BlockSec

Security Audit Report for Ref-Boost-Farm

Date: July 26, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2022.07.29
11:07:23 +08'00'

Contents

Report Manifest

Item	Description
Client	Ref Finance
Target	Ref-Boost-Farm

Version History

Version	Date	Description
1.0	July 26, 2022	First Release

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The repository that has been audited includes boost-farm [1].

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Ref-Boost-Farm	`Version 1`	`28dd78a29df1268d7eaea83c5d7db688b683711`
	`Version 2`	`fc0cada65253eb1d49bcf4855c92c4ff0791239c`

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **contracts/boost-farming** folder contract only. Specifically, the files covered in this audit include:

- actions_of_farmer_reward.rs
- actions_of_farmer_seed.rs
- actions_of_seed.rs
- big_decimal.rs
- booster.rs
- errors.rs
- events.rs
- farmer.rs
- farmer_seed.rs
- legacy.rs
- lib.rs
- management.rs
- owner.rs
- seed.rs
- seed_farm.rs
- storage_impl.rs
- token_receiver.rs
- utils.rs
- view.rs

[1] https://github.com/ref-finance/boost-farm

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Table 1.1: Vulnerability Severity Classification

Impact	*High*	High	Medium
	Low	Medium	Low
		High	*Low*
		Likelihood	

- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **five** potential issues. We also have **five** recommendations, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 5
- Recommendations: 5
- Notes: 0

ID	Severity	Description	Category	Status
1	Low	Lost of Farmer's Reward	Software Security	Fixed
2	Low	No Privileged Function for Withdrawal of Beneficiary	DeFi Security	Confirmed
3	Low	Improper Allowed State for System Configuration	DeFi Security	Confirmed
4	Low	User Controllable Reward with the Change of booster_info	DeFi Security	Confirmed
5	Low	Unfair Reward Distribution in Certain Situation	DeFi Security	Confirmed
6	-	Improved Sanity Checks for System Parameters	Recommendation	Fixed
7	-	Improved Sanity Checks When Removing Operators	Recommendation	Fixed
8	-	Potential Elastic Supply Token Problem	Recommendation	Confirmed
9	-	Improper Attached Gas	Recommendation	Fixed
10	-	Potential Centralization Problem	Recommendation	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Lost of Farmer's Reward

Status Fixed in Version 2

Introduced by Version 1

Description Function callback_post_withdraw_reward is used to handle the execution result for cross contract invocation (i.e., ft_tranfer). If function ft_tranfer fails, all the state changes inside the function withdraw_reward will be recovered. However, the existence of the account (farmer_id) is not checked in function callback_post_withdraw_reward, which may result in panic and the reward of the account cannot be recovered.

```
    #[private]
    pub fn callback_post_withdraw_reward(
        &mut self,
        token_id: AccountId,
        farmer_id: AccountId,
        amount: U128,
    ) {
        require!(
            env::promise_results_count() == 1,
            E001_PROMISE_RESULT_COUNT_INVALID
        );
        let amount: Balance = amount.into();
        match env::promise_result(0) {
            PromiseResult::NotReady => unreachable!(),
            PromiseResult::Successful(_) => {
                Event::RewardWithdraw {
                    farmer_id: &farmer_id,
                    token_id: &token_id,
                    withdraw_amount: &U128(amount),
                    success: true,
                }
                .emit();
            }
            PromiseResult::Failed => {
                // This reverts the changes from withdraw function.
                let mut farmer = self.internal_unwrap_farmer(&farmer_id);
                farmer.add_rewards(&HashMap::from([(token_id.clone(), amount)]));
                self.internal_set_farmer(&farmer_id, farmer);

                Event::RewardWithdraw {
                    farmer_id: &farmer_id,
                    token_id: &token_id,
                    withdraw_amount: &U128(amount),
                    success: false,
                }
                .emit();
            }
        }
    }
```

Listing 2.1: contracts/boost-farming/src/actions_of_famrer_reward.rs

Impact The reward of the unregistered account may be lost.

Suggestion I The existence of the `farmer_id` should be checked in the callback function.

2.2 DeFi Security

2.2.1 No Privileged Function for Withdrawal of Beneficiary

Status Confirmed

Introduced by `Version 1`

Description The reward of the farm will be assigned to the `beneficiary` under certain conditions, which are shown in function `finalize`. However, there is no function for the `beneficiary` to withdraw the reward.

```
    #[payable]
    pub fn remove_farm_from_seed(&mut self, farm_id: String) {
        assert_one_yocto();
        self.assert_owner();
        require!(self.data().state == RunningState::Running, E004_CONTRACT_PAUSED);

        let (seed_id, _) = parse_farm_id(&farm_id);
        let mut seed = self.internal_unwrap_seed(&seed_id);

        let VSeedFarm::Current(mut outdated_farm) = seed.farms.remove(&farm_id).expect(
            E401_FARM_NOT_EXIST);
        outdated_farm.finalize();

        self.data_mut().outdated_farms.insert(&farm_id, &outdated_farm.into());
        self.internal_set_seed(&seed_id, seed);
        self.data_mut().farm_count -= 1;
    }
```

Listing 2.2: contracts/boost-farming/src/actions_of_seed.rs

```
    pub fn finalize(&mut self) {
        require!(self.has_ended(), E405_FARM_NOT_ENDED);
        // remaining unclaimed rewards belongs to beneficiary
        self.amount_of_beneficiary =
            self.distributed_reward - self.claimed_reward;
}
```

Listing 2.3: contracts/boost-farming/src/seed_farm.rs

Impact The unclaimed reward will be locked in the contract.

Suggestion I It is suggested to implement the privileged functions to handle the locked rewards.

Feedback from the Project We will implement corresponding withdraw interface in the future.

2.2.2 Improper Allowed State for System Configuration

Status Confirmed

Introduced by `Version 1`

Description The administrator can set the system configurations (e.g., `daily_reward`) via the privileged functions (e.g., `modify_daily_reward`). However, these priviledged functions can only be invoked when the state of the contract is `Running`. Considering the emergency cases, it is suggested to allow the administrator to set the system configuration in the state of `Paused` as well.

```
    #[payable]
    pub fn modify_daily_reward(&mut self, farm_id: FarmId, daily_reward: U128) {
        assert_one_yocto();
        require!(self.is_owner_or_operators(), E002_NOT_ALLOWED);
        require!(self.data().state == RunningState::Running, E004_CONTRACT_PAUSED);

        let (seed_id, _) = parse_farm_id(&farm_id);
        let mut seed = self.internal_unwrap_seed(&seed_id);

        let VSeedFarm::Current(seed_farm) = seed.farms.get_mut(&farm_id).expect(E401_FARM_NOT_EXIST
            );
        seed_farm.terms.daily_reward = daily_reward.0;

        self.internal_set_seed(&seed_id, seed);
    }
```

Listing 2.4: contracts/boost-farming/src/management.rs

Impact The system configuration can be used to patch the vulnerabilities of the contract. In this case, setting the system configurations in `Running` state can leave the contract under the risk of attack.

Suggestion I Allow the administrator to set the system configurations in the state of `Paused` as well.

Feedback from the Project The contract state PAUSE is for emergency only, the contract state is expected to be untouched then. And those manage interfaces like `modify_daily_reward` are designed to be safe to execute when contract is in RUNNING state.

2.2.3 User Controllable Reward with the Change of booster_info

Status Confirmed

Introduced by `Version 1`

Description Function `modify_booster` can change the `booster_info` for corresponding `booster_id`, which can affect the value of the `seed_power`. However, the new `seed_power`, which can influence the amount of the reward, will not be updated until the user claims the reward.

```
    #[payable]
    pub fn modify_booster(&mut self, booster_id: SeedId, booster_info: BoosterInfo) {
        assert_one_yocto();
        require!(self.is_owner_or_operators(), E002_NOT_ALLOWED);
        require!(self.data().state == RunningState::Running, E004_CONTRACT_PAUSED);
        require!(self.internal_get_seed(&booster_id).is_some(), E301_SEED_NOT_EXIST);
        booster_info.assert_valid(&booster_id);

        let mut config = self.data().config.get().unwrap();
        require!(self.affected_farm_count(&booster_info) <= config.max_num_farms_per_booster,
            E203_EXCEED_FARM_NUM_IN_BOOST);

        config.booster_seeds.insert(booster_id.clone(), booster_info);
        self.data_mut().config.set(&config);
    }
```

Listing 2.5: contracts/boost-farming/src/booster.rs

Impact The time of claiming reward can influence the amount of received reward when the `booster_info` is changed.

Suggestion I Notify the affected users to claim the reward timely once the `booster_info` is updated.

Feedback from the Project Client side would guide users to fresh their reward state if booster policy changes. And in real production env, there is a very rare probability to modify booster policies.

2.2.4 Unfair Reward Distribution in Certain Situations

Status Confirmed

Introduced by `Version 1`

Description The reward to be distributed in each farm is deposited by administrator. If the administrator does not deposit the reward in time (e.g., after the start time of the farm). The users may receive additional reward.

For example, one farm is created and will be started on day N. The distribution time (i.e., distributed_at) will also be set as day N when the farm is created. User A stakes on day N+9 while the reward is deposited on day N+10, which means the reward is not deposited in time. Note that the status of the farm is Pending on day N+9 and is Running on day N+10 after the reward is deposited. In this case, the reward of the farm will not be updated on day N+9 as the farm is not running. After that, when user B stakes on day N+11, the farm will be updated and the reward will be calculated based on the differences between the distribution time (day N) and the current time (day N+11). In this case, user A can receive the reward for staking 11 days while the real staking time for user A is only 2 days.

```
pub fn add_reward(&mut self, reward_token: &AccountId, amount: Balance) -> (Balance, u32) {
        require!(self.terms.reward_token == reward_token.clone(), E404_UNMATCHED_REWARD_TOKEN);
        if self.terms.start_at == 0 {
            self.terms.start_at = nano_to_sec(env::block_timestamp());
            self.distributed_at = env::block_timestamp();
        }
        self.total_reward += amount;
        (self.total_reward, self.terms.start_at)
    }
```

Listing 2.6: contracts/boost-farming/src/seed_farm.rs

```
    pub fn update(&mut self, seed_power: Balance) {
        let block_ts = env::block_timestamp();

        self.internal_update_status(block_ts);

        if block_ts <= self.distributed_at {
            // already updated, skip
            return;
        }

        match self.status.as_ref().unwrap() {
            FarmStatus::Ended => {
                self.distributed_at = block_ts;
            },
            FarmStatus::Running => {
                let reward = std::cmp::min(
                    self.total_reward - self.distributed_reward,
                    u128_ratio(
                        self.terms.daily_reward,
                        u128::from(block_ts - self.distributed_at),
                        u128::from(NANOS_PER_DAY),
                    ),
                );
                self.distributed_reward += reward;
                if seed_power > 0 {
                    self.rps = self.rps + BigDecimal::from(reward).div_u128(seed_power);
                } else {
```

```
                self.amount_of_beneficiary += reward;
            }
            self.distributed_at = block_ts;
            self.internal_update_status(block_ts);
        },
        _ => {},
    }
}
```

Listing 2.7: contracts/boost-farming/src/seed_farm.rs

Impact User can receive additional reward under certain situations.

Suggestion I Add reward in time.

Feedback from the Project Noted, Although it is very rare to happen, we would consider to improve related logic in the future.

2.3 Additional Recommendation

2.3.1 Improved Sanity Checks for System Parameters

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `slash_rate` may be larger than the `BP_DENOM`. In this case, the amount of the slashed seed is larger than the original locked amount, which is unfair for users.

```
#[payable]
pub fn modify_default_slash_rate(&mut self, slash_rate: u32) {
    assert_one_yocto();
    require!(self.is_owner_or_operators(), E002_NOT_ALLOWED);
    require!(self.data().state == RunningState::Running, E004_CONTRACT_PAUSED);

    let mut config = self.data().config.get().unwrap();
    config.seed_slash_rate = slash_rate;
    self.data_mut().config.set(&config);
}
```

Listing 2.8: contracts/boost-farming/src/management.rs

Suggestion I Add the check to make sure the `slash_rate` is smaller than the `BP_DENOM`.

2.3.2 Improved Sanity Checks When Removing Operators

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `owner` of the protocol can remove `operators` via the function `remove_operators`. However, the existence of `operators` is not checked. In this case, if the `operator` does not exist, the program will not panic, which may mislead the `owner` and bring unexpected impact.

```
    #[payable]
    pub fn remove_operators(&mut self, operators: Vec<AccountId>) {
        assert_one_yocto();
        self.assert_owner();
        for operator in operators {
            self.data_mut().operators.remove(&operator);
        }
    }
```

Listing 2.9: contracts/boost-farming/src/owner.rs

Suggestion I Check the return value of function `remove`.

2.3.3 Potential Elastic Supply Token Problem

Status Confirmed

Introduced by `Version 1`

Description Elastic supply tokens could dynamically adjust their price, supply, user's balance, etc. For example, inflation tokens, deflation tokens, rebasing tokens, and so forth. In the current implementation of protocol, elastic supply tokens are not supported. If the token is a deflation token, there will be a difference between the recorded amount of transferred tokens to this smart contract (as a parameter of function `ft_on_ transfer`) and the actual number of transferred tokens (the token smart contract itself). That's because of a small number of tokens will be burned by the token smart contract.

This inconsistency can lead to security impacts for the operations based on the transferred amount of tokens instead of the actual received amount of tokens.

Suggestion I Do not add elastic supply tokens into the whitelist.

Feedback from the Project Noted, we would be very careful on tokens when creating farms.

2.3.4 Improper Attached Gas

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There is no check on whether `attached_gas` is enough for executing function `migrate`.

```
#[no_mangle]
    pub fn upgrade() {
        env::setup_panic_hook();
        let contract: Contract = env::state_read().expect("ERR_CONTRACT_IS_NOT_INITIALIZED");
        contract.assert_owner();
        let current_id = env::current_account_id().as_bytes().to_vec();
        let method_name = "migrate".as_bytes().to_vec();
        unsafe {
            // Load input (wasm code) into register 0.
            sys::input(0);
            // Create batch action promise for the current contract ID
            let promise_id =
                sys::promise_batch_create(current_id.len() as _, current_id.as_ptr() as _);
            // 1st action in the Tx: "deploy contract" (code is taken from register 0)
            sys::promise_batch_action_deploy_contract(promise_id, u64::MAX as _, 0);
            // 2nd action in the Tx: call this_contract.migrate() with remaining gas
            let attached_gas = env::prepaid_gas() - env::used_gas() - GAS_FOR_MIGRATE_CALL;
            sys::promise_batch_action_function_call(
                promise_id,
                method_name.len() as _,
                method_name.as_ptr() as _,
                0 as _,
                0 as _,
                0 as _,
                attached_gas.0,
            );
        }
    }
```

Listing 2.10: contracts/boost-farming/src/owner.rs

Suggestion I Check whether `attached_gas` is larger than a specified value.

2.3.5 Potential Centralization Problem

Status Confirmed

Introduced by `Version 1`

Description This project has potential centralization problems. The project owner needs to ensure the security of the private key of `ContractData.owner_id` and use a multi-signature scheme to reduce the risk of single-point failure.

Suggestion I It is recommended to introduce a decentralization design in the contract, such as a multi-signature or a public DAO.

Feedback from the Project The owner of this contract would be a DAO.