HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Sallar
Date: 23 August, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Sallar
Approved By	Oleksii Zaiats | SC Audits Head at Hacken OÜ
Type	GameFi, Fungible Token
Platform	Solana
Language	Rust
Methodology	Link
Website	sallar.io
Changelog	25.05.2023 - Initial Review 12.07.2023 - Second Review 31.07.2023 - Third Review 23.08.2023 - Fourth Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

HACKEN

Table of Contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Sallar (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

This program manages creation, emission and distribution of a fungible token. Its maximum supply is 54,600,000,000 of which:

- 2,600,000,000 is distributed to a special “organization” account.
- 52,000,000,000 is distributed in batches of so-called “blocks”.

There are 2,600,000 blocks, which are distributed in a game-like fashion one by one in two ways: top-to-bottom (starting from block 1, aka “mining”), and bottom-to-top (starting from block 2,600,000, aka “staking”). Each block needs to be exhausted (for the next one to become available) by user requests, which the off-chain code posts to the contract on behalf of the users; each such request causes transferring a portion of the block tokens to the user that corresponds to the request. The rules and requests are different for the two directions - the details are in the official documentation.

Once all the blocks have been consumed, two distribution modes become available: “final mining” and “final staking”. Each mode has a designated account (exclusively managed by the contract) which can be funded by anybody using the previously distributed funds; funds in the special accounts are distributed according to yet new games, whose rules are described in the official documentation. The two final distribution games are played in cycles, and there can be indefinitely many cycles - as long as there are funds in the special accounts. Like in the normal block distribution modes, the games are played with user requests, which the off-chain code posts to the contract on behalf of the users; each such request causes transferring a portion of the block tokens to the user that corresponds to the request.

Each distribution game has a significant part of its logic outside the contract - **which is out of the scope**.

The smart contract has the following API:

- *initialize* — Sets up program-related accounts and performs initial mint. Can be called at most once.
- *initial_token_distribution* — Distribute 2,600,000,000 tokens to the “organization” account, provided by the owner. Can be called at most once.

- *solve_top_block* – The owner executes a set of user requests to play the “mining” game.
- *solve_bottom_block* – The owner executes a set of user requests to play the “staking” game.
- *final_mining* – The owner executes a set of user requests to play the “final mining” game.
- *final_staking* – The owner executes a set of user requests to play the “final staking” game.
- *change_authority* – The owner changes the contract authority to the new one.

Privileged roles

The only user of the program is *owner* - an account registered by the *initialize()* function and can be changed using *change_authority()*.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation Quality

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

- The functional documentation covers everything in satisfactory detail.
- Technical description is provided.

Code Quality

The total Code Quality score is **7** out of **10**.

- Redundancies are found (I02.2).
- Some confusing points are found (I06).
- Floating point numbers math is used.

Test Coverage

Code coverage of the project is about **90%**.

- *final_mining* and *final_staking* instructions are tested but their effects are not checked during testing.
- The *solve_top_block* and *solve_bottom_block* instructions that take a lot lines of code (due to redundancies and code duplications) are successfully tested.

Security Score

As a result of the audit, the code does not contain security issues. The security score is **10** out of **10**.

All found issues are displayed in the Findings section of the report.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Summary

According to the assessment, the Customer's smart contract has the following score: **9.1**.

The system users should acknowledge all the risks summed up in the Risks section of the report.

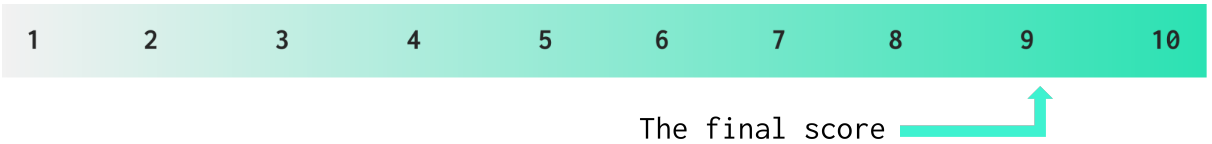

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 May 2023	0	4	2	1
12 July 2023	1	2	0	0
31 July 2023	1	0	0	0
23 August 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Risks

- Unless the smart contract is deployed with the `--final` flag, it could be upgraded and its functionality may be changed.
- The contract is **fully centralized**:
  - The `owner` is the only user and has complete control over it.
  - The token distribution claims are made exclusively by the `owner`, who is free to arbitrarily forge them in their favor or censor/handicap normal user claims.
- A major part of the overall system logic - that is financially relevant - is outside the contract. The contract only processes results of some off-chain computations that are **out of the audit scope**.
- The `final_mining` and `final_staking` APIs are designed to distribute tokens from the special contract accounts, but it is not guaranteed that the accounts will be properly funded.
- Anyone is able to initialize the program by calling `initialize` (can be called only once). It is recommended to perform deployment and initialization in one transaction.
- The `convert_f64_to_u64` and `convert_u64_to_f64` will lose precision for numbers over 2^53, however it is unlikely.
- The program performs computations in `f64`, whereas the token value type is `u64` - which requires casting token values to `f64` and back. `u64` values that take more than 53 bits are likely to be not representable in `f64`, and have to be approximated. Moreover, `f64` computations have limited precision on their own. Together, this may result in losses of portions of token values. However, the collective losses due to `f64` precision are highly unlikely to be significant in USD equivalent.
- The deployment code should not be compiled with the `bpf-tests` feature.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Unchecked Errors	If a function returns a Result, it should not be ignored.	Passed
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed
Signature Reuse	Signed messages that represent an approval of an action should not be reusable.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Data or be predictable.	Not Relevant
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract.	Passed
Compiler Warnings	The code should not force the compiler to throw warnings.	Passed
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. The usage of contracts by multiple users should be tested.	Passed
Stable Imports	The code should not reference draft contracts, that may be changed in the future.	Not Relevant
Unsafe Rust code	The Rust type system does not check the memory safety of unsafe Rust code. Thus, if a smart contract contains any unsafe Rust code, it may still suffer from memory corruptions such as buffer overflows, use after frees, uninitialized memory, etc.	Passed
Missing rent exemption checks	A Solana account must contain enough SOL to be rent exempt. If an account is not rent exempt, it may be purged unexpectedly.	Passed
Unset or unsettable SPL-token metadata	If a contract defines an SPL-token, it should ensure that the token metadata is set or can be set later. If that is not the case, it would be impossible to properly integrate with blockchain explorers, exchanges, etc.	Passed
Too recent Solana libraries used	Due to Solana release conventions, there may be several latest standard library crate versions that are not ready for mainnet.	Passed

Findings

■■■■ Critical

C01. Incorrect Value

Impact	High
Likelihood	High

- ./programs/solana-contracts/src/lib.rs
  - *DUSTS_PER_BLOCK*: the documentation states it must be 20000 Sallar, but it is actually 20 Sallar.
  - *initial_token_distribution*: the documentation states it must distribute 2,600,000,000 Sallar, but it actually distributes 2,600,000 Sallar.
  - final_mining
    - The literal *25_000_000* represents 0.025 Sallar, not 25 Sallar like it is stated in the documentation.
    - The literal *50_000_000* represents 0.05 Sallar, not 50 Sallar like it is stated in the documentation.
    - The literal *100_000_000* represents 0.1 Sallar, not 100 Sallar like it is stated in the documentation.
    - The literal *250_000_000* represents 0.25 Sallar, not 250 Sallar like it is stated in the documentation.
    - The literal *500_000_000* represents 0.5 Sallar, not 500 Sallar like it is stated in the documentation.
- ./programs/solana-contracts/src/token_math.rs
  - *FIRST_BP*: the documentation states it must be 20,000, but it is actually 20.
  - *calculate_top_bp*: according to the documentation, the result of the function must be multiplied by 1000.
  - *TOP_BOOST_REDUCTION*: according to the documentation, the value must be *1.000004498927*, whereas actually, it is *1.0000044986146084*.

Path: ./programs/solana-contracts/src/

Recommendation: Fix the incorrect values according to the individual item descriptions.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

■■■ High

H01. Unset Or Unsettable Token Metadata

Impact	Medium
Likelihood	High

The SPL-token mint account of Sallar token is a PDA of the program. The mint authority is also a PDA of the program, which guarantees that the token management functions can only be performed by the program code.

The program code does not set the token metadata nor provides an API to set it later. This denies the presence of token metadata, which is crucial for integration with many important on and off-chain platforms like blockchain explorers, exchanges, etc.

Path: ./programs/solana-contracts/src/

Recommendation: Set the token metadata in the program during the initialization. Alternatively, provide an admin-only API to set the metadata.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

H02. Stuck Funds

Impact	High
Likelihood	Medium

The expression at line 236 determines the fraction of the final staking account balance that will be distributed in a single final staking cycle; precisely, it is 0.1% of the balance.

Note:

- The interval between the final staking iterations is at least approximately 20 hours.
- There is no other way to extract the balance of the final staking account except this process.

For any given balance of the final staking account, after, for example, 1200 consecutive final staking cycles there will be left

*100 * (1 - 0.001)^1200 = 30 (approximately)*

percent of the initial balance in the account; 1200 cycles run perfectly back-to-back would take approximately 1000 days; therefore, the example shows that it would take about 1000 days (at a minimum) to extract 70% of the value. To extract 99.5%, it would take approximately 10,000 days at a minimum.

Note that the severity of the issue depends on how much funds are in the final staking account. Due to the design of the program, it is not possible to estimate the amount beforehand.

Path: ./programs/solana-contracts/src/lib.rs: final_staking

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Recommendation: Consider a process that distributes the total funds faster and/or devise a mechanism to rescue the remaining funds in a fair way.

Found in: 122d017

Status: Mitigated (Distribution of funds via final_staking instruction takes much time intentionally)

Medium

M01. Invalid Hardcoded Value

Impact	Low
Likelihood	High

The value literal of constant *REDUCTION_INVERSE* is not directly representable by *f64* in Rust. Rust will round the literal silently.

Additionally, according to the documentation, the value must be *0.99999430521433*, whereas actually, it is *0.9999943052143271*.

Path: ./programs/solana-contracts/src/token_math.rs

Recommendation: Correct the constant according to the documentation.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

M02. Immutable Ownership

Impact	High
Likelihood	Low

The contract is designed in a way that ownership cannot be transferred.

This may lead to the impossibility to update the owner in critical situations.

Path: ./programs/solana-contracts/src/lib.rs

Recommendation: Implement an ability to transfer contract ownership.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

M03. Missing Validation

Impact	High
Likelihood	Low

The arguments *mint_nonce*, *top_block_nonce*, *bottom_block_nonce*, *final_staking_account_nonce*, *final_mining_account_nonce* represent the bump seeds of some PDAs that the instruction operates upon. However, arbitrary values can be passed for them, not necessarily the correct bump seeds. If that is the case, the contract state will be corrupted without possibility to fix, and subsequent instruction calls could not work properly.

Path: ./programs/solana-contracts/src/lib.rs:initialize

Recommendation: Check that the values are correct bump seeds. Alternatively, do not pass them as arguments, and derive them in the function instead.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

M04. Denial Of Service State

Impact	Medium
Likelihood	Medium

The code does calculations with the limited precision of *f64* type. There are cases when the results of such calculations are directly compared to exact values in order to decide whether to allow some process to move forward - without regard to the fact that the comparison may not succeed due to the precision errors.

This may result in state transition failures on seemingly valid inputs.

In all the cases, it is possible to find such inputs that would circumvent a failure. However, such inputs may be artificial and require breaking some higher-level rules to produce. For example, it could be needed to provide a set of user mining/staking requests with right parameters to exhaust a block and switch to the next one, but those requests could be not available naturally (based on the off-chain users' actions and their states), so the program owner would have to **manufacture fake/dummy requests** for this purpose. Nevertheless, the **off-chain system is out of scope**; therefore, it is impossible to precisely reason about the impact of these issues and ways to work around them.

Cases:

1. ./programs/sallar/src/utils.rs:(296, 347): there can be cases when the remaining block balance is 0, while the remaining BP is not 0, which would violate the assert.

   Status: Mitigated (The case is possible only in case of small block parts processing at the block end and could be easily fixed by the off-chain system reordering user requests)

2. ./programs/solana-contracts/src/lib.rs:(111, 166): at some program states, for some values of *blocks_state.top_block_available_bp* it is not possible to compute *current_user_total_bp* that will be exactly equal. Multiple separate user requests would be needed to exhaust such *blocks_state.top_block_available_bp*.

   Status: Fixed (Revised commit: 2fc3132)

3. ./programs/solana-contracts/src/lib.rs:267: it may not always be possible to naturally provide individual user shares that add up to exactly *1.0*.

   Status: Mitigated (The logic is a design decision)

Path:

- ./programs/solana-contracts/src/lib.rs
- ./programs/solana-contracts/src/utils.rs

Recommendation: Account for the possibility of precision errors in the mentioned cases. One general approach is to allow an actual value to slightly exceed an expected value for the equality check to succeed.

Found in: 122d017

Status: Refer to respective case status.

M05. Missing Validation & Corrupted Data

Impact	High
Likelihood	Low

Functions performing conversions between integers and floating point numbers are expected to perform conversions safely; however, they can still result in corrupted values due to rounding errors.

In most cases, *convert_u64_to_f64_safely(x)* will have a rounding error for *x > 2^53* (~90,000,000 ALL). For example, *2^58 - 5* will be rounded to *2^58*.

Both the functions have redundant checks that could be removed.

Path: programs/sallar/src/utils.rs

- convert_u64_to_f64_safely
- convert_f64_to_u64_safely

Recommendation: Update the checks to make sure the numbers can be represented.

An `f64` number can be accurately converted into `u64` if: `x < u64::MAX as f64 && x >= u64::MIN as f64`, where `x: f64`. The upper bound may be set to `x <= (1u64 << 53)` to manage possible `f64` precision error.

There are two ways to make sure that a `u64` number can be accurately converted into `f64`:

1. `x != u64::MAX && x == x_f as u64`, where `x: u64`, `x_f = x as f64`;
2. `x <= (1u64 << 53)`, where `x: u64`;

Found in: 122d017

Status: Mitigated (This code is used in a way that will not result in corrupted values or unrecoverable program state)

Low

L01. Inconsistent Data

Impact	Low
Likelihood	Medium

The `FINAL_STAKING_ACCOUNT_BALANCE_PART_FOR_STAKING_DIVISION_FACTOR` was used as `f64` in the previous program version. It was changed into `u64` to omit unnecessary conversions, but they themselves were not removed.

Path: ./programs/sallar/src/lib.rs: final_staking

Recommendation: Remove conversion to/from float.

Found in: d246c9d

Status: Fixed (Revised commit: 787caf3)

Informational

I01. Missing Validation

The code performs conversions from `f64` to `u64` using `as` keyword. This conversion does a saturating cast: if an `f64` is less than zero, the result will be zero; if an `f64` is greater than `u64::MAX`, the result will be `u64::MAX`.

This may lead to a significant silent loss of value. However, this will actually never happen: either the code does not produce such

large values or such cases would be ruled out by accompanying assertions.

Path:

- ./programs/solana-contracts/src/lib.rs
  - initialize
  - final_staking
- ./programs/solana-contracts/src/token_math.rs
  - calculate_top_block_max_boost
  - calculate_bottom_bp
  - calculate_user_reward
- ./programs/solana-contracts/src/utils.rs
  - switch_top_block_to_next_one_if_applicable
  - switch_bottom_block_to_next_one_if_applicable

Recommendation: To be sure that a silent/unintended saturation never happens, explicitly validate that an *f64* is between *0* and *u64::MAX* before casting to *u64*.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

I02.1. Redundant Code

- Returning the result of a *let* binding from a block.

  Path:

  - ./programs/solana-contracts/src/token_math.rs:36:5

  Recommendation: Omit the temporary variable.

- Explicit bool comparison.

  Path:

  - ./programs/solana-contracts/src/utils.rs:77:14
  - ./programs/solana-contracts/src/utils.rs:111:9

  Recommendation: Remove comparison.

- Excessive check for the impossible condition. Value cannot be less than zero.

  Path:

  - ./programs/solana-contracts/src/utils.rs:95:14
  - ./programs/solana-contracts/src/utils.rs:96:14

  Recommendation: Perform comparison to zero.

- Operation and assignment are not combined into one operation.

  Path:

- ./programs/solana-contracts/src/utils.rs:136:9
- ./programs/solana-contracts/src/utils.rs:158:9
- ./programs/solana-contracts/src/lib.rs:112:17
- ./programs/solana-contracts/src/lib.rs:130:13
- ./programs/solana-contracts/src/lib.rs:167:21
- ./programs/solana-contracts/src/lib.rs:185:17
- ./programs/solana-contracts/src/lib.rs:289:17

Recommendation: Reduce operations to a single `-=` or `+=` depending on the context.

- Redundant explicit passing of *Rent* sysvar account.

  Path:

  - ./programs/solana-contracts/src/context.rs:
    - InitializeContext::rent
  - ./programs/solana-contracts/src/utils.rs:
    - Valid_sysvar_address

  Recommendation: Use *Rent::get* to read the sysvar, because for sysvars, Solana does not require declaring the account at the call site.

- Duplicate constant.

  Path:

  - ./programs/solana-contracts/src/lib.rs
    - DUSTS_PER_BLOCK
  - ./programs/solana-contracts/src/token_math.rs
    - DUST_PER_BLOCK

  Recommendation: Deduplicate the code.

- Redundant handwritten struct size calculation.

  Path:

  - ./programs/solana-contracts/src/context.rs:
    - InitializeContext::BLOCKS_STATE_LENGTH

  Recommendation: The struct *BlocksState* already implements *InitSpace* which gives it *INIT_SPACE* associated constant. Import *anchor_lang::Space* trait in *context.rs* to be able to use the constant: *space = 8 + BlocksState::INIT_SPACE*;

Recommendation: Follow recommendations in the individual item descriptions.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

I02.2. Redundant Code

- Unnecessary indirection with `&Box<T>`.

  Path:

  - ./programs/sallar/src/utils.rs

  Recommendation: Consider using just `&T`.

- Indirect check for non-empty collection - `collection.len() > 0`.

  Path:

  - ./programs/sallar/src/lib.rs

  Recommendation: Use `!collection.is_empty()`.

- Redundant checks for signature as `Signer` structure performs it automatically.

  Path:

  - ./programs/sallar/src/utils.rs:
    - valid_signer
  - ./programs/sallar/src/context.rs: InitialTokenDistributionContext, SolveTopBlockContext, SolveBottomBlockContext, FinalStakingContext, FinalMiningContext, ChangeAuthorityContext

  Recommendation: Use `AccountInfo` structure and utilize either `valid_signer` function or `#[account(constraint… @ <custom_error>)]` attribute constraint.

- Redundant owner checks.

  Path:

  - ./programs/sallar/src/lib.rs:
    - initial_token_distribution
    - solve_top_block
    - solve_bottom_block
    - final_mining
    - final_staking
  - ./programs/sallar/src/utils.rs:
    - valid_owner

  Recommendation: Remove the checks, because they are already implemented in the respective instruction account structs in `context.rs` via Anchor's `constraint` attribute. Add a custom error to it using `#[account(<constraint> @ <custom_error>)]`.

- The same code pattern is repeated for the top and the bottom block.

Path:

- ./programs/sallar/src/lib.rs
  - initialize

Recommendation: Do not repeat significant code patterns in the code.

- The functions have almost the same code pattern, which is duplicated.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_top_block
    - solve_bottom_block

  Recommendation: Do not repeat significant code patterns in the code.

- The functions have almost the same code pattern, which is duplicated.

  Path:

  - ./programs/sallar/src/utils.rs
    - switch_top_block_to_next_one_if_applicable
    - switch_bottom_block_to_next_one_if_applicable

  Recommendation: Do not repeat significant code patterns in the code.

- The code pattern for matching accounts between `ctx.remaining_accounts` and `users_info` in the loop is duplicated.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_top_block
    - solve_bottom_block
    - final_mining
    - final_staking

  Recommendation: Do not repeat significant code patterns in the code.

- The hardcoded value `2_600_000_u64` is a duplicate of the constant `MAX_BLOCK_INDEX` defined in `token_math.rs`.

  Path:

  - ./programs/sallar/src/lib.rs
    - initialize

Recommendation: Use the constant.

- The value of the constant *MIN_REQUIRED_STAKE_FOR_BOTTOM_BLOCK_DUST* is the same as *DUST_PER_BLOCK*, yet its value is written as literal, which duplicates the other literal.

  Path:

  - ./programs/sallar/src/token_math.rs

  Recommendation: Explicitly set *MIN_REQUIRED_STAKE_FOR_BOTTOM_BLOCK_DUST* equal to *DUST_PER_BLOCK*.

- The hardcoded values duplicate the information that *decimals* of Sallar token is *8*, which is given in the definition of *InitializeContext::mint*.

  Path:

  - ./programs/sallar/src/token_math.rs
    - DUSTS_PER_BLOCK
    - MIN_REQUIRED_STAKE_FOR_BOTTOM_BLOCK_DUST
    - dust_to_staking_sallar
      - the literal *100_000_000*
  - ./programs/sallar/src/lib.rs
    - DUSTS_PER_BLOCK
    - initial_token_distribution
      - the literal *260_000_000_000_000_u64*
    - final_mining
      - the literal *2_500_000_000*
      - the literal *5_000_000_000*
      - the literal *10_000_000_000*
      - the literal *25_000_000_000*
      - the literal *50_000_000_000*

  Recommendation: Extract the mint decimals into a constant. Compute the mentioned values based on the mint decimals constant.

- Redundant outer scope declaration and mutability of local variables.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_bottom_block
      - current_user_total_bp
      - current_user_transfer_amount

  Recommendation: Declare the variables where they are assigned.

- The *mint* and *authority* parameters are actually always the same at the call site; therefore, the function should have only one of them instead of the two.

  Path:

  - ./programs/sallar/src/utils.rs
    - mint_tokens

  Recommendation: Replace the two parameters with one.

- The code processes a list that is known to have exactly 1 item as if it may have multiple items.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_bottom_block
      - user_find_result
    - final_mining
      - user_find_result
    - final_staking
      - user_find_result

  Recommendation: Explicitly assert-extract the single item from the list, process it without the loop.

- The code pattern for matching accounts between *ctx.remaining_accounts* and *users_info* in the loop is inefficient and produces some error-checking boilerplate to extract the single found account in *users_info*. The inefficiency is because for every account in *ctx.remaining_accounts*, a O(n) search through *users_info* is done.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_top_block
    - solve_bottom_block
    - final_mining
    - final_staking

  Recommendation: Before calling the functions, sort *users_info* in a way that an item that corresponds to account *A* has the same position as *A* in *ctx.remaining_accounts*. Then, use *Iterator::zip* (or similar, e.g. an indexed for loop that takes a pair of items from the two lists by the same index on each iteration) to loop through the pairs of items belonging to *ctx.remaining_accounts* and *users_info* respectively. Note that this approach also allows to eliminate *user_public_key* field from each account info struct, because it would be already given by the item in *ctx.remaining_accounts*. Also, note that

this approach produces less error-checking boilerplate; only the lengths of the two lists should be checked.

- The functions use the *transfer_tokens* function to transfer the token (aka dusts) from a designated distribution address (e.g. *distribution_top_block_address*) to the users. The distribution addresses (PDA) are funded by doing *mint_tokens* to them each time a new block is started. This approach is equivalent to doing *mint_tokens* directly to the users without having the intermediate distribution addresses. Note that the distribution addresses serve no other purpose than just being an intermediate account in the chain of transfers.

  Path:

  - ./programs/sallar/src/lib.rs
    - solve_top_block
    - solve_bottom_block

  Recommendation: Mint tokens directly to the users, eliminate the distribution addresses. This would make the code simpler and more efficient.

- Redundant checks are performed in the function at line 402: *user_sub_info.reward_part <= 1.0* is covered by the check at line 410.

  Path: ./programs/sallar/src/lib.rs:final_staking

  Recommendation: Remove the checks.

- Large logic duplication: a piece of code that involves *has_unprocessed_rest_from_last_block* is used twice in different functions.

  Path: ./programs/sallar/src/lib.rs:solve_top_block, solve_bottom_block

  Recommendation: Extract code into a separate function.

Recommendation: Follow recommendations in the individual item descriptions.

Found in: 787caf3

Status: Reported

I03. Floating Language Version

It is preferable for a production project, especially a smart contract, to have the programming language version pinned explicitly. This results in a stable build output, and guards against unexpected toolchain differences or bugs present in older versions, which could be used to build the project.

The language version could be pinned in automation/CI scripts, as well as proclaimed in README or other kinds of developer documentation. However, in the Rust ecosystem, it can be achieved more ergonomically via a *rust-toolchain.toml* descriptor (see https://rust-lang.github.io/rustup/overrides.html#the-toolchain-file)

Path: ./rust-toolchain.toml

Recommendation: Pin the language version at the project level.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

I04. Unformatted Code

cargo fmt yields changes in some files.

Path:

- ./programs/sallar/src/context.rs
- ./programs/sallar/src/lib.rs
- ./programs/sallar/src/token_math.rs
- ./programs/sallar/src/utils.rs

Recommendation: Format the code using *rustfmt* or equivalent.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

I05. Confusing Code

- *valid_sysvar_address* specifically checks the Rent sysvar address, yet it is named as if it supports any sysvar.

  Path: ./programs/solana-contracts/src/utils.rs

  Recommendation: Rename it to *valid_rent_address*.

- *MyError* enum name does not correspond to its purpose.

  Path: ./programs/solana-contracts/src/error.rs

  Recommendation: Rename it to reflect its purpose.

Recommendation: Follow recommendations in the individual item descriptions.

Found in: 122d017

Status: Fixed (Revised commit: 2fc3132)

I06. Confusing Code

TOKEN_AMOUNT_SCALING_FACTOR is not in the description and is a patch code which harms the readability.

It is hard to validate if all necessary values are multiplied by the constant.

Path: ./programs/sallar/src/token_math.rs

Recommendation: Remove it and adjust other constants that rely on it.

Found in: 787caf3

Status: Reported

I07. Unused Code

`U32ConversionError` and `FinalMiningAheadOfTime` error variants are not used.

Path: ./programs/sallar/src/error.rs

Recommendation: Remove unused code.

Found in: 2fc3132

Status: Fixed (Revised commit: d246c9d)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial Review Scope

Repository	https://github.com/Sallar-Foundation/sallar
Commit	122d0172748ab2489c08d6993d1def9c2318c0b7
Requirements	Link
Contracts	File: ./programs/solana-contracts/src/account.rs SHA3: 3bdb7bb7fddcb367ee0fbb26abfd9dfe2415c947cdd7e4f6bb94a34c2abbdfa3 File: ./programs/solana-contracts/src/context.rs SHA3: de99aa435a6f664380a53731d6c465d398ef39c92207e90521cd5c92de8dcb83 File: ./programs/solana-contracts/src/error.rs SHA3: c9c8c7c5d42cc8c9e63e59d905c5bcfabb0d240aa727c0b496d87dc0d88d8920 File: ./programs/solana-contracts/src/lib.rs SHA3: 50a618f55ced91a572085270dcdb9d9511ab7065bedcf264e238e2b76e9d369a File: ./programs/solana-contracts/src/token_math.rs SHA3: 82fd2f62182e41ad5961711e9a6a908f3b94dcbf4b9974fda199065d66b5f121 File: ./programs/solana-contracts/src/utils.rs SHA3: b30acf3e5e26bfaed1685d06e8034359a63298be5e184c3460f5ff28a7ec5730

Second Review Scope

Repository	https://github.com/Sallar-Foundation/sallar
Commit	2fc31324a370fd33f3e2a745302bc74f423a04ce
Requirements	Link
Technical Description	Concept Link
Contracts	File: ./programs/sallar/src/account.rs SHA3: a674880334bd68bc764104663c485ed091caf9472cc6b95fe7d3d53986cb5819 File: ./programs/sallar/src/context.rs SHA3: 63bd434842e12f117c43c7344ae28e6d8c9b8a59ca1581355d30c9540d256020 File: ./programs/sallar/src/error.rs SHA3: 163394fd73528f5c23f77e30ceaab4ae473c2c1228c404757b35eeae09837dd1 File: ./programs/sallar/src/lib.rs SHA3: ed761f749673e4760d4f870eb14ba420534f7322e592c56d0ebca8be28bd9b4b File: ./programs/sallar/src/token_math.rs SHA3: 0e435e3360423e0b6f04838ec2e5dced24aeb612ff28da0876bec4b9c2a5c04a File: ./programs/sallar/src/utils.rs SHA3: c1b2a2904b6759149c1a2ade0017b38486a7825611bf4e5e69a64c7955151463

Third Review Scope

Repository	https://github.com/Sallar-Foundation/sallar
Commit	d246c9de359597460aa689cdb41c6ee7324dc2f1
Requirements	Link
Technical Description	Concept Link
Contracts	File: ./programs/sallar/src/account.rs SHA3: d1c489b8a007ab90826ace68e59e79bb79c66ae1d85a83186e5af538983cb8e5 File: ./programs/sallar/src/context.rs SHA3: 6940f05bc37cd22528d7f69c8029e1e67abc3b038b1b6b6855bf499957af910e File: ./programs/sallar/src/error.rs SHA3: 6d0e1facb42004d91685d02c1f3546e7ede8080ac243e1e4daebe4c13507a710 File: ./programs/sallar/src/lib.rs SHA3: f25d568577942ea023d84b1d8fe5cbf1d576415e0cd4beb818a4b94e0af00469 File: ./programs/sallar/src/token_math.rs SHA3: bf8b3a512da5a34174720d7aa064c7848df5ca221056cfacfce9845190e6a448 File: ./programs/sallar/src/utils.rs SHA3: cbdf6f3e2565a3eac1bead8f3c96784b4d030ddfeec251ad7aea1f7b187e7a60

Fourth Review Scope

Repository	https://github.com/Sallar-Foundation/sallar
Commit	787caf381e11f75151a34dac0bed5f3cce8bb8d5
Requirements	Link
Technical Description	Concept Link
Contracts	File: programs/sallar/src/account.rs SHA3: d1c489b8a007ab90826ace68e59e79bb79c66ae1d85a83186e5af538983cb8e5 File: programs/sallar/src/context.rs SHA3: b99a2aeb066ec2408382b218a060f44c2d513f5862e5780ef546cd2ef749e6ca File: programs/sallar/src/error.rs SHA3: 6a86ce8946636c9816dbdddb952ec1a4bba4ab7dd2484161d7a1b1289b29fd31 File: programs/sallar/src/lib.rs SHA3: a237cab9b8b6a6999b50745a860c5cf63be8b24dd7524df220793a63b8700e28 File: programs/sallar/src/token_math.rs SHA3: 428ccee58c1dc3b6b95b626755be9e5ca832696922715a74daa7e1cc799be140 File: programs/sallar/src/utils.rs SHA3: e08338565f4015078207cc1df0095abbf3a5a089fe3f39c48cd4f9844e98e519