Security Audit Report for Meta Pool

Date: Jan 17, 2022

Version: 1.0

Contact: contact@blocksecteam.com

Contents

Report Manifest

Item	Description
Client	Meta Pool
Target	Meta Pool

Version History

Version	Date	Description
1.0	Jan 17, 2022	First Release

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

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

The audit scope includes the contract under the directory metapool in the Meta Staking Pool repository [1]. Note the contract under the directory meta-token is not in the audit scope for this report.

The auditing process is iterative. Specifically, we will further audit the commits that fix the founding issues. If there are new issues, we will continue this process. Thus, there are multiple commit SHA values referred in this report. The commit SHA values before and after the audit are shown in the following.

Before and during the audit

Contract Name	Stage	Commit SHA
Meta Pool	Initial	`1739b8782d88ba2793de3f02ef7fe99a7eacee25`

After

Project	Commit SHA
Meta Pool	`2339908956bdded5828c4a0abd3037ac1395b04e`

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 do 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] https://github.com/Narwallets/meta-pool/

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]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

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

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

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

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

Chapter 2 Findings

In total, we find 5 potential issues in the smart contract. We also have 5 recommendation, as follows:

- High Risk: 0
- Medium Risk: 1
- Low Risk: 3
- Recommendations: 5

The details are provided in the following sections.

ID	Severity	Description	Category	Status
1	Middle	*Missing check on the total weight of all the staking pools*	DeFi Security	Confirmed
2	Low	*Conflicts between account roles are not fully considered*	DeFi Security	Fixed
3	Low	*Missing check on the conflicts while setting account roles*	DeFi Security	Fixed
4	Low	*Account* `treasury_account` *cannot be read or modified*	DeFi Security	Fixed
5	-	*Function* `assert_callback_calling` *can be replaced by* `#[private]`	Recommendation	Acknowledged
6	-	*Unused macro is found*	Recommendation	Fixed
7	-	*Dead code is found*	Recommendation	Fixed
8	-	*Inconsistent implementation between function* `realize_meta_massive` *and* `realize_meta`	Recommendation	Confirmed
9	-	*Function* `get_staking_pool_list` *may not work*	Recommendation	Confirmed

2.1 DeFi Security

2.1.1 Missing check on the total weight of all the staking pools

Status Confirmed.

Description This issue is introduced in or before the initial commit.

```rust
#[derive(Default, BorshDeserialize, BorshSerialize)]
pub struct StakingPoolInfo {
    pub account_id: AccountId,

    //how much of the meta-pool must be staked in this pool
    //0=> do not stake, only unstake
    //100 => 1% , 250=>2.5%, etc. -- max: 10000=>100%
    pub weight_basis_points: u16,
```

Listing 2.1: staking_pools.rs

The weight_basis_points of a staking pool is set or modified by the owner with function set_staking_pool_weight in owner.rs:

```rust
        ///update existing weight_basis_points
    pub fn set_staking_pool_weight(&mut self, inx: u16, weight_basis_points: u16) {
        self.assert_operator_or_owner();

        let sp = &mut self.staking_pools[inx as usize];
        if sp.busy_lock {
            panic!("sp is busy")
        }
        // max is 50% for a single pool
        assert!(weight_basis_points < 5_000);
        // TODO: If ‘weight_basis_points‘ is invalid, the owner can break the contract.
        //    Ideally, the owner shouldn’t have any power to break the contract and instead
        //    should only manipulate the pools with verification that it’s a real pool, but it’s
        //    difficult to enforce.
        // option: store "score" for each validator & compute weight_basis_points as score*10_000/
            total_score
        // by doing that there’s no "invalid" score. Note: In order to do that, we should keep
            total_score on contract state
        sp.weight_basis_points = weight_basis_points;
    }
```

Listing 2.2: owner.rs

Impact The total weight_basis_points of staking pools may exceed 100% and starvation may occur in the lightweight staking pool when we operating on the distribute_staking.

Suggestion I Set an owner function in the contract that can set all stacking Pools’ weights at once and check at the end that the sum(weight_basis_points)==100%.

2.1.2 Conflicts between account roles are not fully considered

Status Fixed.

Description This issue is introduced in or before the initial commit. In the init function, there is no check on whether `treasury_account_id` equals to the `DEVELOPERS_ACCOUNT_ID` .

```rust
    #[init]
    pub fn new(
        owner_account_id: AccountId,
        treasury_account_id: AccountId,
        operator_account_id: AccountId,
        meta_token_account_id: AccountId,
    ) -> Self {
        assert!(!env::state_exists(), "The contract is already initialized");

        //all accounts must be different
        // not all combinations tested, we assume the owner deploying the contract knows that
            accounts must be different
        // it does not make sense to burn fees checking all possible combinations
        assert!(&owner_account_id != &treasury_account_id);
        assert!(&owner_account_id != &DEVELOPERS_ACCOUNT_ID);
        assert!(&operator_account_id != &owner_account_id);
        assert!(&operator_account_id != &DEVELOPERS_ACCOUNT_ID);
        assert!(&treasury_account_id != &operator_account_id);
```

Listing 2.3: lib.rs

Impact If `DEVELOPERS_ACCOUNT_ID` equals to `treasury_account_id`, the reward belonging to the treasury will be claimed by the developer.

Suggestion I We can write a separate function to check whether there are repeated account IDs. In the function `new` and the other functions that may change one of the accounts, we should invoke this function to ensure that there are no repeated account IDs.

2.1.3 Missing check on the conflicts while setting account roles

Status Fixed.

Description This issue is introduced in or before the initial commit. The owner can change the `operator_account_id` and `owner_id` by invoking such functions below. However, there is no check on whether the new `operator_account_id` equals the other IDs (e.g., `DEVELOPER_ACCOUNT_ID`), resulting in repeated account IDs.

```rust
    pub fn set_operator_account_id(&mut self, account_id: AccountId) {
        assert!(env::is_valid_account_id(account_id.as_bytes()));
        self.assert_owner_calling();
        self.operator_account_id = account_id;
    }
    pub fn set_owner_id(&mut self, owner_id: AccountId) {
        assert!(env::is_valid_account_id(owner_id.as_bytes()));
        self.assert_owner_calling();
        self.owner_account_id = owner_id.into();
    }
```

Listing 2.4: owner.rs

Impact The repeated account IDs can result in the same impact in issue 2.1.3

Suggestion I See suggestion for issue 2.1.3.

2.1.4 Account `treasury_account` cannot be read or modified

Status Fixed.

Description This issue is introduced in or before the initial commit. The `treasury_account_id` can not be changed after the contract is deployed and initialized.

Impact: N/A

Suggestion I Add the functions for read and modify the `treasury_account_id`.

2.2 Additional Recommendation

2.2.1 Function `assert_callback_calling` can be replaced by `#[private]`

Status Acknowledged.

Description This issue is introduced in or before the initial commit. We can replace function `assert_callback_calling()` in `metapool/src/utils.rs` by the macro `#[private]` provided by the near-sdk-rs.

```
pub fn assert_callback_calling() {
    assert_eq!(env::predecessor_account_id(), env::current_account_id());
}
```

Listing 2.5: utils.rs

Suggestion I Use macro `#[private]` instead of `assert_callback_calling()`.

Feedback from the project I would not recommend this, because the word #[private] conflicts with the pub fn right below. #[private] was a lousy choice from the NEAR team, and I prefer the code be readable. It should be called #[callback-only] to describe exactly what the macro is doing. The fn is actually public and exported in the WASM. Using the #[private] macro and requiring it to be a pub fn exported in the WASM only obscures the control being performed for new programmers and can lead to bugs in the future.

2.2.2 Unused macro is found

Status Fixed.

Description This issue is introduced in or before the initial commit. `#[payable]` is not required in function `set_reward_fee` because it does not require additional attached deposits.

```
    // idem previous function but in basis_points
    #[payable]
    pub fn set_reward_fee(&mut self, basis_points: u16) {
        self.assert_owner_calling();
        assert!(basis_points < 1000); // less than 10%
                                      // DEVELOPERS_REWARDS_FEE_BASIS_POINTS is included
        self.operator_rewards_fee_basis_points =
            basis_points.saturating_sub(DEVELOPERS_REWARDS_FEE_BASIS_POINTS);
    }
```

Listing 2.6: lib.rs

Suggestion I Remove macro `#[payable]` .

2.2.3 Dead code is found

Status Fixed.

Description This issue is introduced in or before the initial commit. Function `between` is not used.

```
#[inline]
pub fn between(value: u128, from: u128, to: u128) -> bool {
    value > from && value < to
}
```

Listing 2.7: utils.rs

Suggestion I The function is not used and can be removed.

2.2.4 Inconsistent implementation between function `realize_meta_massive` and `realize_meta`

Status Confirmed.

Description This issue is introduced in or before the initial commit. Function `realize_meta_massive` is used to realize meta for multiple users while `realize_meta` is used for one user. However, `realize_meta_massive` adds an additional check (line 849) on updating the account.

```
827 #[inline]
828     //------------------
829     // REALIZE META
830     //------------------
831     /// massive convert $META from virtual to secure. IF multipliers are changed, virtual meta can
            decrease, this fn realizes current meta to not suffer loses
832     /// for all accounts from index to index+limit
833     pub fn realize_meta_massive(&mut self, from_index: u64, limit: u64) {
834         for inx in
835             from_index..std::cmp::min(from_index + limit, self.accounts.keys_as_vector().len())
836         {
837             let account_id = &self.accounts.keys_as_vector().get(inx).unwrap();
838             if account_id == NSLP_INTERNAL_ACCOUNT {
839                 continue;
840             }
841             let mut acc = self.internal_get_account(&account_id);
842             let prev_meta = acc.realized_meta;
843 
844             acc.stake_realize_meta(self);
845             //get NSLP account
846             let nslp_account = self.internal_get_nslp_account();
847             //realize and mint meta from LP rewards
848             acc.nslp_realize_meta(&nslp_account, self);
849             if prev_meta != acc.realized_meta {
850                 self.internal_update_account(&account_id, &acc);
851             }
852         }
853     }
854 
```

```rust
pub fn realize_meta(&mut self, account_id: String) {
    let mut acc = self.internal_get_account(&account_id);

    //realize and mint $META from staking rewards
    acc.stake_realize_meta(self);

    //get NSLP account
    let nslp_account = self.internal_get_nslp_account();
    //realize and mint meta from LP rewards
    acc.nslp_realize_meta(&nslp_account, self);

    self.internal_update_account(&account_id, &acc);
}
```

Listing 2.8: lib.rs

Suggestion I Unify the implementation of these two functions.

2.2.5 Function `get_staking_pool_list` may not work

Status Confirmed.

Description This issue is introduced in or before the initial commit. The gas may not be enough for a transaction invoking function `get_staking_pool_list` due to huge number of stacking pools.

```rust
//---------------------------------
// staking-pools-list (SPL) management
//---------------------------------

/// get the current list of pools
pub fn get_staking_pool_list(&self) -> Vec<StakingPoolJSONInfo> {
    let mut result = Vec::with_capacity(self.staking_pools.len());
    for inx in 0..self.staking_pools.len() {
        let elem = &self.staking_pools[inx];
        result.push(StakingPoolJSONInfo {
            inx: inx as u16,
            account_id: elem.account_id.clone(),
            weight_basis_points: elem.weight_basis_points,
            staked: elem.staked.into(),
            unstaked: elem.unstaked.into(),
            last_asked_rewards_epoch_height: elem.last_asked_rewards_epoch_height.into(),
            unstaked_requested_epoch_height: elem.unstk_req_epoch_height.into(),
            busy_lock: elem.busy_lock,
        })
    }
    return result;
}
```

Listing 2.9: owner.rs

Suggestion I Add `from_index` and `end_index` as parameters in this function.