Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.10.23, the SlowMist security team received the team's security audit application for Stability pallets, 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.
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
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Some Risks
5	Block data Dependence Vulnerability	Passed
6	Explicit Visibility of Functions Audit	Passed
7	Arithmetic Accuracy Deviation Vulnerability	Some Risks
8	Malicious Event Log Audit	Passed

NO.	Audit Items	Result
9	Others	Some Risks
10	SAST	Passed
11	State Consistency Audit	Passed
12	Failure Rollback Audit	Passed
13	Unit Test Audit	Passed
14	Integer Overflow Audit	Some Risks
15	Parameter Verification Audit	Some Risks
16	Error Unhandle Audit	Some Risks
17	Boundary Check Audit	Passed
18	Weights Audit	Some Risks
19	Macros Audit	Passed
20	Non-standard token security audit	Passed
21	Prevent misuse audit	Passed

3 Project Overview

3.1 Project Introduction

Implementation of Stability blockchain in Substrate + Rust, a scalability solution for accessing the gas market.

3.2 Coverage

Target Code and Revision:

https://github.com/stabilityprotocol/stability

commit: 82f052a8f25774ee5dc337a8cdafad1d0064b1b8

Audit modules:

```
pallets/custom-balances/src/lib.rs
pallets/dnt-fee-controller/src/lib.rs
pallets/erc20-manager/src/lib.rs
pallets/fee-rewards-vault/src/lib.rs
pallets/root-controller/src/lib.rs
pallets/sponsored-transactions/src/lib.rs
pallets/token-fee-controller/upported-tokens-manager/src/lib.rs
pallets/token-fee-controller/user-fee-selector/src/lib.rs
pallets/token-fee-controller/validator-fee-selector/src/lib.rs
pallets/upgrade-runtime-proposal/src/lib.rs
pallets/validator-keys-controller/src/lib.rs
pallets/validator-set/src/lib.rs
pallets/zero-gas-transactions/src/lib.rs
```

3.3 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The potential loss of precision or accuracy	Arithmetic Accuracy Deviation Vulnerability	High	Fixed
N2	Overflow/underflow risks	Integer Overflow Audit	High	Fixed
N3	Program panic due to division 0	Error Unhandle Audit	High	Fixed
N4	Amount should be greater than zero	Parameter Verification Audit	Suggestion	Fixed
N5	Unreasonable pallet weight	Weights Audit	Low	Fixed
N6	`balance` precision loss due to covert U256 to u128	Arithmetic Accuracy Deviation Vulnerability	Low	Acknowledged
N7	Unimplemented function logic	Others	Suggestion	Acknowledged
N8	Node crash due to	Error Unhandle	High	Fixed

NO	Title	Category	Level	Status
	using `panic!()`	Audit		
N9	Avoid hardcoding values in the code	Others	Suggestion	Fixed
N10	Uncorrect approach to handle an error	Error Unhandle Audit	Low	Ignored
N11	Unreasonable permission	Authority Control Vulnerability Audit	Low	Ignored

4 Findings

4.1 Visibility Description

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

root-controller			
Function Name	Parameters verification coverage	weight	sender
dispatch_as_root	2/2	0	ensure_origin

sponsored-transactions			
Function Name	Parameters verification coverage	weight	sender
send_sponsored_transaction	3/4	gas_to_weight	x

upgrade-runtime-proposal			
Function Name	Parameters verification coverage	weight	sender
propose_code	2/2	0	ensure_origin
set_block_application	2/2	0	ensure_root
reject_proposed_code	1/1	0	ensure_root

validator-keys-controller			
Function Name	Parameters verification coverage	weight	sender
publish_keys	2/3	0	ensure_none

validator-set			
Function Name	Parameters verification coverage	weight	sender
add_validator	2/2	0	ensure_origin
remove_validator	2/2	0	ensure_origin
update_max_missed_epochs	2/2	0	ensure_origin
add_validator_again	2/3	0	ensure_none

zero-gas-transactions			
Function Name	Parameters verification coverage	weight	sender
send_zero_gas_transaction	2/3	gas_to_weight	x

4.2 Vulnerability Summary

[N1] [High] The potential loss of precision or accuracy

Category: Arithmetic Accuracy Deviation Vulnerability

Content

The use of `saturating_add`, `saturating_mul`, and `saturating_sub` in Rust is generally intended to prevent integer overflow and underflow, ensuring that the result remains within the valid range for the data type. However, in certain cases, relying on these functions alone can lead to inaccurate or unexpected results. This occurs when the application logic assumes that saturation alone guarantees accurate results, but ignores the potential loss of precision or accuracy.

Example:

Consider a scenario where you are calculating the total balance of accounts in a financial application. You use

`saturating_add` to add balances together to prevent overflow. However, `saturating_add` doesn't alert you to the potential loss of precision when the sum exceeds the valid range for the data type. If this application deals with very large values, you might end up with a result that is significantly less than the true sum.

Code location:

```
- pallets/dnt-fee-controller/src/lib.rs
  L108: .saturating_mul(conversion_rate.0)
  L126: let over_fee = paid_amount.saturating_sub(actual_amount);
  L128: .saturating_mul(conversion_rate.0)
  L148: .saturating_mul(conversion_rate.0)
  L158: .saturating_mul(validator_share.into())

- pallets/sponsored-transactions/src/lib.rs
  L145: gas_limit.saturating_mul(gas_price.into()),
  L156: let gas_left = gas_limit.saturating_sub(gas_used.into());
  L163: gas_left.saturating_mul(gas_price.into()),
  L285: let actual_weight = weight.saturating_add(
  L311: .saturating_mul(conversion_rate.0)

- pallets/sponsored-transactions/src/tests.rs
  L191: total_deposited = amount.saturating_add(total_deposited);
  L193: total_withdrawn = amount.saturating_add(total_withdrawn);

- pallets/validator-set/src/lib.rs
  L471: validators.len().saturating_sub(1) as u32 >= T::MinAuthorities::get(),

- pallets/zero-gas-transactions/src/lib.rs
  L144: let actual_weight = weight.saturating_add(

- pallets/zero-gas-transactions/src/tests.rs
  L191: total_deposited = amount.saturating_add(total_deposited);
  L193: total_withdrawn = amount.saturating_add(total_withdrawn);
```

Solution

Whenever performing arithmetic operations on numeric variables, use the checked arithmetic functions provided by Rust's standard library, such as `checked_add`, `checked_mul`, and `checked_sub`. These functions return `None` in case of overflow, allowing you to handle the situation gracefully.

Status

Fixed

[N2] [High] Overflow/underflow risks

Category: Integer Overflow Audit

Content

In Rust, numeric variables used in calculations without proper overflow checks, such as `checked_add`, `checked_mul`, or `checked_sub`, may be susceptible to integer overflow. Integer overflow occurs when the result of an arithmetic operation exceeds the maximum value that the data type can represent, leading to an unexpected and potentially unsafe outcome.

Code location:

```
- pallets/custom-balances/src/lib.rs
  L89: let second = self.0 - first;
  L94: Self::new(self.0 + other.0)
  L98: self.0 += other.0;

- pallets/validator-set/src/lib.rs
  L571: let session_end_block = T::SessionBlockManager::session_start_block(end_index
+ 1);

- pallets/custom-balances/src/lib.rs
  L89: let second = self.0 - first;

- pallets/dnt-fee-controller/src/lib.rs
  L161: let dapp_fee = fee_in_user_token - validator_fee;

- pallets/sponsored-transactions/src/lib.rs
  L83: gas_price * transaction_data.gas_limit,
  L
- pallets/custom-balances/src/lib.rs
  L98: self.0 += other.0;

- pallets/sponsored-transactions/src/lib.rs
  L120: SponsorNonce::<T>::mutate(meta_trx_sponsor.clone(), |nonce| *nonce += 1);

- pallets/validator-set/src/lib.rs
  L582: i += 1u32.into();
```

Solution

Whenever performing arithmetic operations on numeric variables, use the checked arithmetic functions provided by Rust's standard library, such as `checked_add`, `checked_mul`, and `checked_sub`. These functions return `None` in case of overflow, allowing you to handle the situation gracefully.

Status

Fixed

[N3] [High] Program panic due to division 0

Category: Error Unhandle Audit

Content

If the value of `conversion_rate.1` is 0, it may lead to a program panic. This is because in Rust, when performing integer division, dividing by 0 will result in a panic. Such a situation is considered undefined behavior, and Rust detects division by 0 at runtime and panics to ensure program safety.

```
- pallets/dnt-fee-controller/src/lib.rs
  L109: .div_mod(conversion_rate.1)
  L129: .div_mod(conversion_rate.1)
  L149: .div_mod(conversion_rate.1)

- pallets/sponsored-transactions/src/lib.rs
  L312: .div_mod(conversion_rate.1)
```

Solution

```
if conversion_rate.1 == U256::zero() {
    return Err(());
}
```

Status

Fixed

[N4] [Suggestion] Amount should be greater than zero

Category: Parameter Verification Audit

Content

Amount should be greater than zero, a parameter of zero is waste of gas.

- pallets/dnt-fee-controller/src/lib.rs

```
fn withdraw_fee(
    from: H160,
```

```
    token: H160,
    conversion_rate: (U256, U256),
    amount: U256, //SlowMist//
)
fn correct_fee(
    from: H160,
    token: H160,
    conversion_rate: (U256, U256),
    paid_amount: U256,  //SlowMist//
    actual_amount: U256,  //SlowMist//
)
fn pay_fees(
    token: H160,
    conversion_rate: (U256, U256),
    actual_amount: U256, //SlowMist//
    validator: H160,
    to: Option<H160>,
)
```

- pallets/fee-rewards-vault/src/lib.rs

```
pub fn add_claimable_reward(address: H160, token: H160, amount: U256) -> Result<(),
&'static str>
pub fn sub_claimable_reward(address: H160, token: H160, amount: U256)-> Result<(),
&'static str>
```

- pallets/token-fee-controller/supported-tokens-manager/src/lib.rs

```
fn ensure_sponsor_balance(sponsor: H160, token: H160, amount: U256) -> Result<(), ()>
fn transfer_fee_token(
    token: &H160,
    conversion_rate: (U256, U256),
    payer: &H160,
    payee: &H160,
    amount: U256, //SlowMist//
) -> Result<(), ()>
```

Solution

Check `amount.is_zero()`

Status

Fixed

[N5] [Low] Unreasonable pallet weight

Category: Weights Audit

Content

If too many operations have their `Weight` set to 0, it may lead to an unreasonable resource allocation, as blockchains require some basic computation and validation to maintain security.

- pallets/root-controller/src/lib.rs

```
#[pallet::call_index(0)]
#[pallet::weight(0)]
pub fn dispatch_as_root
```

- pallets/upgrade-runtime-proposal/src/lib.rs

```
#[pallet::call_index(0)]
#[pallet::weight(0)]
pub fn propose_code(origin: OriginFor<T>, code: Vec<u8>) -> DispatchResultWithPostInfo

#[pallet::call_index(1)]
#[pallet::weight(0)]
pub fn set_block_application

#[pallet::call_index(2)]
#[pallet::weight(0)]
pub fn reject_proposed_code(origin: OriginFor<T>) -> DispatchResultWithPostInfo
```

- pallets/validator-keys-controller/src/lib.rs

```
#[pallet::call_index(0)]
#[pallet::weight(0)]
pub fn publish_keys(
    origin: OriginFor<T>,
    keys: PublishingKeys<T::AuthorityId, T::FinalizationId, T::BlockNumber>,
    _signature: <T::AuthorityId as RuntimeAppPublic>::Signature, //@audit
) -> DispatchResult
```

- pallets/validator-set/src/lib.rs

```
#[pallet::call_index(0)]
#[pallet::weight(0)]
pub fn add_validator(origin: OriginFor<T>, validator_id: T::AccountId) ->
DispatchResult
```

```
#[pallet::call_index(1)]
#[pallet::weight(0)]
pub fn remove_validator(
  origin: OriginFor<T>,
  validator_id: T::AccountId,
) -> DispatchResult

#[pallet::call_index(2)]
#[pallet::weight(0)]
pub fn update_max_missed_epochs(
  origin: OriginFor<T>,
  max_missed_epochs: U256,
) -> DispatchResult

#[pallet::call_index(3)]
#[pallet::weight(0)]
pub fn add_validator_again(
  origin: OriginFor<T>,
  heartbeat: Heartbeat<T::BlockNumber, T::AuthorityId>,
  _signature: <T::AuthorityId as RuntimeAppPublic>::Signature,
) -> DispatchResult
```

- pallets/upgrade-runtime-proposal/src/lib.rs

```
fn on_initialize(n: T::BlockNumber) -> Weight
```

Solution

The formula for calculating the final fee looks like this:

```
inclusion_fee = base_fee + length_fee + [targeted_fee_adjustment * weight_fee];
final_fee = inclusion_fee + tip;
```

All dispatchable functions in Substrate must specify a weight. The way of doing that is using the annotation-based system that lets you combine fixed values for database read/write weight and/or fixed values based on benchmarks.

Status

Fixed

[N6] [Low] `balance` precision loss due to covert U256 to u128

Category: Arithmetic Accuracy Deviation Vulnerability

Content

if `actual_balance` large than `u128::MAX`, `total_balance` function will return `u128::MAX`, it is an incorrect value.

- pallets/custom-balances/src/lib.rs

```
fn total_balance(who: &T::AccountId) -> Self::Balance {
    let evm_address = T::AccountIdMapping::into_evm_address(who);
    let actual_balance =
        <T::UserFeeTokenController as
UserFeeTokenController>::balance_of(evm_address);

    let maximum_balance = sp_core::U256::from(u128::MAX);

    if maximum_balance < actual_balance {
        u128::MAX //SlowMist//
    } else {
        actual_balance.as_u128()
    }
}
```

Solution

Throw an error if the `actual_balance` is overflow.

Status

Acknowledged; This is a known limitation. Since we have to maintain the Substrate's balance interfaces because they are required by pallet_evm::Config, this interfaces are restricted to u128 type instead of the U256 type of ERC20's balance.

No big consequences can derive from this limitation. This limitation will only make fail transactions in pallet_ethereum predispatch checks but we have implemented some fallback checks that will check actual balance (U256-typed) and the transaction would go through then.

[N7] [Suggestion] Unimplemented function logic

Category: Others

Content

The following functions do not have a full implementation of the logical content, just return default value or do few things.

```
can_slash
total_issuance
minimum_balance
burn
issue
transfer
slash
deposit_into_existing
resolve_into_existing
deposit_creating
resolve_creating
withdraw
settle
can_deposit
```

Solution

Check for correct function implementation.

Status

Acknowledged; This trait has been mocked because we don't use any native token logic driven by this interface.

[N8] [High] Node crash due to using `panic!()`

Category: Error Unhandle Audit

Content

If EVM calls `make_free_balance_be`, the node crashes.

- pallets/custom-balances/src/lib.rs

```
fn make_free_balance_be(
    _who: &T::AccountId,
    _balance: Self::Balance,
) -> SignedImbalance<Self::Balance, Self::PositiveImbalance> {
    panic!("make_free_balance_be is not allowed in this pallet")
}
```

Solution

return `Err` instead of using `panic!`

Status

Fixed

[N9] [Suggestion] Avoid hardcoding values in the code

Category: Others

Content

In the provided code, hardcoding an Ethereum address (an `H160` value) directly within the `default` method is not considered a good practice. This approach lack of flexibility and reduced maintainability.

- pallets/token-fee-controller/validator-fee-selector/src/lib.rs

```
fn default() -> Self {
    Self {
        initial_default_conversion_rate_controller: <H160 as
core::str::FromStr>::from_str(
            "0x444212d6E4827893A70d19921E383130281Cda4a",
        )
        .expect("invalid address"),
    }
}
```

Solution

Consider the following approaches:

- Configuration Files.
- Environment Variables.
- Constants or Constants Modules
- Parameterization.

  Status

  Fixed

[N10] [Low] Uncorrect approach to handle an error

Category: Error Unhandle Audit

Content

`conversion_rate` return a default value of `(U256::from(1), U256::from(1))` when `T::SimulatorRunner::call` failed, it is not a correct value and may cause mistake.

- pallets/token-fee-controller/validator-fee-selector/src/lib.rs

```
fn conversion_rate(sender: H160, validator: H160, token: H160) -> (U256, U256) {
    let conversion_rate_controller = Self::conversion_rate_controller(validator);

    let args: sp_std::vec::Vec<H256> =
        sp_std::vec![sender.into(), validator.into(), token.into()];

    T::SimulatorRunner::call(
        H160::from_low_u64_be(0),
        conversion_rate_controller,
        stbl_tools::eth::generate_calldata(
            "getConversionRate(address,address,address)",
            &args,
        ),
        0.into(),
        3_000_000,
        None,
        None,
        None,
        Default::default(),
        false,
        false,
        &pallet_evm::EvmConfig::london(),
    )
    .map(|execution_info| {
        (
            U256::from_big_endian(execution_info.value[0..32].as_ref()),
            U256::from_big_endian(execution_info.value[32..64].as_ref()),
        )
    })
    .unwrap_or((U256::from(1), U256::from(1))) //SlowMist//
}
```

Solution

Throw an error if possible.

Status

Ignored; The rationale behind having a default conversion rate is to not block user's transactions in the case of a bad

validator's setup. The selected default conversion rate is arbitrary and may be needed to change it to (0, 1). In the case, the validator misses to configure right the conversion rate manager won't receive any fees.

[N11] [Low] Unreasonable permission

Category: Authority Control Vulnerability Audit

Content

`ensure_none(origin)`. It is signed by nobody, can be either: included and agreed upon by the validators anyway, or unsigned transaction validated by a pallet.

It is signed by some public key and we provide the `AccountId`.

- pallets/validator-keys-controller/src/lib.rs

```
pub fn publish_keys(
    origin: OriginFor<T>,
    keys: PublishingKeys<T::AuthorityId, T::FinalizationId, T::BlockNumber>,
    _signature: <T::AuthorityId as RuntimeAppPublic>::Signature,
) -> DispatchResult {
    ensure_none(origin)?; //SlowMist//
    //...
```

- pallets/validator-set/src/lib.rs

```
pub fn add_validator_again(
    origin: OriginFor<T>,
    heartbeat: Heartbeat<T::BlockNumber, T::AuthorityId>,
    _signature: <T::AuthorityId as RuntimeAppPublic>::Signature,
) -> DispatchResult {
    ensure_none(origin)?; //SlowMist//
    //...
```

- pallets/zero-gas-transactions/src/lib.rs

```
pub fn send_zero_gas_transaction(
    _origin: OriginFor<T>,
    transaction: Transaction,
    validator_signature: Vec<u8>,
) -> DispatchResultWithPostInfo {
    //...
    //SlowMist// Did not check the `_origin`
```

```
    //...
}
```

Solution

The use of `ensure_none` should be avoided to prevent functions from being called by evil.

Status

Ignored; This is not a vulnerability since there is a proper permission check managed through custom signatures implementations.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002310300001	SlowMist Security Team	2023.10.23 - 2023.10.30	Low Risk

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 4 high risk, 4 low risk, 3 suggestion vulnerabilities. And 2 low risk vulnerabilities were ignored;

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