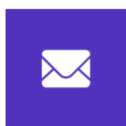
Suigar

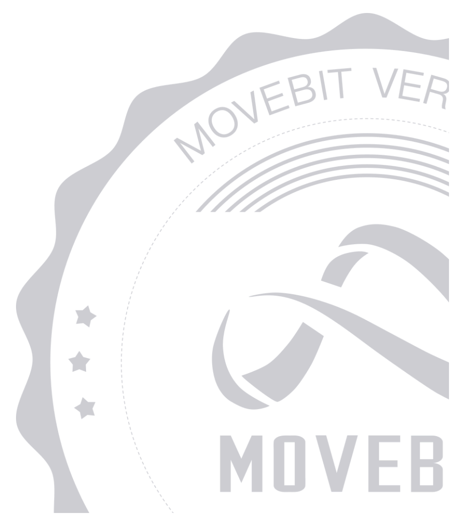
Audit Report



contact@bitslab.xyz

https://twitter.com/movebit_

Fri Oct 24 2025



Suigar Audit Report

1 Executive Summary

1.1 Project Information

Description	Suigar is a game contract.
Type	Game
Auditors	Bear Two, s3cunda
Timeline	Sat Oct 11 2025 - Fri Oct 24 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Suigar-Gaming/suigar-contracts
Commits	1dd47288671cfe1a61541fa62217902ba3bd07c6 41fe60f12fc2d95705d5712500339209f23f2823 500ce3dec9fd2c3816d3eeeaea0c3bce5283ed15

1.2 Files in Scope

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

ID	File	SHA-1 Hash
NFT	sources/nft.move	7ef9004b42a3fe6f24fbb123294581be2f06779c
HOU	sources/house.move	8a69053c85bea599357b2468cec41b242ae028eb
EVE	sources/events.move	e3f951046ed7084ca9fdbf513519e3e1ce086f60
MAT	sources/math.move	e50ff85f5244ef0620c02aa653668069b77d5d2c
LIM	sources/limbo.move	a1e6c1241e7e44db22740b464921cf49f988f3ce
PLI	sources/plinko.move	dbfd5626ed79e54024443ebe29d5306b5754e702
DIC	sources/dice.move	bec0e3928c63d053b1ae807fb8f083f1ba9107de
COI1	sources/coinflip.move	99b893ead051fe6459907a0cd783a7380273b247
GCH	sources/global_chat.move	2ce229208600a157208ea8bbce6275e0b49a77f2
LOO	sources/lootbox.move	3636c607b65ab3d6ad622a3a165551a9dee8f47c

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	13	10	3
Centralization	1	0	1
Critical	0	0	0
Major	1	1	0
Medium	4	4	0
Minor	1	1	0
Informational	5	4	1

1.4 MoveBit Audit Breakdown

MoveBit 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 by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

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) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) 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 Suigar to identify any potential issues and vulnerabilities in the source code of the Suigar 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 13 issues of varying severity, listed below.

ID	Title	Severity	Status
COI-1	Constant Name Misspelling `TotalPropability`	Informational	Fixed
COI-2	Context Parameter Mutability Issue	Informational	Acknowledged
DIC-1	A Guaranteed Win Vulnerability in `place_bet` Function Due to Incorrect Boundary Check on `bet_threshold`	Medium	Fixed
GCH-1	Timestamp Provided by User Allows Message Time Forgery	Informational	Fixed
HOU-1	Reversed Referral Reward Assignment	Medium	Fixed
LIM-1	Inconsistent Payout Calculation Logic Allows `max_payout` Bypass	Medium	Fixed
LIM-2	Missing Validity Check When Setting Game Config Parameters	Minor	Fixed

LIM-3	Unristricted Reveal Bet Function Invokation	Discussion	Acknowledged
LIM-4	Centralization Risk on `edit` Functionality Design	Centralizatio n	Acknowledged
LOO-1	Inconsistency Distributing Reward Logic in `lootbox`	Informational	Fixed
NFT-1	Missing Refund for Excess Payment in `mint_to_sender` Function	Medium	Fixed
NFT-2	Unsafe URL Creation Used in `suigar::nft::create_spec` Function	Informational	Fixed
PLI-1	Potential DoS Caused by Unlimited Bet Persistency	Major	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Suigar Smart Contract :
Admin

- create_house - Creates a new house
- withdraw - Withdraws tokens from a house
- withdraw_all - Withdraws all tokens from a house
- create_coinflip_game - Creates a coin flip game
- edit_coinflip_game - Edits coin flip game parameters
- create_dice_game - Creates a dice game
- edit_dice_game - Edits dice game parameters
- create_limbo_game - Creates a Limbo game
- edit_limbo_game - Edits Limbo game parameters
- create_plinko_game - Creates a Plinko game
- edit_plinko_game - Edits Plinko game parameters
- create_lootbox_game - Create a Lootbox Game
- edit_lootbox_game - Edit Lootbox Game Parameters

Player/Gambler

- deposit - Deposit tokens into the pool
- set_referer - Set a referrer relationship
- place_bet_and_reveal_onchain_randomness (all games) - Place a bet and reveal the result immediately
- mint_nft - Mint an NFT
- burn_nft - Burn an NFT

- transfer_nft - Transfer an NFT
- create_chat - Create a chatroom
- send_message - Send a message
- edit_message - Edit a message
- delete_message - Delete a message

Referrer

- claim_referrer_rewards - Claim rewards earned as a referrer

Referee

- claim_referee_rewards - Claim rewards earned as a referrer

4 Findings

COI-1 Constant Name Misspelling `TotalPropability`

Severity: Informational

Status: Fixed

Code Location:

sources/coinflip.move#20

Descriptions:

The constant `TotalPropability` contains a spelling error — the correct spelling should be `TotalProbability`.

Suggestion:

Rename the constant definition and all its references from `TotalPropability` to `TotalProbability` throughout the entire module.

Resolution:

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

COI-2 Context Parameter Mutability Issue

Severity: Informational

Status: Acknowledged

Code Location:

sources/coinflip.move#1

Descriptions:

The current implementation uses `&mut TxContext` as parameter type, but based on the usage pattern, it should accept `&TxContext` instead. Since the function only needs to read from the context (ctx) rather than modify it, using an immutable reference is more appropriate and follows Move's principle of using the least privileged access level.

Suggestion:

Change parameter type from `&mut TxContext` to `&TxContext` to allow immutable borrowing when only reading context data.

DIC-1 A Guaranteed Win Vulnerability in `place_bet` Function Due to Incorrect Boundary Check on `bet_threshold`

Severity: Medium

Status: Fixed

Code Location:

sources/dice.move

Descriptions:

In the `place_bet` function, the validation for `bet_threshold` is `assert!(bet_threshold <= MaxRange, EInvalidBetThreshold)`;. However, in the `reveal_bet` function, the random number is generated via `random::generate_u64_in_range(&mut generator, 0, MaxRange)`, which returns a value in the range [0, MaxRange - 1]—the upper bound is exclusive. Consequently, a player can place a bet by setting `bet_threshold = MaxRange` and choosing `roll_under = true`. During the reveal phase, the winning condition `rand <= bet_threshold` (i.e., `rand <= MaxRange`) will always evaluate to true, since the maximum possible value of rand is MaxRange - 1. This guarantees a win for the player every time.
Although the payout in this scenario is only 1:1, this predictability can be exploited to farm referral rewards or other betting-related incentives without any risk.

Suggestion:

The validation for bet_threshold should be modified to strictly less than MaxRange. Change the assertion from `assert!(bet_threshold <= MaxRange, EInvalidBetThreshold)`; to `assert!(bet_threshold < MaxRange`, EInvalidBetThreshold);.

Resolution:

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

GCH-1 Timestamp Provided by User Allows Message Time Forgery

Severity: Informational

Status: Fixed

Code Location:

sources/global_chat.move#83

Descriptions:

The `add_message` function accepts a `timestamp` parameter supplied by the caller and stores it as part of the message metadata. It does not rely on a trusted on-chain time source (such as SUI's Clock object) to determine the message time. This allows an attacker to submit messages with arbitrary timestamps — for example, pretending that a message was sent in the past or the future. Such behavior compromises the integrity and trustworthiness of the chat history, as the temporal context of messages becomes unreliable.

Suggestion:

Remove the `timestamp: u64` parameter from the `add_message` function signature. Instead, pass the `clock: &sui::clock::Clock` object as a parameter and obtain the trusted timestamp within the function using `sui::clock::timestamp_ms`(clock). This ensures that all message timestamps are accurate and cannot be forged.

Resolution:

The on-chain chat module was deprecated months ago and is no longer callable or referenced in production. All chat logic now runs off-chain.

HOU-1 Reversed Referral Reward Assignment

Severity: Medium

Status: Fixed

Code Location:

sources/house.move#183

Descriptions:

In `distribute_referral_rewards`, the function incorrectly assigns rewards to the wrong parties. The referee (player) receives the referrer's reward balance, and the referrer receives the referee's reward balance. This reverses the intended reward distribution logic.

Suggestion:

Swap the two `balance::join` calls:

```
balance::join<T0>(&mut referee_data.referee_balance, referee_rewards_balance);
balance::join<T0>(&mut referrer_data.referrer_balance, referrer_rewards_balance);
```

Resolution:

This feature has been disabled and is not active in the current system.

LIM-1 Inconsistent Payout Calculation Logic Allows `max_payout` Bypass

Severity: Medium

Status: Fixed

Code Location:

sources/limbo.move#125

Descriptions:

In the `place_bet` function, the `max_payout` check is based on `max_reward_per_bet`, which is calculated as follows:

`max_reward_per_bet = uq32_32::int_mul(amount_per_bet, target_multiplier)`

`amount_per_bet = amount / (number_of_bets as u64)`

Because of integer division, this operation truncates any remainder.

However, in the `reveal_bet` function, the actual payout calculation differs:

`payout_per_bet_if_win = balance::value(&fund) / (number_of_bets as u64)`

`fund = uq32_32::int_mul(amount, target_multiplier)`

Due to the difference in operation order:

In `place_bet`: `(amount / number_of_bets) * target_multiplier`

In `reveal_bet`: `(amount * target_multiplier) / number_of_bets`

The results of these two expressions can differ under integer division. In most cases, `(amount * target_multiplier) / number_of_bets` is greater than `(amount / number_of_bets) * target_multiplier`.

As a result, an attacker can craft a bet that passes the `max_payout` check in `place_bet`, but whose actual payout during `reveal_bet` exceeds the intended upper limit, causing the house to suffer unexpected losses.

Suggestion:

Ensure payout calculations are consistent across all functions.

Specifically, update the `place_bet` check to match the actual payout logic used in `reveal_bet`:

```
let potential_payout_per_bet = uq32_32::int_mul(amount, target_multiplier) /
(number_of_bets as u64);
assert!(potential_payout_per_bet <= limbo_game.max_payout, EInvalidBetAmount);
```

This change ensures consistent logic and prevents attackers from bypassing the `max_payout` restriction.

Resolution:

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

LIM-2 Missing Validity Check When Setting Game Config Parameters

Severity: Minor

Status: Fixed

Code Location:

sources/limbo.move;

sources/plinko.move;

sources/dice.move;

sources/coinflip.move;

sources/lootbox.move

Descriptions:

A critical oversight exists in the parameter configuration mechanism, where configuration inputs are not validated for correctness or logical consistency. This vulnerability exposes the protocol to significant risks: maliciously crafted values could lead to catastrophic financial loss, while inadvertently erroneous inputs could cause the protocol to enter a failed state, halting core functionality and compromising operational integrity. Many parameters need to be checked here. For example, the maximum `threshold` cannot be greater than the maximum value of the random number that can be generated. The `amount` and `price` must be greater than 0. The `referrer` address cannot be 0. The `referral fee ratio` cannot be too high. The minimum `bet` must be less than the maximum `bet`, etc.

Suggestion:

It is recommended to add explicit check on input parameters.

Resolution:

The team adopted our advice and fixed the issue by adding proper check on input parameters.

LIM-3 Unristricted Reveal Bet Function Invokation

Severity: Discussion

Status: Acknowledged

Code Location:

sources/limbo.move;

sources/plinko.move;

sources/dice.move;

sources/coinflip.move;

sources/lootbox.move

Descriptions:

In current games design, `reveal_bet` function is unristricted on context, which means users can reveal all bets regardless those bet were or not created by them.

Suggestion:

It is recommended to clarify this pattern is by design or not. If not so it is recommended to add ristriction on reveal function call.

Resolution:

The team claim that this behavior is intentional and part of decentralization design, anyone can reveal a bet to avoid stuck funds. In production, their backend reveals most bets automatically.

LIM-4 Centralization Risk on `edit` Functionality Design

Severity: Centralization

Status: Acknowledged

Code Location:

sources/limbo.move;

sources/plinko.move;

sources/dice.move;

sources/coinflip.move;

sources/lootbox.move

Descriptions:

In the current design of all games, an edit function is available to modify game parameters, including odds and probabilities. However, since players can choose to participate in a game and reveal the results at a later time, a gap period exists between participation and revelation. During this period, modifications to game settings are permitted, and the game data used when revealing results corresponds to the current settings rather than those in effect at the time of participation. Consequently, if administrators adjust the parameters during this interval, players may receive outcomes that differ from what they initially anticipated based on the settings at the time of their participation.

Suggestion:

We would like to clarify whether this design is intentional. If not, we recommend storing the game parameter settings at the time of participation within the user's game record.

Resolution:

The team claim that this situation no longer applies. All games on suigar.com exclusively use the bet_and_reveal flow, which combines betting and revealing in one atomic transaction. No user interacts with functions that allow a delayed reveal, making any potential edit gap irrelevant in production.

LOO-1 Inconsistency Distributing Reward Logic in `lootbox`

Severity: Informational

Status: Fixed

Code Location:

sources/lootbox.move

Descriptions:

In `lootbox` game reveal logic, the final reward distribution is as follow:

```
et reward_coin = coin::take(&mut fund, reward, ctx);
        house::join_balance(house, fund);

        transfer::public_transfer(reward_coin, sender);

        let payout_multiplier = uq32_32::from_quotient(reward, stake);
```

However, the reveal logic in other games follows a different distribution pattern:

```
        let balance_coin = coin::from_balance(fund, ctx);
        let balance_coin_mut = &mut balance_coin;
        let payout_coin: Coin<T0> = coin::split(balance_coin_mut, total_payout_amount, ctx);


        // transfer the payout coins to the player
        transfer::public_transfer(payout_coin, gambler_address);

        // add the rest into the bank
        house::deposit(house, balance_coin);
```

For `lootbox` it use coin::take() to extract rewards, then merges remaining funds.

Other games uses coin::split() to separate payouts, then deposits the remainder.

This inconsistency in fund distribution logic across game types may warrant standardization for maintainability.

Suggestion:

Standardize the fund distribution logic across all games by adopting the consistent coin::split() approach used in other games. This eliminates the current inconsistency where lootbox uses coin::take() while others use coin::split(), improving code maintainability and reducing potential edge cases in fund handling.

Resolution:

The team adopted our advice and fixed the issue by unify the reward distribution logic.

NFT-1 Missing Refund for Excess Payment in `mint_to_sender` Function

Severity: Medium

Status: Fixed

Code Location:

sources/nft.move#139

Descriptions:

In the current implementation of the `mint_to_sender function`, the contract checks that the user's payment is greater than or equal to the mint price:

`assert!(coin::value(&payment_coin) >= s.price, EInsufficientPayment);`

However, if the user sends more than the required price (`payment_coin > s.price`), the function deposits the entire payment into the house:

`house::deposit(house, payment_coin);`

No refund is provided for the excess `amount` beyond `s.price` .

Suggestion:

Modify the payment logic to refund any excess amount back to the sender.

Resolution:

All NFTs have already been fully minted, and the `mint_to_sender` function is no longer callable.

NFT-2 Unsafe URL Creation Used in `suigar::nft::create_spec` Function

Severity: Informational

Status: Fixed

Code Location:

sources/nft.move#40

Descriptions:

The `create_spec` function uses `url::new_unsafe_from_bytes`(url) to create a URL object from a byte vector. This function does not validate whether the input bytes represent a valid URL format, which could result in malformed or invalid data being stored in the url field of the Spec object. Such invalid data may affect the proper functioning of off-chain applications or front-end systems that rely on this information.

Suggestion:

It is recommended to use the safer `url::from_bytes(url)` function instead.

Resolution:

The function is unused in production and will be fully deprecated.

PLI-1 Potential DoS Caused by Unlimited Bet Persistency

Severity: Major

Status: Fixed

Code Location:

sources/plinko.move;

sources/dice.move;

sources/limbo.move;

sources/lootbox.move

Descriptions:

In the current code, all game categories follow a common logic: first, the necessary funds for participating in the game are collected from the user, and then, based on the maximum potential payout of the game, the corresponding maximum amount is locked from the vault.

```
let max_reward = uq32_32::int_mul(
        amount,
        uq32_32::mul(bet_multiplier, actual_rtp)
    );
    house::deposit(house, bet_coin);

    let bet = Bet {
        id: object::new(ctx),
        gambler: tx_context::sender(ctx),
        bet_threshold,
        roll_under,
        number_of_dices,
        total_stake_value: amount,
        fund: house::take_fund_balance<T0>(house, max_reward)
    };
```

In terms of code logic and game design theory, the maximum payout is always higher than the user's initial investment. These locked funds are only released when the user decides to

end the game and reveal the outcome. However, at the current design stage, there are no constraints on when users can start or end a game. In other words, users can start and end games arbitrarily at any time and place. Combined with the previous analysis, this could potentially lead to the depletion of funds in the contract.

Consider the following scenario: a malicious user repeatedly places bets in games with the highest multipliers, exhausting the contract's funds, and then chooses not to end the games, causing the funds to remain locked indefinitely. As a result, other users are unable to start new games, creating a denial-of-service effect.

Suggestion:

It is recommended to implement a time limit that forces users to reveal the outcome within a specified period; otherwise, their funds will be forfeited.

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.