Yes or no
Preliminary Audit Report

Fri Aug 01 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Yes or no Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	Event contracts for predicting the market
Type	DEX
Auditors	ScaleBit
Timeline	Thu Jul 10 2025 - Fri Aug 01 2025
Languages	Solidity
Platform	Others
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/yes-or-no-labs/yes_or_no_contracts/
Commits	1fbd4f8a93fa928de954fbe750dac5c0d51fd40b af7f2378d856dcbd4944b18b0907e173ac7f787f

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
PRO1	src/Proxy.sol	415a670956219f671db7942866c1882029332d65
UTO	src/token/UsdtToken.sol	dd56b76fa49e02c122c74ef9be27242bd7dade1b
GBA	src/GuessBase/GuessBase.sol	1a1cfde0f1137c15768e34e80168ee50bb28dd21
GMA	src/GuessMarket/GuessMarket.sol	4d12c3612d26864e163edb1fdddfbd9211aaecaf

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	1	1	0
Minor	3	3	0
Medium	2	2	0
Major	0	0	0
Critical	0	0	0
Discussion	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Chenmo to identify any potential issues and vulnerabilities in the source code of the Yes or no smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
GBA-1	Parent Contract `GuessTokenBase` Lacks Storage Gap for Upgradeability	Medium	Fixed
GBA-2	`settleCondition` Loss Of Agreement Income	Medium	Fixed
GBA-3	Unnecessary `payable` Modifier in ERC20-Based Betting Function	Minor	Fixed
GBA-4	`call()` Should be Used Instead of `transfer()` on An Address Payable	Minor	Fixed
GBA-5	Inconsistent `globalNonce` Increment Timing	Minor	Fixed
GBA-6	`_GuessTokenBase_init` Parameter Is Misspelled	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Yes or no Smart Contract :

Admin

- `addEventEmitter` : Administrator adds an account with the role of "event publisher". Accounts with this role can create new prediction events and conditions.
- `addEventSettler` : Administrator adds an account for the role of "Event Settlement Officer". Accounts with this role can settle the concluded prediction conditions.
- `setAcceptedToken` : Set up the ERC20 token address for the platform to accept bets.
- `setPlatformFee` : Set the proportion of handling fees charged by the platform during settlement.
- `withdraw` : Used to withdraw all the ETH balance in the contract
- `withdrawErc20` : Used to withdraw any amount of ERC20 tokens specified in the contract

Emitter

- `sendGuessEvent` :Create a new prediction event
- `addCondition` : Add specific prediction conditions for the created events

Settler

- `settleCondition` : Used to announce the result of a certain prediction condition (Yes/No)

User

- `bet` : The user selects an option (Yes/No) for a certain condition of the specified event and places a certain number of shares
- `claim` : After the condition settlement, the user calls this function to claim the reward

- `buyToken` : Users purchase the platform's designated bet tokens (acceptedToken) by sending ETH.
- `claimMon` : Users redeem ETH by authorizing and destroying accepted tokens

4 Findings

GBA-1 Parent Contract `GuessTokenBase` Lacks Storage Gap for Upgradeability

Severity: Medium

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#12

Descriptions:

The `GuessTokenBase` contract is designed to be upgradeable and serves as a parent contract for `GuessMarket`. It defines its own state variables (`eventNonce` , `platformFee` , `events` , etc.) which follow a standard linear storage layout. However, it does not include a storage gap (`__gap`) at the end of its state variable declarations.

While its parent contracts from OpenZeppelin (v5+) use the ERC-7201 standard to prevent storage collisions, `GuessTokenBase` 's own variables are not protected. If a future upgrade requires adding new state variables to `GuessTokenBase` , it will shift the storage layout of any child contracts like `GuessMarket` , leading to critical storage corruption and potential loss of all contract data.

Suggestion:

To ensure safe upgradeability, add a storage gap to the `GuessTokenBase` contract. This reserves storage slots for future use without breaking the storage layout of its child contracts.

Add the following line at the end of all state variable declarations within the `GuessTokenBase` contract:

```
contract GuessTokenBase is /* ... */ {
    // ... all other state variables
```

```
    uint256[] public eventIds;

    // Add the storage gap here
    // slither-disable-next-line Unused-state-variable
    uint256[50] private __gap;

    // ... functions
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GBA-2 `settleCondition` Loss Of Agreement Income

Severity: Medium

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#369

Descriptions:

In the 'settleCondition' function, the calculation of the platform fee thisFee uses integer division, resulting in the result being rounded down. This will cause systematic revenue loss because any fee less than one of the minimum token units will be rounded to zero, which will accumulate over a large number of transactions and lead to a loss of protocol revenue

```
    function settleCondition(uint256 eventId, uint256 conditionId, uint8 result) external
nonReentrant onlyRole(EVENT_SETTLER_ROLE) {
        Condition storage c = events[eventId].conditions[conditionId];
        require(!c.settled, "Already settled");
        c.result = result;
        c.settled = true;
        if (uintToOption(c.result) == Option.Yes) {
            uint256 thisFee = ((c.totalPaid[Option.No] * platformFee) / 10000);
            c.totalReward = c.totalPaid[Option.No] - thisFee;
            c.thisTimeProtocolFee = thisFee;
        } else {
            uint256 thisFee = ((c.totalPaid[Option.Yes] * platformFee) / 10000);
            c.totalReward = c.totalPaid[Option.Yes] - thisFee;
            c.thisTimeProtocolFee = thisFee;
        }
        globalNonce++;
        //
        emit ConditionSettled(globalNonce, eventId, conditionId,
            result, c.totalReward, c.thisTimeProtocolFee,
            msg.sender);
    }
```

Suggestion:

Change the calculation to rounding up

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GBA-3 Unnecessary `payable` Modifier in ERC20-Based Betting Function

Severity: Minor

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#268

Descriptions:

The `bet()` function allows a user to place a bet on a specific event and condition by selecting an option and staking a certain amount of ERC20 tokens.

```
function bet(
    uint256 eventId,
    uint256 conditionId,
    uint8 optionUint,
    uint256 amount
) external payable nonReentrant returns (bool){
    GuessEvent storage e = events[eventId];
    require(e.id != 0, "Event not found");
    require(block.timestamp >= e.startTime, "Event not started");
```

However, the function is marked with the payable modifier, which is unnecessary and misleading because the function does not make use of msg.value or handle native ETH in any way.

Suggestion:

It is recommended to remove the `payable` modifier.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GBA-4 `call()` Should be Used Instead of `transfer()` on An Address Payable

Severity: Minor

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#471

Descriptions:

The `transfer()` and `send()` functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction. https://swcregistry.io/docs/SWC-134

In the `claimMon()` function, the protocol utilizes the `transfer()` method to transfer the ETH to the target address.

```
    function claimMon(uint256 amount) external nonReentrant returns (bool) {
     require(amount > 0, "Amount must be greater than 0");
     //     1 Mon = 100 usdo
     uint256 monAmt = amount / 100; //
     //                    usdo
     require(IERC20(acceptedToken).balanceOf(msg.sender) >= amount, "Insufficient
Mon balance");
     //                       ETH
     require(address(this).balance >= monAmt, "Insufficient ETH in contract");
     //    mon
     IERC20(acceptedToken).safeTransferFrom(msg.sender, address(this), amount);
     // ETH
     payable(msg.sender).transfer(monAmt);
     emit ClaimToken(globalNonce, msg.sender, address(acceptedToken), monAmt,
```

```
amount);
        return true;
    }
```

The use of the deprecated `transfer()` function for an address will inevitably make the transaction fail when:

The claimer smart contract does not implement a payable function.

The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.

The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.

Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Suggestion:

It is recommended to use `call()` instead of `transfer()`, but be sure to respect the CEI pattern and/or add re-entrancy guards, as several hacks already happened in the past due to this recommendation not being fully understood.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GBA-5 Inconsistent `globalNonce` Increment Timing

Severity: Minor

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#369-388

Descriptions:

The `globalNonce` state variable is intended to provide a sequential, unique identifier for every state-changing event in the contract. However, its increment logic is inconsistent across different functions.

In functions like `sendGuessEvent`, `addCondition`, and `bet`, the `globalNonce` is incremented after the corresponding event is emitted. In contrast, the `settleCondition` function increments `globalNonce` before emitting its event.

This inconsistency can lead to ambiguity and errors for off-chain indexers or clients that rely on the event log. A client consuming these events expects a predictable pattern for ordering and state reconciliation. The current implementation breaks this predictability, as the nonce in the emitted event sometimes represents the state before the increment and sometimes after.

Suggestion:

Adopt a consistent pattern for updating and emitting the `globalNonce` across all relevant functions.

```
    function settleCondition(uint256 eventId, uint256 conditionId, uint8 result) external
nonReentrant onlyRole(EVENT_SETTLER_ROLE) {
        Condition storage c = events[eventId].conditions[conditionId];
        require(!c.settled, "Already settled");
        c.result = result;
        c.settled = true;
        if (uintToOption(c.result) == Option.Yes) {
```

```
        uint256 thisFee = ((c.totalPaid[Option.No] * platformFee) / 10000);
        c.totalReward = c.totalPaid[Option.No] - thisFee;
        c.thisTimeProtocolFee = thisFee;
    } else {
        uint256 thisFee = ((c.totalPaid[Option.Yes] * platformFee) / 10000);
        c.totalReward = c.totalPaid[Option.Yes] - thisFee;
        c.thisTimeProtocolFee = thisFee;
    }
    globalNonce++;
    //
    emit ConditionSettled(globalNonce, eventId, conditionId,
        result, c.totalReward, c.thisTimeProtocolFee,
        msg.sender);
}
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

GBA-6 `_GuessTokenBase_init` Parameter Is Misspelled

Severity: Informational

Status: Fixed

Code Location:

src/GuessBase/GuessBase.sol#166

Descriptions:

The parameter `_MultisiteWallet` of the initialization function `__GuessTokenBase_init` may have a spelling mistake. In blockchain development, the Wallet used for managing permissions is usually called a "Multi-signature wallet", and its common abbreviation is `_MultisigWallet`

Suggestion:

Correct the misspelled words

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.