BondHive Audit

September 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the BondHive project.

During this audit we found two critical issues, one high issue, two medium issues and two minor issues. Also, two enhancements were proposed. All the issues were resolved except for one of the medium ones which was mitigated. One of the enhancements was implemented.

Scope

The audited files are from the git repository located at https://github.com/Bond-Hive/soroban-contracts.git. The audit is based on the commit `f115457909194201576d1ca0c1134ac6314df4af`. Fixes were checked on commit `e0d41720a730089eca9006d30c6c03f20f90dc92`.

The scope for this audit includes and is limited to the following files:

- `bond_contract/src/lib.rs`: `bond_contract` implementation. See the Contract Descriptions section for details.
- `bond_contract/src/token.rs`: Client and trait for token contracts.
- `farm_contract/src/lib.rs`: `farm_contract` implementation. See the Contract Descriptions section for details.
- `farm_contract/src/token.rs`: Client and trait for token contracts[1].

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent two weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions

[1] They are called receipt tokens in the `farm_contract` implementation.

- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

While looking for vulnerabilities we used Scout Soroban[2] to aid our search.

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

Id	Title	Severity	Status
CR-01	Initialize bond_contract Again	Critical	Resolved
CR-02	Initialize farm_contract Again	Critical	Resolved
CR-03	Anyone Can Withdraw Unallocated Rewards	Critical	Resolved
HI-01	Locked Funds	High	Resolved
ME-01	Unbound Instance Storage	Medium	Resolved
ME-02	Funds Locked Through Overflow	Medium	Mitigated

[2] https://github.com/CoinFabrik/scout-soroban

Id	Title	Severity	Status
MI-01	Minted Shares Slippage in bond_contract	Minor	Resolved
MI-02	No Initialization Event	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Initialize bond_contract Again

Location:

- `bond_contract/src/lib.rs: 324-363`

Classification:

- CWE-284: Improper access control[3]

An attacker may invoke the `initialize` function of the `bond_contract`. Doing so will:

- change the contract admin.
- change the start and end time of the contract.
- change the token of the contract.
- change the shares token contract[4].
- change the quote period.
- change the treasury.
- change the min deposit.
- reset the total shares, the total deposit, the redemption available, and the current quote.

If the call of the `initialize` function is made after the `set_total_redemption` call has been made the contract funds are locked because the available redemption is set to zero on initialization.

Recommendation

Check in the `initialize` function that it has not run yet by checking if the admin has been set.

Status

Resolved. The `initialize` function now checks if the contract has already been initialized.

CR-02 Initialize farm_contract Again

Location:

- `farm_contract/src/lib.rs: 311-354`

[3] https://cwe.mitre.org/data/definitions/284.html
[4] This needs to be changed for the attack to work given that the `initialize` function initializes this contract and the currently used token checks to not get initialized twice.

Classification:

- CWE-284: Improper access control[5]

An attacker may invoke the `initialize` function of the `bond_contract`. Doing so will:

- change the burn wallet[6].
- change the shares token contract[7].
- change the contract's admin.
- change the reward tokens.
- change the maturity.
- reset the allocated rewards
- reset the pool counter.

After changing both the admin and resetting the allocated rewards, an attacker may transfer away all the reward funds in the contract.

Recommendation

Check in the `initialize` function that it has not run yet by checking if the admin has been set.

Status

Resolved. The `initialize` function now checks if the contract has already been initialized.

CR-03 Anyone Can Withdraw Unallocated Rewards

Location:

- `farm_contract/src/lib.rs: 632-682`

Classification:

- CWE-284: Improper access control[8]

In the `withdraw_unallocated_rewards` function of the `farm_contract` the admin account is taken from a parameter instead of being fetched from the storage in `farm_contract/src/lib.rs:636`, allowing an attacker to sign the transaction to retrieve all the rewards.

[5] https://cwe.mitre.org/data/definitions/284.html
[6] This address is set only on initialization. If the second rewards account is set to this address it signals that only the first reward token will be used.
[7] This needs to be changed for the attack to work given that the `initialize` function initializes this contract and the currently used token (which sits compiled in the `soroban_token_contract.wasm` file in the root directory of the repository) checks to not get initialized twice.
[8] https://cwe.mitre.org/data/definitions/284.html

Recommendation

Use the `get_admin` function to obtain the admin's address instead of receiving it as a parameter.

Status

Resolved. The `withdraw_unallocated_rewards` function now retrieves the admin address directly from storage using the `get_admin` function.

High Severity Issues

HI-01 Locked Funds

Location:

- `farm_contract/src/lib.rs: 480, 488, 514, 546`

In the `farm_contract`, if a user deposits some tokens and then transfers the receipt tokens obtained to another account, neither of the accounts can withdraw the deposited funds and/or its corresponding rewards.

Recommendation

Do not use receipt tokens in the `farm_contract`.

Status

Resolved. Receipt tokens have been removed.

Medium Severity Issues

ME-01 Unbound Instance Storage

Location:

- `farm_contract/src/lib.rs: 182-191, 193-201, 203-212, 214-221, 224-235`

Classification:

- CWE-770: Allocation of Resources Without Limits or Throttling[9]

In the `farm_contract`, the information stored in the `DataKey::UserMap` and `DataKey::PoolMap` instance slots contain a map with unbounded size. So when these

[9] https://cwe.mitre.org/data/definitions/770.html

maps grow, an extra cost is incurred on every interaction with the `farm_contract`, and if the required storage exceeds 64Kb it may lead to a denial of service.

Recommendation
Store each entry of each map on its own persistent storage slot.

Status
Resolved. Now each entry of each map is in its own persistent storage slot.

ME-02 Funds Locked Through Overflow

Location:
- `farm_contract/src/lib.rs: 430, 438, 531, 539`

Classification:
- CWE-190: Integer Overflow or Wraparound[10]

In the `farm_contract`, both the `deposit` and `withdraw` functions contain a possible overflow that may prevent a user from permanently depositing or withdrawing its funds.

For example see `farm_contract/src/lib.rs:531`.

```
(user_data.deposited * pool.reward_ratio1 * time_elapsed as i128) / 10i128.pow(DECIMALS)
```

The multiplication may overflow even if the final result fits in a `i128`.

Recommendation

To solve this issue, make sure that the multiplication will not overflow by restricting the max settings for pool reward ratios, maximum deposits for each user and the maximum maturity date.

Status

Mitigated. Max reward ratios are set on initialization now. Also the development team informed us that the maturity period will not extend more than 6 months.

Minor Severity Issues

MI-01 Minted Shares Slippage in bond_contract

Location:
- `bond_contract/src/lib.rs: 390-417`

[10] https://cwe.mitre.org/data/definitions/190.html

Classification:

- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')[11]

If the administrator changes the quote in the `bond_contract` while a deposit operation is made, by submitting those transactions simultaneously to the blockchain, the number of obtained shares in the deposit operation may vary depending on the ordering on which those operations are incorporated.

This issue's severity was lowered because only an admin may change the quote.

Recommendation

Add an additional parameter to the deposit function with the minimum number of shares to be obtained while depositing, allowing an EOA to mitigate the possible loss.

Status

Resolved. If the `current_quote` is different than the `expected_quote` then the transaction reverts.

MI-02 No Initialization Event

Location:

- `farm_contract/src/lib.rs: 311-354`

Classification:

- CWE-223: Omission of Security-relevant Information[12]

When initializing the `farm_contract` it writes state to the storage but no event is generated informing that change.

It is a good practice to emit events on all state changes, as it helps analyze the contract's behavior.

Recommendation

Log an event with all the state persisted in the initialization

Status

Resolved. The event emission was added.

[11] https://cwe.mitre.org/data/definitions/362.html
[12] https://cwe.mitre.org/data/definitions/223.html

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Id	Title	Status
EN-01	Stop Switches	Implemented
EN-02	Decouple Rewards	Not implemented

EN-01 Stop Switches

Neither the `bond_contract` or the `farm_contract` gives the admin the option to stop operations. This is particularly problematic in the `bond_contract` given that users doing deposits expect its deposit to be fulfilled but if there is a problem the admin cannot stop them from doing more deposits. In the `farm_contract` having the ability to stop operations would also be nice but it does not have the same impact as the reward funds are already stored in the contract when the deposit is made.

These stop switches can be leveraged in case of an attack, to restrict its impact.

Recommendation

Add configuration options, configured by the admin of the contract to prevent both deposit and withdrawals operations.

Status

Implemented. The stop switches were added.

EN-02 Decouple Rewards

Location:

- `farm_contract/src/lib.rs`

In the `farm_contract`, all the accumulated rewards for a user and pool are transferred to the user if a call to the withdraw function is made, irrespective of the number of shares withdrawn.

Recommendation

Decouple rewards withdrawing by adding a different function to extract the rewards.

Status

Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

bond_contract

The admin account needs to be trusted.

The funds to be withdrawn by the users are to be provided by the admin account via the `set_total_redemption` function after the bonds are purchased via the `deposit` function, so they need to trust the admin to properly fund the contract.

The treasury account can be changed at any time by the admin.

farm_contract

The admin account is significantly less powerful in this contract than in the bond contract, given that its code ensures that the rewards are present when depositing funds in this contract.

Upgrades

The analyzed contracts do not have any provisions for code upgrades.

Funds Flow Analysis

bond_contract

Bond Token

Bond tokens are:

- transferred from the `from` address to the treasury address in the `deposit` function.
- transferred from the contract to the `to` address in the `withdraw` function.
- transferred from the admin to the contract in the `set_total_redemption` function.

Shares Token

Shares tokens are:

- minted for the `from` address in the `deposit` function.
- transferred from the `to` address to the contract and then burned in the `withdraw` function.

farm_contract

Pool Tokens

Pool tokens are:

- transferred from the `depositor` to the contract in the `deposit` function.
- transferred from the contract to the `withdrawer` in the `withdraw` function.

Rewards Tokens

Rewards tokens are:

- transferred from the contract to the `withdrawer` in the `withdraw` function.
- transferred from the contract to the admin in the `withdraw_unallocated_rewards` function.

Receipt Tokens

Receipt tokens are:

- minted for the `depositor` in the `deposit` function.
- transferred from the `withdrawer` to the contract and then burned in the `withdraw` function.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

bond_contract

Admin

The account with this role can:

- set the quote via the `set_quote` function.
- set the total redemption via the `set_total_redemption` function.
- set the address of the treasury via the `set_treasury` function.

- transfer the admin rights to a new account via the `set_admin` function.

The initial admin is set in the contract initialization.

farm_contract

Admin

The account with this role can:

- create a new pool via the `create_pool` function.
- transfer the admin rights to a new account via the `set_admin` function.
- withdraw the unallocated rewards via the `withdraw_unallocated_rewards` function[13].

Token Minting

Both the `bond_contract` and the farm_contract use tokens to track the shares of the users [14]. It is implicit in their source code that only them mint new tokens. If another account mints them then they will not work properly as they internally track the number of tokens that they mint and burn[15].

Contract Descriptions

The development team provided us with the following description of the analyzed contracts

bond_contract

This smart contract represents a vault designed for issuing yield-bearing bond-like tokens, each with specific maturities and underlying assets, initially including ETH and BTC with September and December 2024 maturities. The primary purpose is to accept definable assets (initially USDC) and issue zero coupon bond-like instruments.

Key Functionalities

Asset Collection and Bond Issuance

Assets are collected and tokens are issued via smart contract interactions. An oracle, absent on Stellar for these delivery futures instruments, is simulated by manually writing quotes

[13] But see CR-03 Anyone Can Withdraw Unallocated Rewards.
[14] In the case of the `farm_contract` they are called receipt tokens.
[15] But see HI-01 Locked Funds.

which are valid for a set duration. The quote represents the difference between the spot and futures perpetual value (basis), for issuing tokens.

Treasury and CEX Interactions

Collected assets are sent to a treasury wallet, which then transfers them to a centralized exchange (CEX) where positions are entered.

Bond Redemption

At maturity, bondholders can redeem their tokens at face value, similar to zero-coupon bonds. Redemption values are predefined as $100 per token (but may vary and is not promised)

Administrative Functions

Admin Controls

Various administrative functions are implemented to manage the contract effectively. The admin has the ability to set quotes, manage treasury details, and handle redemptions.

Setting Total Redemption

This function determines the total funds available for token redemption, shared proportionally among token holders. In case of a shortfall, this mechanism ensures an equitable distribution of available funds.

Detailed Function Breakdown

Initialize

Sets up the contract with initial parameters like token addresses, administrative controls, and timing for bond maturity.

Quote Management

Due to the lack of a native oracle, quotes are manually set and have a limited validity period to mitigate the risk of stale pricing. The contract requires an active quote for transactions to proceed.

Token Handling

Minting and Burning

Bond tokens are minted when funds are deposited and burned upon redemption or withdrawal.

Deposit and Withdrawal

Users can deposit collateral to receive bond tokens or withdraw their investments, respectively. Withdrawals are tied to the bond's maturity and available redemption funds.

Admin Functions

Set Admin

Updates the administrator of the contract.

Set Treasury

Updates the treasury address, directing where funds should be sent within the ecosystem.

Set Total Redemption

Admin-controlled setting that determines the total amount available for redemption.

farm_contract

This smart contract is designed for the farming of rewards by holders of yield-bearing bond tokens. The contract facilitates the distribution of rewards according to emission rate and the amounts of bond tokens staked in various pools. The pools can have defined reward tokens, emission times, and can accommodate multiple pools with the same maturity.

Key Functionalities

Reward Distribution

Reward rates are predefined and distributed proportionally based on the amount of tokens staked by each user in the pools.

Rewards can consist of one or two different tokens, with emission durations clearly defined.

Token Deposits for Farming

Bond tokens issued by a separate bond contract are used as stakes for farming rewards.

Users can deposit these bond tokens into various pools to start accumulating rewards.

Pool Management

Administrators can create any number of pools, each possibly with different reward structures but sharing the same maturity.

Upon reaching maturity, unused rewards can be withdrawn by the admin.

Early Withdrawals

Users have the flexibility to withdraw their stakes before the pool's maturity. Upon withdrawal, users receive all accrued rewards up to that point, without any penalties.

Contract Specifications

Admin Functions

Include critical controls for managing reward tokens, creating and adjusting pools, and handling withdrawals of unallocated rewards after pool maturity.

Token Handling

Bond tokens are deposited into the farming contract to receive farming rewards. These tokens are tracked within the contract to ensure that rewards are distributed based on the amount staked.

Maturity and Rewards

Each pool has a defined maturity date. Rewards accumulate until this point, after which they can either be withdrawn by users or reclaimed by the admin if unallocated.

Withdrawal Mechanisms

Provides mechanisms for both regular withdrawals and early withdrawals, with the latter allowing users to exit the pool before maturity while still claiming their accumulated rewards.

Changelog

- 2024-09-02 – Initial report based on commit `f115457909194201576d1ca0c1134ac6314df4af`.
- 2024-09-30 – Check fixes on commit `e0d41720a730089eca9006d30c6c03f20f90dc92`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the BondHive project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.