double-free memory
Severity	Low
Status	Fixed in 38982d88

Description

The issue has been identified within the HanjiTrie.sol#L1274 function of the `HanjiTrie` contract.

Currently, there is no check to verify whether a pointer has already been freed before the `freeMemory()` function is called. Although the current implementation does not have any double `freeMemory()` calls or other destructive actions involving the `arena` mapping, future modifications to the codebase could introduce such risks. If a pointer is freed more than once, the `last_free_pointer` variable may be set to the index of `arena` element where the `last_free_pointer` is also stored, causing the `allocateMemory()` function to repeatedly return the same pointer, leading to potential memory corruption or unexpected behavior.

The issue is classified as **low** severity because it does not pose an immediate risk but could result in critical errors or vulnerabilities if the code is modified in the future.

Recommendation

We recommend adding a check in the `freeMemory()` function to ensure that the pointer being freed is valid and has not already been freed.

L-25	Use of native ETH `transfer` limits gas availability
Severity	Low
Status	Fixed in 38982d88

Description

This issue has been identified within the HanjiLOB.sol#L1001 and HanjiLOB.sol#L1457 functions of the `HanjiLOB` contract.

These functions use the `transfer` method inside to handle the native ETH transfers. The `transfer` function restricts the gas available for the receiving contract's function to 2300 gas. If the receiver has any logic in its `fallback` or `receive` function that requires more than 2300 gas, the transaction will revert. This limitation can lead to unexpected failures and disrupt the contract's operation, especially when interacting with complex contracts.

The issue is classified as **low** severity because it does not pose an immediate risk but can lead to transaction failures in specific scenarios involving more complex receiving contracts.

Recommendation

We recommend replacing the use of `transfer` with `call`, which doesn't limit the amount of gas to forward. This approach is safer and more flexible, accommodating receivers that may need more than 2300 gas to execute their logic.

Example:

```
(bool success, ) = msg.sender.call{value: msg.value}("");
require(success, "Transfer failed");
```

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes