

# Ref.Finance Security Audit

**Second Draft**

May 2022

Auditor: Dr J. Rousselot

# Executive Summary

Ref Finance is the most widely used DEX on NEAR protocol

It is written in the Rust programming language.

The audit covers the Ref Finance codebase available at

<https://github.com/ref-finance/ref-contracts>

commit `3c04fd20767ad7f1c383deee8e0a2b5ab47fbc18` dated 17 December 2021 on branch `main`,

As well as the Ref Boost Farm codebase available at

<https://github.com/marco-sundsk/boost-farm>

commit `eadddbcd7a69569534e60596596844ed83e9cab6` dated 4 May 2022 on branch `main`

The Ref Finance code is organized in two projects: `ref-exchange` and `ref-farming`. The code in the `ref-farming` directory was replaced by the second generation farming code: Ref Boost Farm and thus the `ref-farming` codebase was not audited.

To the best of our knowledge, we did not find any critical security issue with the codebase in the version audited.

## Issues

We made the following observations during our audit. More details and recommendations can be found in the respective sections.

Each issue risk severity is estimated following the OWASP risk rating methodology. An overall risk severity is made based on both the likelihood of the issue being exploited and the technical/business/financial impact of the issue.

Issue	Risk Severity	Explanation
1.4.1 Version pinning	High Likelihood: 5 Impact: 7	It is theoretically possible for an attacker to compromise a third party dependency and inject an attack through it. Recommendation: add version pinning.
1.4.3 Reproducible builds	High Likelihood: 5 Impact: 9	Currently, it is not possible to prove that the DEX code deployed on chain matches the source code on github. Recommendations: improve docker builds and prioritize reproducible builds.
1.6.1.3 Always use <code>assert_one_yocto</code> in <code>owner.rs</code>	High Likelihood: 5 Impact: 7	All functions restricted to owner and/or guardians should use the <code>assert_one_yocto</code> . This prevents these user

		roles from delegating access to sub-accounts.
1.5.1 Unit test coverage	Low Likelihood:2 Impact: 7	Some security critical smart contract functions could be covered by unit testing.
1.5.2 Unit test framework	Low Likelihood: 2 Impact: 7	The NEAR unit testing framework relies on a simulator rather than actual blockchain node code.
1.6.2.3/1.6.2.4 the code is slightly too complex as only one action is supported at the moment.	Comment Likelihood: 2 Impact: 2	Only the SWAP action is currently implemented. If no other actions are planned, the code could be simplified.
1.6.2.5 Non fully standardized error codes	Comment Likelihood: 2 Impact: 2	Instead of using a string for the error, it is better coding practice to return a specific item from the Errors enum (ERR_NO_POOL).
1.6.2.5 Temporary double spend within an atomic tx	Comment Likelihood: 1 Impact: 7	When adding liquidity to a pool, very temporarily more tokens exist in the system during the function call. As transactions are atomic and not parallelized there is no security risk.
1.6.3.1 always use the same numeric constant NUM_TOKENS	Comment Likelihood: 1 Impact: 3	The constant NUM_TOKENS can be used to check whether function arguments have the correct size. Using this coding style is clearer than referencing other objects which have previously been checked with NUM_TOKENS such as token_account_ids.len()
1.6.3.9 refactor assert to compare object length with NUM_TOKENS into a function check_num_tokens()	Comment Likelihood: 1 Impact: 3	Input validation of vectors can be further refactored by introducing an internal function check_num_tokens_length(Vec<>).

The following table summarizes the issues found with boost-farm:

Issue	Risk Severity	Explanation
-------	---------------	-------------

2.1 Reproducible builds Fix typo in script that always return success	High Likelihood: 5 Impact: 7	scripts/codehash.py contains a typo that compares the reference file with itself.  Replace: code_hash_build = bytes.decode(base58.b58encode(CalcFileSha256(release_file))) with: code_hash_build = bytes.decode(base58.b58encode(CalcFileSha256(build_file)))
2.2 Unit tests Fix docker / root files permission issue	Low Likelihood: 5 Impact: 3	<b>Recommendation:</b> change permissions of all files copied from the Docker image (running as root) to the local directory. Alternatively, run tests from within a Docker container.
2.2 Unit tests Increase tests coverage	Low Likelihood: 2 Impact: 7	As the codebase is new, unit tests coverage is still relatively low (<20%). Increasing unit tests coverage in the future would further improve trust in the system.
2.3.7 management.rs	Comment Likelihood: 1 Impact: 3	Recommendation: in return_seed_lostfound, the farmerId account could be compared with the null account to prevent accidentally burning funds.
2.3.9 seed_farm.rs	Low Likelihood: 5 Impact: 3	Recommendation: clarify what happens to the amount_of_beneficiary? How is it withdrawn?
2.3.10 token_receiver.rs	Comment Likelihood: 3 Impact: 1	Recommendation: consider defining an upper bound value for max_locking_multiplier.
2.3.18 farmer.rs	Comment Likelihood: 3 Impact: 1	Recommendation: to improve understanding of the code, rename seed_power to farmer_seed_power when computing reward claims.

## Disclaimer

The auditor and the auditing company made their best efforts to identify any security issue related to the codebase under audit.

They cannot be held responsible for any remaining security vulnerability in the software product or for any financial loss that the usage of the software product may cause to anyone.

A security audit is not financial advice.

<b>1 REF-Exchange</b>	<b>8</b>
1.1 User Roles	8
1.2 Pools	8
1.3 Tokens	9
1.3.1 Fee on Transfer	9
1.3.2 Rebasing Tokens	9
1.4 Software Architecture	10
1.4.1 Version pinning	11
1.4.2 Docker builds	11
1.4.3 Reproducible builds	12
1.5 Unit Tests	12
1.5.1 Unit test coverage	12
1.5.2 NEAR Simulation framework	14
1.6 Smart Contracts	14
1.6.1 owner.rs	14
1.6.2 lib.rs	14
1.6.3 simple_pool.rs	15
1.7 Sequences	16
1.7.1 Creation of a new pool	16
1.7.2 Swap	17
1.7.3 Providing liquidity	17
1.7.4 Withdrawing liquidity	17
<b>2 Boost Farm</b>	<b>18</b>
2.1 Reproducible builds	18
2.2 Unit Tests and Coverage	18
2.3 Smart Contracts	19
2.3.1 events.rs	19
2.3.2 lib.rs	20
2.3.3 owner.rs	20
2.3.4 errors.rs	20
2.3.5 utils.rs	20
2.3.6 view.rs	20
2.3.7 management.rs	20
2.3.8 seed.rs	21
2.3.9 seed_farm.rs	21
2.3.10 token_receiver.rs	21
2.3.11 storage_impl.rs	22

2.3.12 actions_of_seed.rs	22
2.3.13 actions_of_farmer_seed.rs	22
2.3.14 actions_of_farmer_reward.rs	23
2.3.15 big_decimal.rs	23
2.3.16 booster.rs	23
2.3.17 farmer_seed.rs	23
2.3.18 farmer.rs	24
2.4 Sequences	24
2.4.1 Creation of a new seed	24
2.4.2 Creation of a new farm	24
2.4.3 Cancel a farm	25
2.4.4 Remove a farm	25
2.4.5 Update a booster	25
2.5 Annex 1: unit tests	25
2.6 Annex 2: Code Coverage	27

# 1 REF-Exchange

REF-Exchange is a decentralized exchange running on the NEAR blockchain.

The following sections respectively describe the different user roles within REF, the functioning of the pools, the software architecture of the DEX, a review of the unit tests and the audit of the smart contract codebase itself.

## 1.1 User Roles

A DEX user can be either a *liquidity provider (LP)* or a *trader*.

Any REF user can be a LP or a trader.

Besides LP and traders, there are two other user roles in the system: *owner* and *guardians*.

As a *liquidity provider*, any NEAR user can:

- create a new trading pool using the white listed tokens,
- provide liquidity to an already existing ref-exchange trading pool,
- withdraw liquidity from an existing pool.

As a *trader*, any NEAR user can also:

- make a trade on a pool.

The *owner* can:

- add and remove guardians,
- create new pools,
- upgrade the smart contract,
- change the owner,
- change the REF Exchange state (running or paused),
- set the fees,
- do most/all actions allowed for guardians.

*Guardians* can:

- add/remove whitelisted tokens,
- set the REF Exchange state to paused,
- set the fees.

## 1.2 Pools

The trading pool implements the same pricing mechanism as uniswapV1 and V2:  $X * Y = K$ .

As described in the UniSwapV2 docs: *Where X and Y represent the respective reserve balances of two ERC-20 tokens, and “K” represents the product of the reserves.*

There are two challenges with such pools:



1. They require a large initial deposit by the pool creator.
2. Impermanent loss to LP.

In practice, the large initial deposit is difficult to quantify. The REF team made the design choice to let pools trade with any amount of tokens. If we consider very valuable tokens, it is realistic for users to trade small amounts of token A against small amounts of token B.

The REF frontend implements several features to help the user make informed decisions.

## 1.3 Tokens

Besides, there are also several possible trading errors associated with  $X*Y=K$ , which depend on the algorithmic behavior of the tokens being traded. With ETH / UniswapV2, the most common ones are *fee on transfer* tokens, and *rebasing* tokens.

These issues are caused by the token smart contracts and their operators themselves, rather than the DEX codebase. However, these issues can interfere with the proper operation of a DEX.

More info on the UniswapV2 issues can be found at the following link:

<https://docs.uniswap.org/protocol/V2/reference/smart-contracts/common-errors#:~:text=The%20Uniswap%20constant%20product%20formula,the%20%E2%80%9CK%E2%80%9D%20error%20refers.>

### 1.3.1 Fee on Transfer

Fee on transfer is caused by some ERC20 tokens charging a fee on each transfer.

On Ethereum, this can break the pool as the equation  $X*Y = K$  is not true anymore.

With REF / NEAR, the user first deposits tokens to the exchange before moving them into the pool. Tokens moved to/from the pool are not moved from the token's point of view (see section 1.7 for details). Therefore the Ethereum ERC20 Fee on Transfer should not be an issue with the REF pool operation. At the time of writing no such tokens seem to exist on NEAR yet.

It would be useful to implement a unit test to confirm this analysis.

### 1.3.2 Rebasing Tokens

Rebasing tokens are caused by the token administrators / issuers issuing more tokens, or alternatively burning tokens.

- A) Consider a token A with 100 units in circulation. If the administrators suddenly issue 900 more tokens and distribute them equally among token holders, all things being equal, the value of this token is likely going down by a factor 10.
- B) Conversely, if the admin of token A with 100 units in circulation burns 90 of the tokens, equally from each token holder, then the value of each token suddenly climbs by a factor 10, all things being equal.

In scenario A, the DEX would hold for instance 10 units. All 10 tokens have been allocated to one pool. Then, token A admins issue and distribute 90 more tokens. These new tokens have

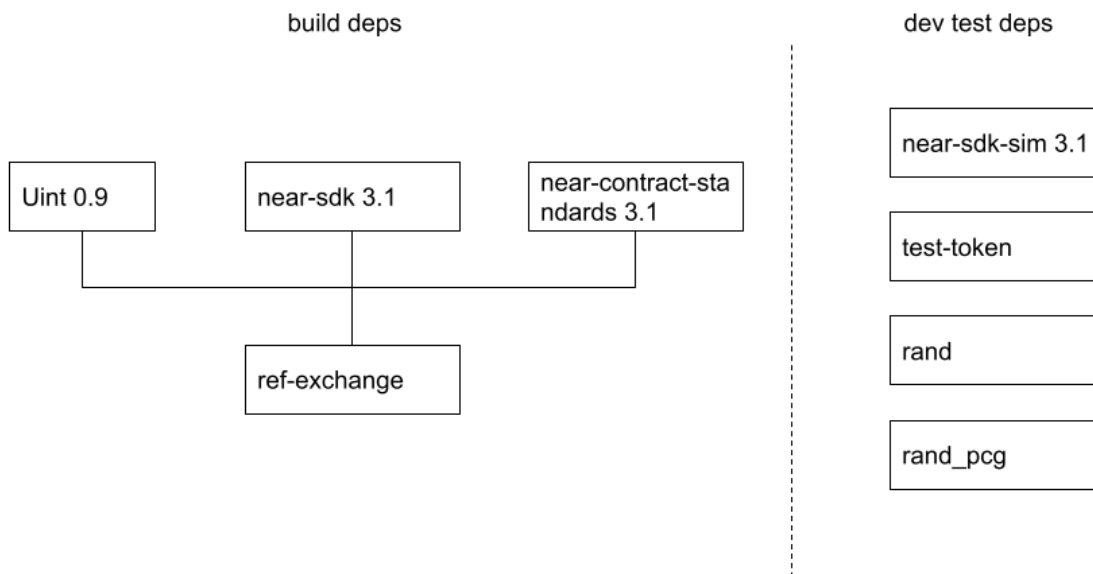
not been allocated to any pool. So the pool balance remains the same. These tokens are not owned by anyone. Arbitrage players will rebalance the pool, adding more A and withdrawing Y. In scenario B, the DEX would hold 10 units. All 10 tokens have been allocated to one pool. After burning 90% of the tokens, only 1 is left from the token's point of view. However, the DEX does not know about this. In the pool, there are still 10 (assuming all allocated to the pool). When users try to withdraw these tokens from the DEX, this will fail as from the token's contract point of view, there are 10 times fewer tokens than the DEX assumes.

## 1.4 Software Architecture

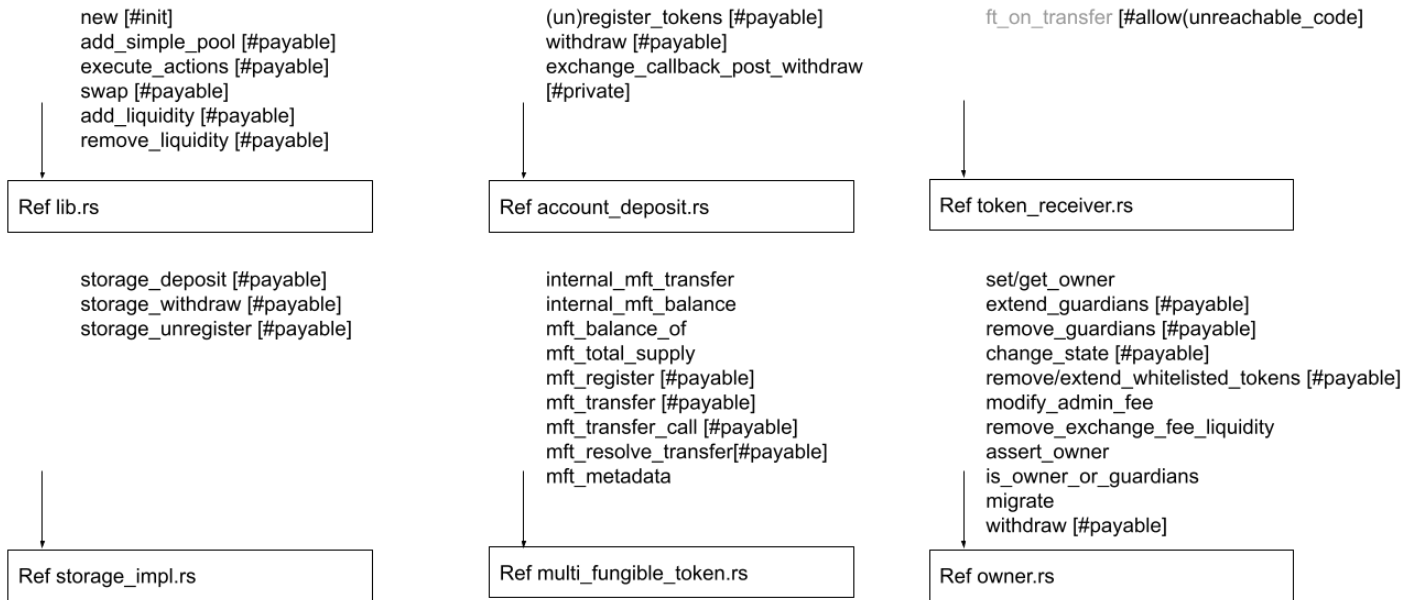
The ref-exchange codebase is organized as follows.

1. src holds the smart contracts themselves,
2. tests directory holds all the tests.
3. tests/fuzzy holds some randomized tests.
4. ../test-token an external directory holds the source code for a test token.

The figure below represents all package dependencies for both the production build target and for the development build target. We can see that few packages are used. This is a good security practice as it minimizes third party risks.



The next figure shows the list of functions that can be called by users of the smart contract.



Not shown: legacy.rs and views.rs

### 1.4.1 Version pinning

The build system uses Cargo, the Rust package manager. Some dependencies in the Cargo manifest file are insufficiently specified. For instance, package `uint` is set at version 0.9. This lets the build system silently download newer patch versions (0.9.2, 0.9.3) than the one intended by the developer (0.9.1). As the `uint` package maintainers released a breaking change in 0.9.2, the code becomes impossible to build (as it requires a more recent version of `rust`). Requesting package versions including the patch number, or using Cargo lock files would make sure all developers build using the exact same libraries.

The Rust specification explains that patch number should not introduce breaking changes:

<https://doc.rust-lang.org/cargo/reference/manifest.html#the-version-field>

The `uint` library upgraded Rust edition from 2018 to 2021 here:

<https://github.com/paritytech/parity-common/commit/c3ef97d403bf30fb4ff35ce9ac74b60c3a0c8f42#diff-fef31bc61fbea5a6eaf423dc2ebb52c192debb61a579da44d9b58768a27f2f7>

Which requires `rustc 1.56.1`:

<https://github.com/paritytech/parity-common/pull/601>

### 1.4.2 Docker builds

Docker is an important tool to standardize the build system and to make sure developers work on the same software version.

We recommend that REF fixes the Docker build scripts at least for the intel/amd64 Linux platform and introduce a Dockerfile in the repository instead of relying on Makefiles / shell scripts.

The table below summarizes our attempts to build the software on various platforms.

	amd64 Ubuntu 20.04 build_local.sh	amd64 Ubuntu 20.04 build_docker.sh	Apple Silicon MacOS 12.2 build_local.sh	Apple Silicon MacOS 12.2 build_docker.sh
ref-exchange contracts	OK	breaks	OK	breaks
ref-farming	OK	breaks	OK	breaks

### 1.4.3 Reproducible builds

We found a major issue around the system on-chain deployment. Despite our best effort, we could not reproduce the WASM binary used by REF Finance for on-chain deployment. We recommend REF to prioritize reproducible builds. This is partly a wider NEAR ecosystem issue, but immediate actions related to dependency packages pinning can be taken now, as well as using docker build automation. As of writing, we have no way of proving that the code deployed on chain is the same as the code being audited.

The ultimate goal of reproducible builds is to generate the exact same binary file (verified with a hash) on various platforms: amd64 and arm architectures, Linux and MacOS operating systems. We recommend that REF implements version pinning, fixes Docker builds and works on reproducible builds.

## 1.5 Unit Tests

We reviewed several Rust tools to generate a unit test code coverage report: GRCOV, LCOV and Cargo Tarpaulin.

Cargo Tarpaulin is a framework that generates html reports and we recommend its use.

More details can be found in the Tarpaulin report.

For reference, tarpaulin can be run as follows:

```
Requires: Linux x64
apt install pkg-config libssl-dev
cargo install cargo-tarpaulin
cargo tarpaulin
```

### 1.5.1 Unit test coverage

The figure below from the tarpaulin report shows that the files admin\_fee.rs, lib.rs, action.rs, simple\_pool.rs, utils.rs are well covered (85 to 100% test coverage).
















We also observe that the files legacy.rs, owner.rs, storage\_impl.rs, token\_receiver.rs and views.rs have low or minimal coverage. While some code such as view functions may be low risk, it would be good practice to increase the coverage for most of the code. It is especially important for security sensitive functions such as the code in owner.rs.

This effort would prove programmatically (within the tests scenarios considered) that the security mechanisms are correctly implemented. It would also prevent any future mistake from degrading this security level, either by changes to the REF smart contract themselves or from silent dependency package upgrades through the Cargo package manager. Currently REF does not use a Cargo lock file, and does not use pin package versions in the Cargo.toml file directly. Third party dependencies can thus introduce changes, although luckily few dependencies are used by the codebase at the moment.

To summarize, we recommend to increase unit tests code coverage for the following modules:

- account\_deposit.rs
- owner.rs (set/get owner/guardians, change\_state, modify\_admin\_fee, remove\_exchange\_fee\_liquidity)
- pool.rs
- token\_receiver.s (especially the instant swap does not seem to be covered)

Ideally, all functions in the list of entry points in the previous section should be well tested.

 stable_swap	507 / 599 (84.64%)
 account_deposit.rs	110 / 164 (67.07%)
 action.rs	6 / 7 (85.71%)
 admin_fee.rs	4 / 4 (100.00%)
 errors.rs	0 / 0
 legacy.rs	0 / 5 (0.00%)
 lib.rs	443 / 500 (88.60%)
 multi_fungible_token.rs	55 / 101 (54.46%)
 owner.rs	10 / 77 (12.99%)
 pool.rs	31 / 73 (42.47%)
 simple_pool.rs	176 / 186 (94.62%)
 storage_impl.rs	21 / 49 (42.86%)
 token_receiver.rs	6 / 27 (22.22%)
 utils.rs	19 / 19 (100.00%)
 views.rs	19 / 85 (22.35%)

Note that because of the current state of Rust unit tests coverage tools, it is possible that tarpaulin report is not entirely accurate. Some new features exist in the experimental branch of

Rust toolchain but the NEAR framework is not compatible with them yet (as they require wasm 2 and NEAR uses wasm 1.x).

## 1.5.2 NEAR Simulation framework

All unit tests from REF use the so-called NEAR simulation framework. This is the most widely used and most developed tool for unit testing within REF.

However, NEAR has developed the near-sandbox to run a local blockchain for testing.

This approach has several advantages over the NEAR simulator:

- More realistic as uses same codebase as production NEAR nodes,
- Gas cost estimations,
- Ability to write and run tests in NodeJS,
- As tests are run through RPC they can also be run on testnet and mainnet.

At the time of writing, it seems that NEAR Sandbox is not ready for production yet.

We recommend that at some point in the future, test development migrate over Sandbox.

## 1.6 Smart Contracts

### 1.6.1 owner.rs

We found no specific issue in this module. The code is clean and easy to understand and to audit. Improving unit testing would further improve the code quality.

1. **set\_owner:**  
This is a critical function. If an incorrect new owner is set, there is no way to recover from this and all fees will be lost.
2. **change\_state:**  
Only the owner can resume the contract after it has been paused.
3. **assert\_one\_yocto:**  
All functions in this module will benefit from using `assert_one_yocto()` to prevent owner and guardians from delegating their capabilities and increasing attack surface (more keys -> more possibility that a key gets stolen).

### 1.6.2 lib.rs

This is the core module of ref-exchange.

Contract

1. **init/new**
2. **add\_simple\_pool:**  
by design, the system lets users create multiple pools with the same token pair. This enables competition on fees but can reduce liquidity.  
The total fee set by the pool creator is verified to be under a constant maximum value.

### 3. **execute\_actions:**

batched atomic DEX actions execution for any account. Some input validation. At the moment only one action (Action::Swap) is supported. The code will reject any other action type as enum Action only has one enum. We recommend filtering either explicitly at the entry\_point the Action values, and/or implementing a default error handler for internal\_execute\_action. The code can be made more explicit and robust by filtering out immediately unimplemented actions. Incorrect poolId is filtered and will stop tx.

### 4. **swap:**

Let a user execute multiple SwapActions in a single tx. Contract mode is checked. The difference between the functions swap and execute\_actions is not clear. If no other actions are under development, it may be better to simplify the codebase and keep only one entry point.

### 5. **add\_liquidity:**

This function adds liquidity to a pool in proportion to the pool ratio.

"ERR\_NO\_POOL" should be from the errors enum .

Ideally, we always want to subtract amounts before adding amounts when operating fund transfers. Here we add tokens to the pool before removing them from the user's account. As this is all internal accounting within the DEX smart contract / account, and contained within a single atomic transaction, it is very unlikely to be a risk. Still, for a very short time we create too many tokens in the system (tokens are both in the user's deposit account and in the pool).

### 6. **remove\_liquidity:**

This function removes some or all of the liquidity previously deposited by a LP.

## 1.6.3 simple\_pool.rs

### 1. **new**

Use NUM\_TOKENS for initialization of amounts, volumes, instead of token\_account\_ids.len()

### 2. **share\_register**

### 3. **share\_transfer**

### 4. **share\_balance\_of**

### 5. **share\_total\_balance**

### 6. **tokens**

### 7. **add\_liquidity**

### 8. **mint\_shares**

### 9. **remove\_liquidity**

Refactor assert\_eq!(...) wrong token count into a function check\_token\_count(Vec<T>); always compare directly with NUM\_TOKENS (rather than self.token\_accounts\_ids.len())  
ERR\_WRONG\_TOKEN\_COUNT, ERR\_SHOULD\_HAVE\_2\_TOKENS  
Add ERR to Errors enum

**10. token\_index**

Simple function but important for security. As input tokens are validated through this call.  
Add ERR to Errors enum

**11. internal\_get\_return**

Critical function for trade estimation. Returns the estimated amount V of token B for input amount of token A. The formulas used in the codebase are an optimization to improve arithmetic accuracy. The following computations below verify the formulas implemented in the codebase.

We have:

$$K = \text{amount\_in} * (\text{FEE\_DIV} - \text{total\_fee}) = \text{amount\_in} * \text{FEE\_DIV} * (1 - \text{total\_fee} / \text{FEE\_DIV}) \\ = \text{amount\_in} * \text{FEE\_DIV} - \text{amount\_in} * \text{total\_fee}$$

$$V = K * \text{out\_balance} / (\text{FEE\_DIV} * \text{in\_balance} + K) \\ = [ \text{amount\_in} * \text{FEE\_DIV} * \text{out\_balance} - \text{amount\_in} * \text{total\_fee} * \text{out\_balance} ] / [ \text{FEE\_DIV} * \text{in\_balance} + \text{amount\_in} * \text{FEE\_DIV} - \text{amount\_in} * \text{total\_fee} ] \\ = \text{amount\_in} * \text{out\_balance} (1 - \text{total\_fee} / \text{FEE\_DIV}) / [ \text{in\_balance} + \text{amount\_in} (1 - \text{total\_fee} / \text{FEE\_DIV}) ] \\ = \text{amount\_in} * \text{out\_balance} / (\text{in\_balance} + \text{amount\_in})$$

**12. get\_return**

wrapper for internal\_get\_return.

This function validates that both input and output tokens requested correspond to the pool and return the result of internal\_get\_return as a Balance object.

**13. get\_fee**

Getter function for total\_fee

**14. get\_volumes**

Getter function for volume statistics

**15. swap**

Performs the swap operation based on the amount from internal\_get\_return.

Verifies that the pool invariant is respected.

Mint and distributes shares as needed.

Finally, update volume statistics.

## 1.7 Sequences

This section describes how the product features described in section 1.1 are implemented function call by function call.

It is possible to execute these features using alternate / simplified flows if the user already has some internal balance within the DEX.

### 1.7.1 Creation of a new pool

1. The owner or a guardian whitelists a new guardian G through extend\_guardians.
2. The new guardian G whitelists a new token T through extend\_whitelisted\_tokens.\*
3. A user U (possibly same as G) creates a new pool add\_simple\_pool.



4. User U deposits some amount of token T to the contract account and NEAR calls `ft_on_transfer` and `internal_deposit`.
5. User U deposits some amount of previously whitelisted token W in the same manner as in the previous step.
6. User U adds liquidity with token T, NEAR or token T, token W to the pool using `add_liquidity`.
7. If the pool has enough liquidity, it is possible to trade.

(\*) Step 2 is optional. It is technically possible to create a pool of non whitelisted tokens. However, the REF frontend will filter it out from end users.

### 1.7.2 Swap

Once a pool exists and has enough liquidity, the following sequence is possible:

1. The user deposits an amount of token T to the DEX.
2. The user calls function `swap` with an action of type `SWAP` (only action supported so far) to convert some of his token T balance to the pool pair asset. If the amount is above the minimum amount, the user balance in the pool is updated accordingly.

### 1.7.3 Providing liquidity

A user can contribute liquidity to a pool as follows:

1. User U deposits some amount of token T to the contract account and NEAR calls `ft_on_transfer` and `internal_deposit`.
2. User U deposits some amount of previously whitelisted token W in the same manner as in the previous step.
3. `add_liquidity` transfers the liquidity from the user's internal account within the DEX to the actual pool. The user is credited shares in the pool.

### 1.7.4 Withdrawing liquidity

If the user already contributed to a pool liquidity, it is possible to withdraw it using the following function call:

1. `remove_liquidity` will reduce the pool liquidity accordingly, remove the shares from the user's balance and credit its internal token balance.

## 2 Boost Farm

User Roles

Pools

Tokens

Software Architecture

### 2.1 Reproducible builds

Boost Farm implements reproducible builds.

The Dockerfile used for building Boost Farm always uses the same exact Docker image: rust:1.56.1.

The Docker build is obtained by calling: make release

And the build output file is placed in directory res/boost\_farming\_release.sh

This file can then be compared with the reference file provided by the Boost Farm dev team in the repository at: releases/boost\_farming\_release.sh

This is implemented in codehash.sh and scripts/codehash.py.

**Recommendation:** the file scripts/codehash.py contains a typo that compares the reference file with itself. The line to obtain the hash of the build file is:

```
code_hash_build = bytes.decode(base58.b58encode(CalcFileSha256(release_file)))
```

While it should be:

```
code_hash_build = bytes.decode(base58.b58encode(CalcFileSha256(build_file)))
```

### 2.2 Unit Tests and Coverage

We encountered some errors while trying to run the tests. This is due to permission issues with files copied from the Docker images.

**Recommendation:** change permissions of all files copied from the Docker image (running as root) to the local directory. Alternatively, run tests from within a Docker container.




















See Annex 1 for tests output.

All tests run and pass successfully.

19% of the rust boost-farm smart contracts codebase are covered by unit tests.

This is a very new codebase so this is understandable.

**Recommendation:** in the medium to long term, a team should aim for 100% smart contracts unit tests code coverage.

 actions_of_farmer_reward.rs	0 / 34 (0.00%)
 actions_of_farmer_seed.rs	0 / 102 (0.00%)
 actions_of_seed.rs	0 / 59 (0.00%)
 big_decimal.rs	110 / 151 (72.85%)
 booster.rs	0 / 40 (0.00%)
 errors.rs	0 / 0
 events.rs	93 / 95 (97.89%)
 farmer.rs	0 / 62 (0.00%)
 farmer_seed.rs	0 / 88 (0.00%)
 legacy.rs	0 / 6 (0.00%)
 lib.rs	0 / 24 (0.00%)
 management.rs	0 / 120 (0.00%)
 owner.rs	0 / 19 (0.00%)
 seed.rs	0 / 29 (0.00%)
 seed_farm.rs	0 / 56 (0.00%)
 storage_impl.rs	0 / 38 (0.00%)
 token_receiver.rs	0 / 59 (0.00%)
 utils.rs	6 / 32 (18.75%)
 view.rs	0 / 95 (0.00%)

## 2.3 Smart Contracts

### 2.3.1 events.rs

The following events are defined:

SeedCreate

FarmCreate

FarmCancel

RewardDeposit

SeedDeposit

SeedFreeToLock

SeedUnlock

SeedWithdraw

SeedWithdrawLostFound

SeedWithdrawSlashed

RewardWithdraw

Events structure implements the NEP297 standard.

fn emit\_event: We recommend to use the standard name "nep297" rather than "boost-farm", to be compliant with the standard. It is possible to start every event type with the string "boost-farm" if needed (e.g. boost-farm::SeedCreate, boost-farm::FarmCreate, etc).

### 2.3.2 lib.rs

The contract defines the main data structures used by Farm Boost. Most of the logic is implemented in other modules.

### 2.3.3 owner.rs

This module lets the owner upgrade the contract to a new version, change the owner, extend and remove operators.

Recommendation: check for invalid address 0 for functions set\_owner, extend\_operators and remove\_operators.

### 2.3.4 errors.rs

This module lists all error messages used by the codebase.

Error messages are easy to understand and this module facilitates reviewing all possible error conditions.

### 2.3.5 utils.rs

The util module defines important types (U256) and protocol constants.

The minimum locking duration is set to 30 days and the maximum to 360 days.

Several constants set the prices to use for blockchain storage costs.

The minimum seed deposit is set to 1E18.

### 2.3.6 view.rs

This module provides view functions for many important contract state variables. This is very useful to understand the state of the system at any time.

There are no restrictions on who can call these functions.

We reviewed the functions and can confirm that they do not modify the contract state.

### 2.3.7 management.rs

The management module implements functions that can all be called by an operator or the owner, return\_seed\_lostfound excepted.

modify\_daily\_reward retrieves a farm and its seed if they both exist, and update the daily reward as requested.

modify\_locking\_policy updates both the max\_duration and max\_ratio for locking.

modify\_max\_farm\_num\_per\_seed updates the maximum number of farms for a given seed.

modify\_default\_slash\_rate updates the default slash rate.

`modify_seed_min_deposit` updates the minimum seed deposit for a given seed.

`modify_seed_min_locking_duration` updates the minimum locking duration and verifies that the new value is smaller than the global max value.

`modify_seed_slash_rate` updates the slash rate for the selected seed.

`withdraw_seed_slashed` is likely the most sensitive function in this module. It enables the owner or one of the operators to withdraw the seed slashed amount. For security reasons, the funds are always sent to the owner id account. If for some reason the transaction fails, the callback function transfers the funds back into the seed slashed pool. In any case (success or failure), an event is generated.

`return_seed_lostfound` returns the lost and found seed amounts to a farmer id address set by the owner. Requested amount should be lower than the available amount, which can be found with the `list_lostfound` function. The logic is similar to `withdraw_seed_slashed`.

**Recommendation:** in `return_seed_lostfound`, the `farmerId` account could be compared with the null account to prevent accidentally burning funds.

### 2.3.8 seed.rs

This module represents a seed, which corresponds to a farming token.

There is only one seed per farming token.

The seed keeps track of all the farms associated with it.

The never decreasing counter `next_index` corresponds to all farms ever created for that seed.

The `total_seed_amount` reflects the total seed amount staked to the farm free and locked from `token_receiver`, and `total_seed_power` the equivalent total seed power.

Only one version of Seed is currently supported.

### 2.3.9 seed\_farm.rs

This module represents a farm for a seed.

Besides `FarmTerms`, a Farm has several balances:

- `total_reward`: an always increasing counter representing all rewards ever deposited into the farm (equal to zero until the farm starts),
- `distributed_reward`: amount of rewards that have been distributed so far (if `total_reward` = `distributed_reward` all rewards have been distributed and the farm has ended)
- `claimed_reward`: amount of rewards that have been claimed by the farmers (always smaller than or equal to `distributed_reward`)
- `amount_of_beneficiary`: once the farm has ended, amount of distributed reward that were not claimed by farmers.

**Recommendation:** clarify what happens to the `amount_of_beneficiary`? How is it withdrawn?

### 2.3.10 token\_receiver.rs

This module implements the logic for token reception.

The transfer should come together with a `TokenReceiverMessage` which can have three values; Free, Lock and Reward.

In case of Free, the token amount transferred to the farm is moved to the internal token balance. First, the code computes all rewards due to the farmer who deposited the new token amount (taking into account seed power), and all such rewards are claimed. The code updates the boost ratios for the farmer, as well as the total seed power for that seed.

Seed power is computed as follows. It is the sum of the basic seed power and all of the extras. The basic seed power is the sum of the free amount and of the power of the locked amount. The extras

This computation depends on the configuration parameter `max_locking_multiplier`. It can only be updated by `modify_locking_policy`, which verifies that the new value is within range.

**Recommendation:** consider defining an upper bound value for `max_locking_multiplier`.

### 2.3.11 storage\_impl.rs

This module handles data storage costs.

It implements the following functions:

- `storage_deposit`
- `storage_withdraw`
- `storage_unregister`
- `storage_balance_bounds`
- `storage_balance_of`

### 2.3.12 actions\_of\_seed.rs

`create_seed` can only be called by the owner or an operator.

There can be only one seed per token.

A seed has a default slash rate, a minimum deposit and a minimum locking duration in seconds. If all conditions are met, an event `SeedCreate` is generated.

`create_farm` can only be called by owner or an operator.

Each farm has its own farm terms, and up to `max_num_farms` can be created for the same seed.

The data structure representing a farm is `SeedFarm`, together with `FarmTerms`.

The counter `farm_count` is updated to reflect the total number of farms.

`cancel_farm` enables farm creators to cancel a farm before any reward was deposited, and `remove_farm_from_seed` takes it offline (and removes it from the farms count).

### 2.3.13 actions\_of\_farmer\_seed.rs

This module defines the actions that a farmer can take with a seed.

lock\_free\_seed enables the farmer to lock a previously deposited free amount.

The increased seed power is computed and used to update the total seed power.

A SeedFreeToLock is generated in case of success.

Unlock\_and\_withdraw\_seed lets the farmer unlock and withdraw the seed. If the slash penalty should be applied, the transaction will fail. Instead, the farmer should call force\_unlock. Both functions generate a SeedUnlock event in case of success, and a SeedWithdraw event at the end of the withdrawal attempt.

### 2.3.14 actions\_of\_farmer\_reward.rs

claim\_reward\_by\_seed lets a farmer claim his rewards for a seed.

Withdraw\_reward lets a farmer withdraw the claimed rewards by token, optionally with a set amount. The event RewardWithdraw is generated, with a success flag set to 1 or 0.

### 2.3.15 big\_decimal.rs

This module implements code to support big decimal arithmetic.

This code comes from

[https://github.com/NearDeFi/burrowland/blob/main/contract/src/big\\_decimal.rs](https://github.com/NearDeFi/burrowland/blob/main/contract/src/big_decimal.rs)

Two uint types are defined: one built over 256 bits, and another higher precision with 384 bits.

A BigDecimal type is built on the uint384 type.

The year is defined as 31536000000 ms, which is equal to 365 days.

The first 27 digits of a 384 bits uint are treated as decimals.

There is some unit test coverage for the types, including power computations which indirectly covers multiplication. The exponent uses a pseudo random number generator, which extends the coverage of the unit test.

### 2.3.16 booster.rs

This module holds the code related to booster functionality.

Modify\_booster enables the owner or operator to modify a booster. This includes the possibility to add a BoosterInfo to a seed with no such information, or to remove a BoosterInfo from a seed.

### 2.3.17 farmer\_seed.rs

The farmer seed module represents the farmer deposit staked for a seed.

We have the free\_amount, the locked\_amount and the x\_locked\_amount which is the power of the locked amount.

The unlock\_timestamp sets the time after which a stake can be unlocked without a slash penalty.

The data structure also holds the boost ratios for the farmer.

### 2.3.18 farmer.rs

The Farmer data structure keeps track of all of a farmer's seeds and of all the rewards that the farmer claimed.

Rewards for one seed are calculated over all farms in which the farmer has a stake in `pub fn internal_calc_farmer_claim.`

The farmer has a set seed power for that seed.

The amount of reward token for each farm is the product of the farmer seed power with the difference between the seed\_farm rps (reward per total seed power) and the farmer rps (reward per farmer seed power) :

$$\text{reward} = (\text{seed\_farm.rps} - \text{farmer\_rps}) * \text{farmer\_seed\_power}.$$

The first time this function is called, farmer\_rps is equal to zero and we have:

$$\begin{aligned} \text{reward} &= \text{seed\_farm.rps} * \text{farmer\_seed\_power} \\ &= (\text{reward}/\text{total\_seed\_power}) * \text{farmer\_seed\_power} \end{aligned}$$

We verify here that the farmer receives its relative seed power share of the reward.

The farmer reward per seed power is updated to the seed\_farm rps. If the function is called again before any new reward is deposited, no new reward will be claimed.

**Recommendation:** to improve understanding of the code, rename seed\_power to farmer\_seed\_power when computing reward claims.

## 2.4 Sequences

### 2.4.1 Creation of a new seed

Only operators or owner can create a new seed, through create\_seed.

The call fails if a seed already exists for this token. A new seed always starts with the default slash rate.

In case of success, the event SeedCreate is generated.

### 2.4.2 Creation of a new farm

Only operators or owner can create a new farm through create\_farm.

The farm is created if the seed already exists and if the maximum amount of farms for a seed has not been reached yet for that seed.

In case of success, the event FarmCreate is generated.



### 2.4.3 Cancel a farm

Only operators or owner can cancel a farm through `cancel_farm`, and only if it has not yet started.

### 2.4.4 Remove a farm

Only operators or owner can remove a farm from a seed using `remove_farm_from_seed`. This is only possible if the farm has ended.

### 2.4.5 Update a booster

Only operators or owner can update a booster through `modify_booster`.

A booster cannot affect itself and cannot affect more than

`MAX_NUM_SEEDS_PER_BOOSTER` (16) and cannot affect more than `config.max_num_farms_per_booster`.

## 2.5 Annex 1: unit tests

running 18 tests

```
test big_decimal::tests::test_compound_pow ... ok
test big_decimal::tests::test_compound_pow_precision ... ok
test big_decimal::tests::test_display ... ok
test big_decimal::tests::test_pow ... ok
test big_decimal::tests::test_simple_add ... ok
test big_decimal::tests::test_simple_div ... ok
test events::tests::event_farm_cancel ... ok
test events::tests::event_farm_create ... ok
test events::tests::event_reward_deposit ... ok
test events::tests::event_reward_withdraw ... ok
test events::tests::event_seed_create ... ok
test events::tests::event_seed_deposit ... ok
test events::tests::event_seed_free_to_lock ... ok
test events::tests::event_seed_unlock ... ok
test events::tests::event_seed_withdraw ... ok
test events::tests::event_seed_withdraw_lostfound ... ok
test events::tests::event_seed_withdraw_slashed ... ok
test big_decimal::tests::test_compound_pow_random ... ok
```

test result: **ok**. 18 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 11.71s

**Running** tests/test\_actions\_of\_farmer\_reward.rs

(/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_actions\_of\_farmer\_reward-f5580bf45883ac07)

running 2 tests

```
test test_claim_reward_by_seed ... ok
test test_withdraw_reward ... ok
```

test result: **ok**. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 9.22s

#### Running tests/test\_actions\_of\_farmer\_seed.rs

(/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_actions\_of\_farmer\_seed-0d0cbf48f9c4d568)

running 3 tests

test test\_force\_unlock ... **ok**  
 test test\_lock\_free\_seed ... **ok**  
 test test\_unlock\_and\_withdraw\_seed ... **ok**

test result: **ok**. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 8.35s

#### Running tests/test\_actions\_of\_seed.rs

(/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_actions\_of\_seed-d95fc9642301b91d)

running 4 tests

test test\_candle\_farm ... **ok**  
 test test\_create\_seed ... **ok**  
 test test\_remove\_farm\_from\_seed ... **ok**  
 test test\_create\_farm ... **ok**

test result: **ok**. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 30.81s

#### Running tests/test\_booster.rs (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_booster-6e72baae0a143ac2)

running 1 test

test test\_modify\_booster ... **ok**

test result: **ok**. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 28.64s

#### Running tests/test\_management.rs

(/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_management-37536fbe3c0efe82)

running 9 tests

test test\_modify\_default\_slash\_rate ... **ok**  
 test test\_modify\_locking\_policy ... **ok**  
 test test\_modify\_max\_farm\_num\_per\_seed ... **ok**  
 test test\_modify\_daily\_reward ... **ok**  
 test test\_modify\_seed\_min\_locking\_duration ... **ok**  
 test test\_modify\_seed\_min\_deposit ... **ok**  
 test test\_modify\_seed\_slash\_rate ... **ok**  
 test test\_return\_seed\_lostfound ... **ok**  
 test test\_withdraw\_seed\_slashed ... **ok**

test result: **ok**. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 13.45s

#### Running tests/test\_migrate.rs (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_migrate-902fa4c7b23710db)

running 1 test

test test\_update ... **ok**

test result: **ok**. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 4.69s

#### Running tests/test\_owner.rs (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_owner-afd48650f45ab3c8)

running 2 tests  
 test test\_set\_owner ... **ok**  
 test test\_operators ... **ok**

test result: **ok**. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 4.44s

**Running** tests/test\_scene.rs (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_scene-e1a76164c6ec6561)

running 5 tests  
 test test\_booster\_seed\_mutli\_farm ... **ok**  
 test test\_normal\_mutli\_farm ... **ok**  
 test test\_mutli\_seed\_with\_booster\_and\_normal ... **ok**  
 test test\_verify\_users\_reward\_total\_amount ... **ok**  
 test test\_normal\_seed\_single\_farm ... **ok**

test result: **ok**. 5 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 50.87s

**Running** tests/test\_storage.rs (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_storage-68cfa882c744f895)

running 1 test  
 test test\_storage ... **ok**

test result: **ok**. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 8.50s

**Running** tests/test\_token\_receiver.rs  
 (/home/ubuntu/src/near/boost-farm/target/debug/deps/test\_token\_receiver-38dc5eeda1bb4cdc)

running 3 tests  
 test test\_free ... **ok**  
 test test\_lock ... **ok**  
 test test\_reward ... **ok**

test result: **ok**. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 9.94s

**Doc-tests** boost-farming

running 0 tests

test result: **ok**. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

## 2.6 Annex 2: Code Coverage

|| Tested/Total Lines:  
 || contracts/boost-farming/src/actions\_of\_farmer\_reward.rs: 0/34  
 || contracts/boost-farming/src/actions\_of\_farmer\_seed.rs: 0/102  
 || contracts/boost-farming/src/actions\_of\_seed.rs: 0/59  
 || contracts/boost-farming/src/big\_decimal.rs: 110/151  
 || contracts/boost-farming/src/booster.rs: 0/40  
 || contracts/boost-farming/src/events.rs: 93/95  
 || contracts/boost-farming/src/farmer.rs: 0/62  
 || contracts/boost-farming/src/farmer\_seed.rs: 0/88  
 || contracts/boost-farming/src/legacy.rs: 0/6

```

|| contracts/boost-farming/src/lib.rs: 0/24
|| contracts/boost-farming/src/management.rs: 0/120
|| contracts/boost-farming/src/owner.rs: 0/19
|| contracts/boost-farming/src/seed.rs: 0/29
|| contracts/boost-farming/src/seed_farm.rs: 0/56
|| contracts/boost-farming/src/storage_impl.rs: 0/38
|| contracts/boost-farming/src/token_receiver.rs: 0/59
|| contracts/boost-farming/src/utils.rs: 6/32
|| contracts/boost-farming/src/view.rs: 0/95
|| contracts/boost-farming/tests/setup/booster.rs: 7/7
|| contracts/boost-farming/tests/setup/farmer_reward.rs: 9/9
|| contracts/boost-farming/tests/setup/farmer_seed.rs: 16/16
|| contracts/boost-farming/tests/setup/management.rs: 37/37
|| contracts/boost-farming/tests/setup/mod.rs: 60/62
|| contracts/boost-farming/tests/setup/owner.rs: 14/14
|| contracts/boost-farming/tests/setup/seed.rs: 18/18
|| contracts/boost-farming/tests/setup/storage_impl.rs: 13/13
|| contracts/boost-farming/tests/setup/token_receiver.rs: 28/35
|| contracts/boost-farming/tests/setup/tokens.rs: 35/35
|| contracts/boost-farming/tests/setup/users.rs: 9/9
|| contracts/boost-farming/tests/setup/utils.rs: 29/29
|| contracts/boost-farming/tests/setup/views.rs: 47/47
|| contracts/boost-farming/tests/test_actions_of_farmer_reward.rs: 88/88
|| contracts/boost-farming/tests/test_actions_of_farmer_seed.rs: 112/115
|| contracts/boost-farming/tests/test_actions_of_seed.rs: 122/122
|| contracts/boost-farming/tests/test_booster.rs: 55/55
|| contracts/boost-farming/tests/test_management.rs: 198/201
|| contracts/boost-farming/tests/test_migrate.rs: 7/7
|| contracts/boost-farming/tests/test_owner.rs: 27/27
|| contracts/boost-farming/tests/test_scene.rs: 893/893
|| contracts/boost-farming/tests/test_storage.rs: 52/52
|| contracts/boost-farming/tests/test_token_receiver.rs: 92/92
|| contracts/mock-ft/src/lib.rs: 28/36
|| contracts/mock-mft/src/lib.rs: 35/38
|| contracts/mock-mft/src/mft.rs: 33/89
||
69.83% coverage, 2273/3255 lines covered

```