Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.06.06, the SlowMist security team received the team's security audit application for OAK, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Reordering Vulnerability	Passed

NO.	Audit Items	Result
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Block data Dependence Vulnerability	Passed
6	Explicit Visibility of Functions Audit	Passed
7	Arithmetic Accuracy Deviation Vulnerability	Passed
8	Malicious Event Log Audit	Some Risks
9	Others	Some Risks
10	State Consistency Audit	Passed
11	Failure Rollback Audit	Passed
12	Unit Test Audit	Passed
13	Integer Overflow Audit	Some Risks
14	Parameter Verification Audit	Passed
15	Error Unhandle Audit	Passed
16	Boundary Check Audit	Passed
17	Weights Audit	Passed
18	Macros Audit	Passed

3 Project Overview

3.1 Project Introduction

Project source code repository:

https://github.com/OAK-Foundation/OAK-blockchain

Module:

pallets/automation-time/src/*.rs

Commit:

643342e936bbc821d2fb91be69872e4fcecd2273

Project address:

https://github.com/OAK-Foundation/moonbeam

pallets/parachain-staking/src/*.rs

Commit:

15b1a1f62483aa6babe1412d3ea1f18770a6b2a4

Review Commit:

https://github.com/OAK-Foundation/OAK-blockchain/commit/cba1acd6961fce877cef95c6b6a198ea8b415a0f

3.2 Coverage

Target Code and Revision:

https://github.com/OAK-Foundation/OAK-blockchain

3.3 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Calculate inaccurate risk	Integer Overflow Audit	Suggestion	Fixed
N2	User balance is not checked	Others	Suggestion	Ignored

NO	Title	Category	Level	Status
N3	Return Value Not Checked	Others	Low	Ignored
N4	Calculate inaccurate risk	Integer Overflow Audit	Suggestion	Ignored
N5	Missing logic	Others	Low	Ignored
N6	Missing error message	Malicious Event Log Audit	Suggestion	Ignored

4 Findings

4.1 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

automation-time				
Function Name	Parameter verification	Overflow	Authority	Weight
schedule_notify_task	4/4	ok	ensure_signed	133_327_009+
schedule_native_transfer_task	5/5	ok	ensure_signed	87_915_009+
cancel_task	2/2	ok	ensure_signed	1_192_000_052+
force_cancel_task	2/2	ok	ensure_root	1_203_000_052+

parachain-staking				
Function Name	Parameter verification	Overflow	Authority	Weight

parachain-staking				
set_staking_expectations	2/2	ok	ensure_origin	18_965_002+
set_inflation	2/2	ok	ensure_origin	54_552_002+
set_parachain_bond_account	2/2	ok	ensure_origin	18_127_002+
set_parachain_bond_reserve_percent	2/2	ok	ensure_origin	17_902_002+
set_total_selected	2/2	ok	ensure_root	20_823_002+
set_collator_commission	2/2	ok	ensure_root	17_257_002+
set_blocks_per_round	2/2	ok	ensure_root	59_571_002+
join_candidates	3/3	ok	ensure_root	65_829_011+
schedule_leave_candidates	2/2	ok	ensure_signed	63_608_004+
execute_leave_candidates	3/3	ok	ensure_signed	33_442_012+
cancel_leave_candidates	2/2	ok	ensure_signed	62_474_004+
go_offline	1/1	ok	ensure_signed	28_328_004+
go_online	1/1	ok	ensure_signed	28_177_004+
candidate_bond_more	2/2	ok	ensure_signed	28_177_008+
schedule_candidate_bond_less	2/2	ok	ensure_signed	26_339_002+
execute_candidate_bond_less	2/2	ok	ensure_signed	56_074_008+
cancel_candidate_bond_less	1/1	ok	ensure_signed	23_146_002+

parachain-staking				
delegate	5/5	ok	ensure_signed	114_718_012+
schedule_leave_delegators	1/1	ok	ensure_signed	27_072_002+
execute_leave_delegators	3/3	ok	ensure_signed	33_909_008+
cancel_leave_delegators	1/1	ok	ensure_signed	24_002_002+
schedule_revoke_delegation	2/2	ok	ensure_signed	32_552_004+
delegator_bond_more	3/3	ok	ensure_signed	63_313_012+
schedule_delegator_bond_less	3/3	ok	ensure_signed	32_393_004+
execute_delegation_request	3/3	ok	ensure_signed	76_987_014+
cancel_delegation_request	3/3	ok	ensure_signed	36_326_004+

4.2 Vulnerability Summary

[N1] [Suggestion] Calculate inaccurate risk

Category: Integer Overflow Audit

Content

- pallets/automation-time/src/lib.rs

There are some risks of value overflow.

`saturating_mul`, `saturating_sub`, `saturating_add` and `+-*/, +=, -=`

saturating at the numeric bounds instead of overflowing, The returned result is inaccurate.

Solution

Use `checked_add/checked_sub/checked_mul/checked_div` instead of `saturating_add/saturating_sub/saturating_mul/saturating_div` and `+-*/, +=, -=`.

Status

Fixed

[N2] [Suggestion] User balance is not checked

Category: Others

Content

- pallets/automation-time/src/lib.rs

The amount transferred by the user is not compared with the user's balance here, and the user may not have enough NativeToken.

```
pub fn schedule_native_transfer_task(
  origin: OriginFor<T>,
  provided_id: Vec<u8>,
  execution_times: Vec<UnixTime>,
  recipient_id: T::AccountId,
  #[pallet::compact] amount: BalanceOf<T>,
) -> DispatchResult {
  let who = ensure_signed(origin)?;

  // check for greater than existential deposit
  if amount < T::NativeTokenExchange::minimum_balance() {
    Err(<Error<T>>::InvalidAmount)?
  }
  // check not sent to self
  if who == recipient_id {
    Err(<Error<T>>::TransferToSelf)?
  }
  let action =
    Action::NativeTransfer { sender: who.clone(), recipient: recipient_id, amount };
  Self::validate_and_schedule_task(action, who, provided_id, execution_times)?;
  Ok(().into())
}
```

Solution

Compare the user's transfer amount with the balance

Status

Ignored; This is expected behaviour.

[N3] [Low] Return Value Not Checked

Category: Others

Content

- pallets/parachain-staking/src/lib.rs

```
//#L962
T::Currency::unreserve(&bond.owner, bond.amount);
//#L1004
T::Currency::unreserve(&candidate, state.bond);
//#L1390
T::Currency::unreserve(&delegator, amount);
```

- pallets/parachain-staking/src/types.rs

```
//#L463
T::Currency::unreserve(&who, request.amount.into());
//#L648:651
T::Currency::unreserve(&lowest_bottom_to_be_kicked.owner,lowest_bottom_to_be_kicked.a
mount,);
```

- pallets/parachain-staking/src/delegation_requests.rs

```
//#L279
T::Currency::unreserve(&delegator, amount);
```

- pallets/parachain-staking/src/migrations.rs

```
//#L404
T::Currency::unreserve(&owner, *amount);
```

The return value of `unreserve` needs to be checked.

Solution

Check the return value.

Status

Ignored; If the account has less than that locked up. Not only is this unlikely to happen, there's nothing for parachain-staking to do if it occurs.

[N4] [Suggestion] Calculate inaccurate risk

Category: Integer Overflow Audit

Content

- pallets/parachain-staking/src/types.rs
- pallets/parachain-staking/src/lib.rs
- pallets/parachain-staking/src/delegation_requests.rs

There are some risks of value overflow.

`saturating_mul`, `saturating_sub`, `saturating_add` and `+-*/, +=, -=`

saturating at the numeric bounds instead of overflowing, The returned result is inaccurate.

Solution

Use `checked_add/checked_sub/checked_mul/checked_div` instead of

`saturating_add/saturating_sub/saturating_mul/saturating_div` and `+-*/, +=, -=`.

Status

Ignored; These functions are performed on the `Balance` type. Since the same type is used for `total_issuance` I don't think we need to be worried about a portion of the issuance overflowing the data type.

[N5] [Low] Missing logic

Category: Others

Content

- pallets/parachain-staking/src/lib.rs

It is necessary to make a judgment in the case of `deposit_into_existing failure`. If the transfer fails, the entire transaction needs to be rolled back.

```
fn prepare_staking_payouts(now: RoundIndex) {
    // payout is now - delay rounds ago => now - delay > 0 else return early
    let delay = T: :RewardPaymentDelay: :get();
    if now <= delay {
        return;
    }
    let round_to_payout = now.saturating_sub(delay);
    let total_points = <Points < T >> ::get(round_to_payout);
    if total_points.is_zero() {
        return;
    }
    let total_staked = <Staked < T >> ::take(round_to_payout);
    let total_issuance = Self: :compute_issuance(total_staked);
    let mut left_issuance = total_issuance;
    // reserve portion of issuance for parachain bond account
    let bond_config = <ParachainBondInfo < T >> ::get();
    let parachain_bond_reserve = bond_config.percent * total_issuance;
    if let Ok(imb) = T: :Currency: :deposit_into_existing( & bond_config.account,
parachain_bond_reserve) {
        // update round issuance iff transfer succeeds
        left_issuance = left_issuance.saturating_sub(imb.peek());
        Self: :deposit_event(Event: :ReservedForParachainBond {
            account: bond_config.account,
            value: imb.peek(),
        });
    }

    let payout = DelayedPayout {
        round_issuance: total_issuance,
        total_staking_reward: left_issuance,
        collator_commission: <CollatorCommission < T >> ::get(),
```

```
    };

    < DelayedPayouts < T >> ::insert(round_to_payout, payout);
}
```

```
pub(crate) fn pay_one_collator_reward(paid_for_round: RoundIndex, payout_info:
DelayedPayout < BalanceOf < T >> , ) - >(Option < (T: :AccountId, BalanceOf < T > ) >
, Weight) {
    // TODO: it would probably be optimal to roll Points into the DelayedPayouts
storage
    // item so that we do fewer reads each block
    let total_points = <Points < T >> ::get(paid_for_round);
    if total_points.is_zero() {
        // TODO: this case is obnoxious... it's a value query, so it could mean one
of two
        // different logic errors:
        // 1. we removed it before we should have
        // 2. we called pay_one_collator_reward when we were actually done with
deferred
        //    payouts
        log: :warn ! ("pay_one_collator_reward called with no <Points<T>> for the
round!");
        return (None, 0u64.into());
    }

    let mint = |amt: BalanceOf < T > ,
    to: T: :AccountId | {
        if let Ok(amount_transferred) = T: :Currency: :deposit_into_existing( & to,
amt) {
            Self: :deposit_event(Event: :Rewarded {
                account: to.clone(),
                rewards: amount_transferred.peek(),
            });
        }
    };

    let collator_fee = payout_info.collator_commission;
    let collator_issuance = collator_fee * payout_info.round_issuance;

    if let Some((collator, pts)) = <AwardedPts < T >>
::iter_prefix(paid_for_round).drain().next() {
        let mut extra_weight = 0;
        let pct_due = Perbill: :from_rational(pts, total_points);
```

```
        let total_paid = pct_due * payout_info.total_staking_reward;
        let mut amt_due = total_paid;
        // Take the snapshot of block author and delegations
        let state = <AtStake < T >> ::take(paid_for_round, &collator);
        let num_delegators = state.delegations.len();
        if state.delegations.is_empty() {
            // solo collator with no delegators
            mint(amt_due, collator.clone());
            extra_weight += T: :OnCollatorPayout: :on_collator_payout(paid_for_round,
collator.clone(), amt_due, );
        } else {
            // pay collator first; commission + due_portion
            let collator_pct = Perbill: :from_rational(state.bond, state.total);
            let commission = pct_due * collator_issuance;
            amt_due = amt_due.saturating_sub(commission);
            let collator_reward = (collator_pct *
amt_due).saturating_add(commission);
            mint(collator_reward, collator.clone());
            extra_weight += T: :OnCollatorPayout: :on_collator_payout(paid_for_round,
collator.clone(), collator_reward, );
            // pay delegators due portion
            for Bond {
                owner,
                amount
            } in state.delegations {
                let percent = Perbill: :from_rational(amount, state.total);
                let due = percent * amt_due;
                if ! due.is_zero() {
                    mint(due, owner.clone());
                }
            }
        }

        (Some((collator, total_paid)), T: :WeightInfo:
:pay_one_collator_reward(num_delegators as u32) + extra_weight, )
    } else {
        // Note that we don't clean up storage here; it is cleaned up in
        // handle_delayed_payouts()
        (None, 0u64.into())
    }
}
```

Solution

If the transfer fails, the transaction should be rolled back.

Status

Ignored; The project party considers that it will be updated in subsequent versions.

[N6] [Suggestion] Missing error message

Category: Malicious Event Log Audit

Content

- pallets/parachain-staking/src/types.rs

`let new_bottom_delegation = top_delegations.delegations.pop().expect("");` missing error message.

```
pub fn add_top_delegation < T: Config > ( & mut self, candidate: &T: :AccountId,
delegation: Bond < T: :AccountId, BalanceOf < T >> , ) - >Option < Balance > where
BalanceOf < T > :Into < Balance > +From < Balance > ,
{
    let mut less_total_staked = None;
    let mut top_delegations = <TopDelegations < T >>
::get(candidate).expect("CandidateInfo existence => TopDelegations existence");
    let max_top_delegations_per_candidate = T: :MaxTopDelegationsPerCandidate:
:get();
    if top_delegations.delegations.len() as u32 == max_top_delegations_per_candidate
{
        // pop lowest top delegation
        let new_bottom_delegation = top_delegations.delegations.pop().expect("");
        top_delegations.total =
top_delegations.total.saturating_sub(new_bottom_delegation.amount);
        if matches ! (self.bottom_capacity, CapacityStatus: :Full) {
            less_total_staked = Some(self.lowest_bottom_delegation_amount);
        }
        self.add_bottom_delegation: :<T > (true, candidate, new_bottom_delegation);
    }
    // insert into top
    top_delegations.insert_sorted_greatest_to_least(delegation);
    // update candidate info
```

```
    self.reset_top_data: :<T > (candidate.clone(), &top_delegations);
    if less_total_staked.is_none() {
        // only increment delegation count if we are not kicking a bottom delegation
        self.delegation_count = self.delegation_count.saturating_add(1u32);
    } < TopDelegations < T >> ::insert( & candidate, top_delegations);
    less_total_staked
}
```

Solution

Record the corresponding error message.

Status

Ignored; The project party considers that it will be updated in subsequent versions.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002206220001	SlowMist Security Team	2022.06.06 - 2022.06.22	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 low risk, 4 suggestion vulnerabilities.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist