Security Audit Report for Lockup Stake Metapool

Date: August 22nd, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2023.08.29 11:43:52 +08'00'

Contents

Report Manifest

Item	Description
Client	Metapool
Target	Lockup Stake Metapool

Version History

Version	Date	Description
1.0	August 22nd, 2023	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 the **Lockup Stake Metapool** contract [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 (`Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Lockup Stake Metapool	`Version 1`	`faa0212181183d9bd69d63835cfcf1e226af663b`
	`Version 2`	`4643a70b9577dec66b68278b1c59039ea11af66e`

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **lockup-stake/src** folder contract only. Specifically, the file covered in this audit include:

- account.rs
- internal.rs
- owner.rs
- ping.rs
- staking.rs
- utils.rs
- views.rs

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.

[1] https://github.com/Meta-Pool/lockup-stake-metapool

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
- Permission management
- 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.

Table 1.1: Vulnerability Severity Classification

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

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.
- **Fixed** The item 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 **one** potential issues. We also **one** notes as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Recommendations: 0
- Notes: 1

ID	Severity	Description	Category	Status
1	Low	Lack of Check on the Accounts of the Contract	Defi Security	Fixed
2	-	Timely Invocation of Function ping to Maintain Accurate share_near_price	Notes	Confirmed

The details are provided in the following sections.

2.1 Defi Security

2.1.1 Lack of Check on the Accounts of the Contract

Severity Low

Status Fixed in Version 2

Introduced by Version 1

Description According to the design, only users whose account IDs end with .lockup.near will interact with this contract. However, the current implementation allows normal users (account IDs that do not end with .lockup.near) to interact with the contract. This can result in two issues.

First, the function perform_withdraw() assumes that the withdrawal from Metapool will precisely match the user-specified amount. Consequently, the callback function after_metapool_withdraw_to_lockup() directly subtracts the specified amount from the account.unstaked_in_metapool. However, Metapool does not guarantee that the actual withdrawn amount will align with the normal user's request. Second, the contract utilizes Metapool's function stake_for_lockup(), unstake_from_lockup_shares(), and withdraw_to_lockup() to facilitate staking, unstaking, and withdrawal actions on behalf of users. Unfortunately, Metapool does not differentiate between funds deposited via this contract and funds deposited directly into Metapool. Consequently, NEAR staked via this contract by normal users can be unstaked and withdrawn directly from Metapool. These discrepancies may result in inaccurate values for account.unstaked_in_metapool, account.stake_shares, and account.unstaked_available_epoch_height.

```rust
    #[payable]
    pub fn deposit_and_stake(&mut self) -> Promise {
        let account_id = env::predecessor_account_id();
        let amount = env::attached_deposit();

        // we're managing lockup.accounts, keep a sane minimum
        assert!(amount >= 10 * ONE_NEAR, "minimum deposit amount is 10 NEAR");

        // avoiding re-entry
        self.set_account_busy_flag_or_panic(&account_id);
        // call meta pool to stake
        ext_metapool::stake_for_lockup(
            account_id.to_string(),
            //---
            self.meta_pool_contract_id.clone(),
            amount, // send the NEAR
            Gas(META_POOL_DEPOSIT_AND_STAKE_GAS),
            )
        .then(ext_self::after_stake_for_lockup(
            account_id,
            amount.into(),
            //---
            env::current_account_id(),
            0,
            Gas(AFTER_STAKE_FOR_LOCKUP_GAS),
        ))
    }
```

Listing 2.1: lockup-stake/src/staking.rs

```
    pub fn unstake_all(&mut self) -> Promise {
        let account_id = env::predecessor_account_id();
        let account = self.internal_get_account(&account_id);
        self.inner_unstake_shares(&account_id, account.stake_shares)
    }

    /// Unstakes the given amount (in NEARs) from the inner account of the predecessor.
    /// The inner account should have enough staked balance.
    /// The new total unstaked balance will be available for withdrawal in four epochs.
    /// given that the share price increases with staking rewards, it is possible that final
         amount
    /// withdrawn could be higher because of the inclusion of new staking rewards
    /// (the amount could only be higher, not lower)
    pub fn unstake(&mut self, amount: U128) -> Promise {
        let amount: Balance = amount.into();
        let shares = mul_div(amount, ONE_NEAR, self.share_near_price);
        self.inner_unstake_shares(&env::predecessor_account_id(), shares)
    }
```

Listing 2.2: lockup-stake/src/staking.rs

```
    pub fn withdraw_all(&mut self) -> Promise {
        let account_id = env::predecessor_account_id();
        let account = self.internal_get_account(&account_id);
        self.perform_withdraw(&account_id, account.unstaked_in_metapool)
    }

    /// Withdraws the non staked balance for given account.
    /// It's only allowed if the ‘unstake‘ action was not performed in the four most recent epochs
         .
    pub fn withdraw(&mut self, amount: U128) -> Promise {
        self.perform_withdraw(&env::predecessor_account_id(), amount.into())
    }
```

Listing 2.3: lockup-stake/src/staking.rs

Impact The state of the contract can be incorrect.

Suggestion Ensure that only users whose account IDs end with `.lockup.near` can interact with this contract.

2.2 Notes

2.2.1 Timely Invocation of Function ping to Maintain Accurate share_near_price

Status Confirmed

Introduced by `Version 1`

Description The function `ping()` is implemented to retrieve `share_near_price` and `meta_pool_fee_bp` from `Metapool`. Users can use the function `unstake()` to withdraw a certain amount of `NEAR` that they have staked in the `Metapool`, which should be calculated based on the current `share_near_price`. As a result, it is crucial for the project team to promptly invoke the function `ping()` to ensure that the `share_near_price` is up-to-date. Failing to do so may result in users unintentionally unstaking more shares than they intended.

Feedback from the Project `ping()` is automatically called from the same bot that monitor validators performance and cranks the main `Metapool` contract.