HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Leancoin
Date: May 29, 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 Leancoin
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OÜ
Type	Fungible token; Vesting; Migration
Platform	Solana
Language	Rust
Methodology	Link
Website	https://leancoin.io/
Changelog	31.03.2023 - Initial Review 25.04.2023 - Second Review 22.05.2023 - Third Review 29.05.2023 - Fourth Review

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

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 Leancoin (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.

Scope

The scope of the project includes review and security analysis of the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/Leancoin/Leancoin/
Commit	7a155aac8da784746962499de1846390b91ab3fb
Whitepaper	https://docs.leancoin.io/leancoin-white-paper/
Functional Requirements	https://docs.leancoin.io/swap-lean/
Technical Requirements	./README.md
Contracts	File: ./programs/LeanManagementToken/src/account.rs SHA3: 74d4fac5d659d5216af687cb9e47d40cb00bbb6f2b0da743ff7b8f8b1a6d8361 File: ./programs/LeanManagementToken/src/context.rs SHA3: c470d636727c093941479005e6fb13471e3b03194e7b7409e31ae50deaa790c1 File: ./programs/LeanManagementToken/src/error.rs SHA3: 89e7875d373ae70f1eed8620f7ff0296a1344371c72fd22343fec1a38440a621 File: ./programs/LeanManagementToken/src/lib.rs SHA3: 8d6d6c153f43c29959719e08ee8cb2185f4029d0a7738aa54a4fbde29bd31473 File: ./programs/LeanManagementToken/src/utils.rs SHA3: 9516bdcd6b555c3474a595dcb4ee7a76ad0cd8fe1d0765df4f9cdbf53e64e83b

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

Second review scope

Repository	https://github.com/Leancoin/Leancoin/
Commit	0e060f54bfd7dacbd01d802a72824bd980b1d346
Whitepaper	https://docs.leancoin.io/leancoin-white-paper/
Functional Requirements	https://docs.leancoin.io/swap-lean/
Technical Requirements	./README.md
Contracts	File: ./programs/LeanManagementToken/src/account.rs SHA3: 87cd8efd48b9738b990f4e76b715648599e4a3861ada23624925bc223e3ffb31 File: ./programs/LeanManagementToken/src/context.rs SHA3: 2c921a5b3b7f69466819ab01e47cfc22ed8883f8cab2630802a637c26d0c2649 File: ./programs/LeanManagementToken/src/error_codes.rs SHA3: 79f71282fb242d001dcfbad2c072bd7674d08db46048231208411be8a7921e4e File: ./programs/LeanManagementToken/src/lib.rs SHA3: 587f8a2049543fd2a3d6282ccbe2676044c28aa2fe92b978b715fcf16e94aa9c File: ./programs/LeanManagementToken/src/utils.rs SHA3: ca901abb88aca20d3371851a7b1171dc6643157657eb358d0b2a85c3f98f3aea

Third review scope

Repository	https://github.com/Leancoin/Leancoin/
Commit	52746b8fee5780e38c93f3ee5b202049cf4e5666
Whitepaper	https://docs.leancoin.io/leancoin-white-paper/
Functional Requirements	https://docs.leancoin.io/swap-lean/
Technical Requirements	./README.md
Program Id	CeFVa5iijJASnRmMCvrHep8wVYRZ3XxAmgXArNJhpjmx
SPL-token mint address	7297kX7SEZ1do223VsjTAC2MS9gLxPJoxFs9UMwiG4oS
Contracts	File: ./programs/LeanManagementToken/src/account.rs SHA3: bda0484adf3ca234d3b668d82ca381d2464207f314bf05018a7bdf7b7fc671a0 File: ./programs/LeanManagementToken/src/context.rs SHA3: ce4eab2a24dffb79890f730e17047d22550e02d1d3b260b060c586ab1076f439 File: ./programs/LeanManagementToken/src/error_codes.rs SHA3: 81322272435c5e8b2597b4dbbd53bb30022e20f1716c4642409e101ebce57309 File: ./programs/LeanManagementToken/src/lib.rs SHA3: 4752febf3ca2e8b2f57502eb39dc050edf581fc6f9552ca1ffdd09e2bce37888 File: ./programs/LeanManagementToken/src/utils.rs SHA3: da7daa072b790ac241c0d4bec0c95c501335636295c7d9cb44f00067b1e63a5e

Fourth review scope

Repository	https://github.com/Leancoin/Leancoin/
Commit	c5102aa2fba7fb9044b7d88dfcea3c026a8f1d8e
Whitepaper	https://docs.leancoin.io/leancoin-white-paper/
Functional Requirements	https://docs.leancoin.io/swap-lean/
Technical Requirements	./README.md
Program Id	CeFVa5iijJASnRmMCvrHep8wVYRZ3XxAmgXArNJhpjmx
SPL-token mint address	7297kX7SEZ1do223VsjTAC2MS9gLxPJoxFs9UMwiG4oS
Contracts	File: ./programs/LeanManagementToken/src/account.rs SHA3: bda0484adf3ca234d3b668d82ca381d2464207f314bf05018a7bdf7b7fc671a0

	File: ./programs/LeanManagementToken/src/context.rs SHA3: 8fc675d7bf3ef6cdfdb37f368f99634c02f3bf9c4c1a2ef6080a51a867839efc File: ./programs/LeanManagementToken/src/error_codes.rs SHA3: 81322272435c5e8b2597b4dbbd53bb30022e20f1716c4642409e101ebce57309 File: ./programs/LeanManagementToken/src/lib.rs SHA3: 24638d8b5a1a007474656ff04bffc0d31d4cc6da14ef87d4002addd9f6fabd44 File: ./programs/LeanManagementToken/src/utils.rs SHA3: da7daa072b790ac241c0d4bec0c95c501335636295c7d9cb44f00067b1e63a5e

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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
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 by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality.

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 technical description is clear and contains all essential commands on how to build/test/deploy the project.
- The functional requirements fully describe the user interaction flow and system-owned functionality.

Code quality

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

- Development environment is configured.
- Architecture and code purpose are clear.
- There is minor duplication of code and redundancy.
- Redundant `version` field in `./rust-toolchain.toml`.

Test coverage

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

- All contract methods are called during testing.
- Comments representing human-readable representation of used timestamps are incorrect.
- The Rust tests for `set_token_metadata(..)` are vacuous. However, the typescript integration tests are satisfactory; yet, an extra effort (the steps are not provided in the docs) is required to make them work.

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.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.

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

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

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
March 31, 2023	9	5	3	1
April 25, 2023	2	0	0	2
May 22, 2023	0	0	0	0
May 29, 2023	0	0	0	0

Risks

- The deployed code may differ from the one audited.
- Unless the smart contract is deployed with the `--final` parameter, it could be upgraded and its functionality may be changed.
- 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 correctness of migration via `leancoin::import_ethereum_token_state()` cannot be statically verified in the scope of the audit. Therefore, users should ensure that the contract state after the migration meets expectations.
- In case no one called the `leancoin::burn()` function in the first five days of the month, the “burning” wallet balance is not changed.
- It may be impossible to migrate a lot of accounts from Ethereum to Solana using the `leancoin::import_ethereum_token_state()` function.

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

System Overview

Leancoin is a fungible token that is migrated from Ethereum (ERC20) to Solana (SPL-Token based).

After contract deployment, a special migration function is supposed to be executed (no more than once). Its goal is to reflect the system wallet balances from the token on Ethereum, scaled by a constant factor (which represents the difference between the old and the new token precision/total supply).

Four special wallets are migrated: “community”, “partnership”, “marketing”, and “liquidity”. Each of the special wallet balances is locked according to a corresponding vesting schedule, whereby the contract owner can trigger a transfer of an unlocked vested amount to the designated externally-owned “deposit” wallet.

There is also a “burning” wallet which allows burning 5% of held funds in the first 5 days of each month.

Therefore, the token comes into circulation in two ways:

- By the initial migration. Some migrated accounts may be owned by general market participants.
- By vesting unlocks. The token amounts are moved from the special vesting wallets to the externally-owned deposit wallet, which in turn can distribute its balance to general market participants.

Privileged roles

Owner (represented by `account::ContractState::authority`) has the exclusive right to execute:

- `leancoin::importn_s_ethereum_toketate(..)` – allows the owner to import balances from implementation on Ethereum (only once)
- `leancoin::withdraw_tokens_from_community_wallet(..)` – allows the owner to withdraw vested funds from the community wallet
- `leancoin::withdraw_tokens_from_partnership_wallet(..)` – allows the owner to withdraw vested funds from the partnership wallet
- `leancoin::withdraw_tokens_from_marketing_wallet(..)` – allows the owner to withdraw vested funds from the marketing wallet
- `leancoin::withdraw_tokens_from_liquidity_wallet(..)` – allows the owner to withdraw vested funds from the liquidity wallet
- `leancoin::change_authority(..)` – allows the owner to transfer ownership
- `set_token_metadata(..)` - allows the owner to change the `name`, `symbol` and `uri` parameters of the token metadata.

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 Call Return Value	The return value of a message call should be checked.	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
Deprecated Rust Functions	Deprecated built-in functions should never be used.	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.	Not Relevant
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Attributes 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

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

Data Consistency	Smart contract data should be consistent all over the data flow.	Passed
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.	Passed
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	All Solana accounts holding an Account, Mint, or Multisig must contain enough SOL to be considered rent exempt. Otherwise, the accounts may fail to load.	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	Passed

	explorers, exchanges, etc.	
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. Denial of Service State

In the *calculate_month_difference* function, the *end.month - start.month* action is performed, where *DateTime.month* is *u8*. The case *end.month < start.month* is not processed, so an underflow (or panic, depending on the compiler configuration) may happen.

This may lead to the inability to perform vesting withdrawal if the current month is lower than the month of vesting start (for example, if the start date is in April 2023 and a withdrawal is done in January 2024).

Path: ./programs/LeanManagementToken/src/utils.rs: calculate_month_difference()

Recommendation: Convert the values to signed integers to avoid an integer underflow.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

C02. Denial of Service

The functions perform subtraction of unsigned integers to compute *already_withdrawn_amount* as: *initial_wallet_balance - wallet_account.amount*

For example, for the community wallet, *already_withdrawn_amount* is set as: *vesting_state.initial_community_wallet_balance - ctx.accounts.community_account.amount*

It is possible to increase *wallet_account.amount* by directly transferring tokens there. Once the migration is performed, *initial_wallet_balance* equals to *wallet_account.amount*, so it is enough to send *1* to *wallet_account* to make the computation of *already_withdrawn_amount* cause an integer underflow and panic.

The withdrawal functions attacked this way would be blocked forever, and the vested funds would become stuck.

Path: ./programs/LeanManagementToken/src/lib.rs:

- withdraw_tokens_from_community_wallet()
- withdraw_tokens_from_partnership_wallet()
- withdraw_tokens_from_marketing_wallet()
- withdraw_tokens_from_liquidity_wallet()

Recommendation: Process the case consciously. Consider making *already_withdrawn_amount* a signed integer. Alternatively - track

already_withdrawn_amount in a separate variable, instead of deriving it from the account balance.

Found in: 7a155aa

Status: Fixed (Revised commit: b5f7fa1)

C03. Denial of Service State

It is stated in the official documentation that the total supply of LEAN is 10 billion, and that the community wallet share is 10%; therefore, the initial community wallet balance is 1 billion or *10^9*.

According to *InitializeContext* (at src/context.rs), 1 LEAN is equal to *10^9* base units (note *mint::decimals = 9* at the *mint* field declaration). Therefore, the initial community wallet balance is *10^(9 + 9)* base units.

Because the computation of *amount_unlocked* is done in *u64*, the overflow in the expression *vesting_start_account_balance * (months_since_vesting_start + 1)* will happen when *months_since_vesting_start* becomes greater than or equal to *ceil(2^64 / vesting_start_account_balance - 1)*, where *vesting_start_account_balance* is equal to *10^18*. Therefore, *months_since_vesting_start* needs to be at least 18 for the overflow to happen.

Consequently, after 18 months since the vesting started, the community wallet withdrawal function *withdraw_tokens_from_community_wallet()* will become blocked indefinitely.

Note that the vesting period duration is supposed to be 39 months. Therefore, about 5.5% of the total supply would be blocked.

Path: ./programs/LeanManagementToken/src/lib.rs: calculate_unlocked_amount_community_wallet()

Recommendation: Do the computations in *u128*, then map the result back to *u64*.

Found in: 0e060f5

Status: Fixed (Revised commit: b5f7fa1)

High

H01. Requirements Violation

According to the vesting requirements, for each vesting wallet, it should not be possible to withdraw more funds than have been unlocked up to this moment.

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

The functions allow the withdrawal of funds that are not unlocked yet. This is because *unlocked_amount* equals the total unlocked amount of tokens at the moment, and the value is not modified by already withdrawn funds.

Once *unlocked_amount* is greater than zero, the wallet can be fully drained.

Path: ./programs/LeanManagementToken/src/lib.rs:

- withdraw_tokens_from_community_wallet()
- withdraw_tokens_from_partnership_wallet()
- withdraw_tokens_from_marketing_wallet()
- withdraw_tokens_from_liquidity_wallet()

Recommendation: Consider the amount that has been withdrawn up to the current point in time during the *amount_available_to_withdraw* value calculation.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

H02. Denial of Service State

In the function, the loop over the *year* variable may never exit.

This happens when the *days* variable equals *365*, and the *year* variable contains a leap year. In this case, no actions with *days* and *year* variables are performed within the cycle, and the break condition is not reached, so an infinite loop happens.

This may lead to the unavailability of some smart contract methods on December 31 of a leap year.

Path: ./programs/LeanManagementToken/src/utils.rs: parse_timestamp()

Recommendation: Add a *break* instruction for this case.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

Medium

M02. Inconsistent Data

In the function, the *match* statement should not process special wallets (“community”, “partnership”, “marketing”, “liquidity”) twice i.e. it should require that a wallet name is not duplicated.

This may lead to the wallets obtaining more funds than their initial balances are assigned, causing an inconsistent state situation.

Path: ./programs/LeanManagementToken/src/lib.rs: import_ethereum_token_state()

Recommendation: Implement the check for wallet duplicates.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

M03. Eager Division

Division is done too early, which worsens the rounding error.

- In `utils::calculate_unlocked_amount_marketing_wallet(..)`, the division by 100 on lines 330, 331 could be done as

  ```
  let amount_unlocked = (vesting_start_account_balance * 40 +
  (months_since_vesting_start - 12) *
  (vesting_start_account_balance * 5)) / 100
  ```

- In `utils::calculate_unlocked_amount_community_wallet(..)`, line 358 could be written as

  ```
  let amount_unlocked = vesting_start_account_balance *
  (months_since_vesting_start + 1) / 40
  ```

This may make it impossible to withdraw a small amount of vested tokens.

Path: ./programs/LeanManagementToken/src/utils.rs

Recommendation: Defer division as much as possible according to the suggestions in the description.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

M04. Documentation Mismatch

The functions contain the expression `amount_unlocked.max(1)` which produces a sharp rounding-up that goes against the vesting schedule formula declared in the documentation/comments.

Additionally, the situation hitting this rounding-up should not be possible in practice, because otherwise it means that the initial vesting balance is impractically small.

Path: ./programs/LeanManagementToken/src/utils.rs:
- calculate_unlocked_amount_marketing_wallet()
- calculate_unlocked_amount_community_wallet()

Recommendation: Remove `.max(1)` or explicitly document this behavior.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

M05. Immutable Ownership

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/LeanManagementToken/src/lib.rs

Recommendation: Implement an ability to transfer contract ownership.

Status: Fixed (Revised commit: 0e060f5)

M06. Best Practice Violation

The *utils::calculate_month_difference* function fully relies on *start <= end*.

Assuming that this condition is true, the conversion of result value via *unsigned_abs()* may lead to wrong assumptions about which input data is accepted and unexpected hidden bugs during future development.

Path: ./programs/LeanManagementToken/src/utils.rs: calculate_month_difference()

Recommendation: Use *try_from* instead of getting absolute value for cases where the value processed is expected to not be a negative number. Implement a *require* check to ensure that *start <= end*.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

Low

L02. Redundant Code

- *./programs/LeanManagementToken/src/utils.rs:338*: needless return
- *./programs/LeanManagementToken/src/utils.rs:231-235*: the *try_from* results can be explicitly unwrapped (the panic is impossible in those cases) to avoid having the verbose *match* statement.
- *./programs/LeanManagementToken/src/utils.rs:267*: *try_from* can be replaced with *from*, and the following *match* statement could be removed.

- *./programs/LeanManagementToken/src/utils.rs:320-321*: *try_from* can be replaced with *from*, and the *match* statement could be removed.
- *./programs/LeanManagementToken/src/utils.rs:335*: the *try_from* result can be explicitly unwrapped (the panic is impossible in that case, because the value is at most *vesting_start_account_balance*, which is known to fit *u64*), and the match statement could be eliminated.

Path: ./programs/LeanManagementToken/src/utils.rs

Recommendation: Eliminate the mentioned redundancies.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

L03. Usage Of Star Imports

*-imports are widely considered a bad style.

They complicate tracking dependencies, cause namespace pollution, and may lead to unexpected name clashes.

Paths: ./programs/LeanManagementToken/src/*

Recommendation: Import needed objects explicitly.

Found in: 7a155aa

Status: Mitigated (Star imports are actually dictated by the Anchor framework. Without them, it would be tricky to correctly add explicit member imports.)

L04. 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)

Paths: ./rust-toolchain.toml

Recommendation: Pin the language version at the project level.

Found in: 7a155aa

Status: Fixed (Revised commit: b5f7fa1)

L05. Vulnerable Dependency (Informational)

Vulnerability info: https://rustsec.org/advisories/RUSTSEC-2020-0071

Dependency path:
time 0.1.45
<- (...) <- solana-sdk 1.15.2
<- (...) <- (the project)

Path: ./programs/LeanManagementToken/Cargo.toml

Recommendation: N/A.

Found in: 7a155aa

Status: Mitigated (The issue does not affect the program code.)

L06. Redundant Architecture

The *DateTime* struct contains unused fields (*hours*, *minutes*, *seconds*, *days*). The field values are calculated and assigned in the function.

The code should not contain redundant variables and computations.

Path: ./programs/LeanManagementToken/src/utils.rs: DateTime, parse_timestamp()

Recommendation: Use the data or do not compute and store it.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

L07. Redundant Architecture

The error is never used. It is considered best practice to remove unused statements.

Path: ./programs/LeanManagementToken/src/error.rs: CannotGetBump

Recommendation: Remove the error declaration.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

L08. Redundant Calculations

The *is_leap* variable is recalculated in the cycle several times. However, it keeps the same value.

Path: ./programs/LeanManagementToken/src/utils.rs: parse_timestamp()

Recommendation: Declare the variable outside of the cycle.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

L09. Misleading Architecture

The *month_days* variable contains 13 values including the first month, which is zero days long. In such a way an additional cycle iteration is performed and month numbers start with *1*.

Similar purpose functionality is implemented in the same function for the *days* variable but in another way: it is increased by *1* at the end of the function.

Path: ./programs/LeanManagementToken/src/utils.rs: parse_timestamp()

Recommendation: Avoid usage of hard fixes like adding the 13th zero-length month, and implement code similarly to improve its readability.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

L10. Contradiction

The comment to the function contains wrong information:

So after 2 months: 5% of the initial balance is unlocked, after 3 months: 7.5%, after months: 10% etc.

However, as 2.5% is unlocked immediately after 2 months 7.5% would be unlocked, etc.

Path: ./programs/LeanManagementToken/src/utils.rs: calculate_unlocked_amount_community_wallet()

Recommendation: Provide correct examples to the code.

Found in: 7a155aa

Status: Fixed (Revised commit: 0e060f5)

H03. Documentation Mismatch

According to the comment, the function should return the number of full months between the dates. However, it is implemented in a way to return month differences ignoring *DateTime.day* values.

This may lead to wrong assumptions on the functionality behavior.

Path: ./programs/LeanManagementToken/src/utils.rs: calculate_month_difference()

Recommendation: Consider the *DateTime.day* value during the difference calculation or update the comment according to the implementation.

Found in: 0e060f5

Status: Fixed (Revised commit: 0e060f5)

L11. Redundant Code

- *./programs/LeanManagementToken/src/error_codes.rs:338*: the errors *CannotConvertToI64*, *CannotConvertToU8*, *CannotConvertToU128*, *CannotConvertToU64* are unused
- *./programs/LeanManagementToken/src/utils.rs:parse_timestamp()*:
  - The code pattern that determines whether a year is a leap year is repeated twice.
  - The last entry in *month_days* is never used.
  - *month_days* could be *const*.
- *./programs/LeanManagementToken/src/utils.rs:transfer_tokens()*:
  - Redundant lifetime specifier *'b*

Path: ./programs/LeanManagementToken/src/

Recommendation: Eliminate the mentioned redundancies.

Found in: 0e060f5

Status: Fixed (Revised commit: b5f7fa1)

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.