Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2021.11.15, the SlowMist security team received the Bifrost team's security audit application for Bifrost, 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	Passed
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	Some Risks
16	Boundary Check Audit	Some Risks
17	Weights Audit	Passed
18	Macros Audit	Passed

3 Project Overview

3.1 Project Introduction

A parachain focused on building bridges of chains based on PoS consensus.

Project official website: https://bifrost.finance/

Project source code repository:

https://github.com/bifrost-finance/bifrost/(Version: v0.9.2-audit)

https://github.com/zenlinkpro/Zenlink-DEX-Module

(Version: 3121aacf71f65d69e6dd06828726eea11c86a041)

3.2 Coverage

Target Code and Revision:

bifrost:

node/primitives/src/currency.rs

pallets/flexible-fee/src/fee_dealer.rs

pallets/flexible-fee/src/lib.rs

pallets/flexible-fee/src/misc_fees.rs

pallets/vsbond-auction/src/lib.rs

pallets/salp/src/lib.rs

pallets/liquidity-mining/src/lib.rs

Zenlink-DEX-Module:

bifrost/zenlink-protocol/src/swap/mod.rs

bifrost/zenlink-protocol/src/*.rs (exclude rpc.rs)

3.3 Vulnerability Information

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

NO	Title	Category	Level	Status

NO	Title	Category	Level	Status
N1	Inaccurate calculation method used	Integer Overflow Audit	High	Fixed
N2	Deflation tokens are not compatible	Others	Suggestion	Ignored
N3	The size of the list is not limited	Boundary Check Audit	Low	Fixed
N4	The returned result is not handled	Error Unhandle Audit	Low	Fixed
N5	RUSTSEC Findings	SAST	Suggestion	Fixed

4 Findings

4.1 Visibility Description

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

liquidity-mining			
Function Name	Parameter verification	State consistency	Modifiers
create_mining_pool	4/7	ok	ensure_origin

liquidity-mining			
create_farming_pool	6/9	ok	ensure_origin
create_eb_farming_pool	6/9	ok	ensure_origin
charge	1/2	ok	ensure_signed
kill_pool	2/2	ok	ensure_origin
force_retire_pool	2/2	ok	ensure_origin
deposit	3/3	ok	ensure_signed
redeem	3/3	ok	ensure_signed
redeem_all	2/2	ok	ensure_signed
volunteer_to_redeem	2/3	ok	-
claim	2/2	ok	ensure_signed

vsbond-auction			
Function Name	Parameter verification	State consistency	Modifiers
create_order	3/7	ok	ensure_signed
revoke_order	2/2	ok	ensure_signed
clinch_order	2/2	ok	ensure_signed
partial_clinch_order	3/3	ok	ensure_signed

flexible-fee			
Function Name	Parameter verification	State consistency	Modifiers
set_user_fee_charge_order	1/2	ok	ensure_signed

salp			
Function Name	Parameter verification	State consistency	Modifiers
fund_success	2/2	ok	ensure_origin
fund_fail	2/2	ok	ensure_origin
fund_retire	2/2	ok	ensure_origin
fund_end	2/2	ok	ensure_origin
edit	3/6	ok	ensure_origin
unlock	1/3	ok	-
batch_unlock	2/2	ok	ensure_signed
create	4/5	ok	ensure_origin
contribute	3/3	ok	ensure_signed
confirm_contribute	3/5	ok	ensure_origin
withdraw	2/2	ok	ensure_origin
refund	1/2	ok	ensure_signed
batch_refund	1/2	ok	ensure_signed
redeem	3/3	ok	ensure_signed

dissolve	2/2	ok	ensure_origin
add_proxy	1/2	ok	ensure_origin
remove_proxy	1/2	ok	ensure_origin
mint	1/2	ok	ensure_signed

zenlink-protocol			
Function Name	Parameter verification	State consistency	Modifiers
set_fee_receiver	3/3	ok	ensure_root
set_fee_point	2/2	ok	ensure_root
transfer	4/4	ok	ensure_signed
transfer_to_parachain	5/6	ok	ensure_origin
create_pair	3/3	ok	ensure_root
add_liquidity	6/8	ok	ensure_signed
remove_liquidity	4/8	ok	ensure_signed
swap_exact_assets_for_assets	6/6	ok	ensure_signed
swap_assets_for_exact_assets	6/6	ok	ensure_signed
bootstrap_create	3/8	ok	ensure_root
bootstrap_contribute	6/6	ok	ensure_signed
bootstrap_claim	4/5	ok	ensure_signed
bootstrap_end	3/3	ok	ensure_signed

zenlink-protocol			
bootstrap_update	3/8	ok	ensure_root
bootstrap_refund	3/3	ok	ensure_signed

4.2 Vulnerability Summary

[N1] [High] Inaccurate calculation method used

Category: Integer Overflow Audit

Content

- zenlink-protocol/src/swap/mod.rs
- zenlink-protocol/src/foreign/mod.rs
- pallets/liquidity-mining/src/lib.rs
- pallets/salp/src/lib.rs
- pallets/vsbond-auction/src/lib.rs

`saturating_mul`, `saturating_sub`, `saturating_add`

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

Solution

Can use `checked_div`, `checked_mul`, `checked_add`, `checked_sub` to throw an exception when the calculation result overflows and do not continue to execute.

Status

Fixed;

- zenlink-protocol/src/swap/mod.rs
- zenlink-protocol/src/foreign/mod.rs

  Fixed in https://github.com/zenlinkpro/Zenlink-DEX-

Module/commit/59df37ad51ccddaef199879aa04fb541cd08095b

- pallets/liquidity-mining/src/lib.rs
- pallets/salp/src/lib.rs
- pallets/vsbond-auction/src/lib.rs

  Fixed in https://github.com/bifrost-finance/bifrost/commit/bb009e3917428bf980a80509d2db7322678e808c

[N2] [Suggestion] Deflation tokens are not compatible

Category: Others

Content

- pallets/liquidity-mining/src/lib.rs

  `fn deposit`

```
pub fn deposit(origin: OriginFor < T > , pid: PoolId, value: BalanceOf < T > , ) -
>DispatchResultWithPostInfo {
    let user = ensure_signed(origin) ? ;

    let mut pool: PoolInfo < T > =Self: :pool(pid).ok_or(Error: :<T >
::InvalidPoolId) ? .try_retire().try_update();

    ensure ! (pool.state == PoolState: :Charged || pool.state == PoolState: :Ongoing,
Error: :<T > ::InvalidPoolState);

    ensure ! (value >= T: :MinimumDepositOfUser: :get(), Error: :<T >
::TooLowToDeposit);

    let mut deposit_data: DepositData < T > =Self: :user_deposit_data(pid,
user.clone()).unwrap_or(DepositData: :<T > ::from_pool( & pool));
    if pool.state == PoolState: :Ongoing && pool.update_b != deposit_data.update_b {
        pool.try_settle_and_transfer( & mut deposit_data, user.clone()) ? ;
    }

    deposit_data.deposit = deposit_data.deposit.saturating_add(value);
    pool.deposit = pool.deposit.saturating_add(value);
    ensure ! (pool.deposit <= T: :MaximumDepositInPool: :get(), Error: :<T >
```

```
::ExceedMaximumDeposit);

    // To "lock" the deposit
    match pool.r#type {
        PoolType: :Mining = >{
            let lpt = Self: :convert_to_lptoken(pool.trading_pair) ? ;

            T: :MultiCurrency: :transfer(lpt, &user, &pool.keeper, value).map_err( |
_e | Error: :<T > ::NotEnoughToDeposit) ? ;** //SlowMist If it is a deflationary
currency, then the actual amount received will be less than the value passed in.**
        },
        PoolType: :Farming = >{
            let(token_a, token_b) = pool.trading_pair;

            T: :MultiCurrency: :transfer(token_a, &user, &pool.keeper,
value).map_err( | _e | Error: :<T > ::NotEnoughToDeposit) ? ;
            T: :MultiCurrency: :transfer(token_b, &user, &pool.keeper,
value).map_err( | _e | Error: :<T > ::NotEnoughToDeposit) ? ;
        },
        PoolType: :EBFarming = >{
            let(token_a, token_b) = pool.trading_pair;

            ensure ! (T: :MultiCurrency: :reserved_balance(token_a, &user) >=
deposit_data.deposit, Error: :<T > ::NotEnoughToDeposit);
            ensure ! (T: :MultiCurrency: :reserved_balance(token_b, &user) >=
deposit_data.deposit, Error: :<T > ::NotEnoughToDeposit);
        },
    }

    let r#type = pool.r#type;
    let trading_pair = pool.trading_pair;

    TotalPoolInfos: :<T > ::insert(pid, pool);
    TotalDepositData: :<T > ::insert(pid, user.clone(), deposit_data);

    Self: :deposit_event(Event: :UserDeposited(pid, r#type, trading_pair, value,
user));

    Ok(().into())
}
```

If the deposit is in deflationary currency, the actual received amount does not match the recorded amount.

Solution

Record the amount of the account before the transfer. Record the amount of the account once after the transfer. The two are subtracted and compared with the incoming amount

Status

Ignored; At present, Bifrost has no plan to access deflationary assets, and will not be modified for the time being. The follow-up is to add restrictions on deflationary assets.

[N3] [Low] The size of the list is not limited

Category: Boundary Check Audit

Content

- pallets/flexible-fee/src/lib.rs

  `fn set_user_fee_charge_order`

  `asset_order_list_vec` does not limit the length of the list.

```
pub fn set_user_fee_charge_order(origin: OriginFor < T > , asset_order_list_vec:
Option < Vec < CurrencyIdOf < T >>> , ) - >DispatchResult {
    let who = ensure_signed(origin) ? ;

    if let Some(mut asset_order_list) = asset_order_list_vec {
        asset_order_list.insert(0, T: :NativeCurrencyId: :get());
        asset_order_list.dedup();//SlowMist No sorting first, de-duplication
        UserFeeChargeOrderList: :<T > ::insert( & who, asset_order_list);
    } else {
        UserFeeChargeOrderList: :<T > ::remove( & who);
    }

    Ok(().into())
}
```

Solution

The maximum length of `asset_order_list_vec` can be judged.

`asset_order_list` can be sorted first in deduplication.

Status

Fixed

[N4] [Low] The returned result is not handled

Category: Error Unhandle Audit

Content

- zenlink-protocol/src/swap/mod.rs

  `fn do_bootstrap_claim`

```
pub(crate) fn do_bootstrap_claim(who: T: :AccountId, recipient: T: :AccountId,
asset_0: AssetId, asset_1: AssetId, ) - >DispatchResult {
    let pair = Self: :sort_asset_id(asset_0, asset_1);
    match Self: :pair_status(pair) {
        Trading(_) = >BootstrapPersonalSupply: :<T > ::try_mutate_exists((pair,
&who), |contribution | {
            if let Some((amount_0_contribute, amount_1_contribute)) =
contribution.take() {
                if let Bootstrap(bootstrap_parameter) = Self:
:bootstrap_end_status(pair) {
                    ensure ! (!Self: :bootstrap_disable( & bootstrap_parameter),
Error: :<T > ::DisableBootstrap);
                    let exact_amount_0 =
amount_0_contribute.saturating_mul(bootstrap_parameter.accumulated_supply.1).saturati
ng_add(amount_1_contribute.saturating_mul(bootstrap_parameter.accumulated_supply.0),
).checked_div(bootstrap_parameter.accumulated_supply.1.saturating_mul(2)).ok_or(Error
: :<T > ::Overflow) ? ;

                    let exact_amount_1 =
amount_1_contribute.saturating_mul(bootstrap_parameter.accumulated_supply.0).saturati
ng_add(amount_0_contribute.saturating_mul(bootstrap_parameter.accumulated_supply.1),
).checked_div(bootstrap_parameter.accumulated_supply.0.saturating_mul(2)).ok_or(Error
: :<T > ::Overflow) ? ;

                    let calculated_liquidity =
exact_amount_0.saturating_mul(exact_amount_1).integer_sqrt();
```

```
                    let pair_account = Self: :pair_account_id(pair.0, pair.1);
                    let lp_asset_id = Self: :lp_pairs(pair).ok_or(Error: :<T >
::InsufficientAssetBalance) ? ;

                    T: :MultiAssetsHandler: :transfer(lp_asset_id, &pair_account,
&recipient, calculated_liquidity) ? ;

                    Self: :deposit_event(Event: :BootstrapClaim(pair_account,
who.clone(), recipient, pair.0, pair.1, amount_0_contribute, amount_1_contribute,
calculated_liquidity, ));

                    Ok(())
                } else {
                    Err(Error: :<T > ::NotInBootstrap.into())
                }
            } else {
                Err(Error: :<T > ::ZeroContribute.into())
            }
        }),//SlowMist The returned result is not handled
        _ = >Err(Error: :<T > ::NotInBootstrap.into()),
    }
}
```

Solution

The result returned by `try_mutate_exists` needs to be handled

Status

Fixed

[N5] [Suggestion] RUSTSEC Findings

Category: SAST

Content

(1)

Crate: hyper

Version: 0.12.36

Title: Integer overflow in `hyper`'s parsing of the `Transfer-Encoding` header leads to data loss

Date: 2021-07-07

ID: RUSTSEC-2021-0079

URL: https://rustsec.org/advisories/RUSTSEC-2021-0079

Solution: Upgrade to >=0.14.10

Dependency tree:

hyper 0.12.36

(2)

Crate: hyper

Version: 0.12.36

Title: Lenient `hyper` header parsing of `Content-Length` could allow request smuggling

Date: 2021-07-07

ID: RUSTSEC-2021-0078

URL: https://rustsec.org/advisories/RUSTSEC-2021-0078

Solution: Upgrade to >=0.14.10

(3)

Crate: hyper

Version: 0.13.10

Title: Integer overflow in `hyper`'s parsing of the `Transfer-Encoding` header leads to data loss

Date: 2021-07-07

ID: RUSTSEC-2021-0079

URL: https://rustsec.org/advisories/RUSTSEC-2021-0079

Solution: Upgrade to >=0.14.10

(4)

Crate: libsecp256k1

Version: 0.3.5

Title: libsecp256k1 allows overflowing signatures

Date: 2021-07-13

ID: RUSTSEC-2021-0076

URL: https://rustsec.org/advisories/RUSTSEC-2021-0076

Solution: Upgrade to >=0.5.0

Dependency tree:

libsecp256k1 0.3.5

(5)

Crate: time

Version: 0.1.44

Title: Potential segfault in the time crate

Date: 2020-11-18

ID: RUSTSEC-2020-0071

URL: https://rustsec.org/advisories/RUSTSEC-2020-0071

Solution: Upgrade to >=0.2.23

Dependency tree:

time 0.1.44

(6)

Crate: wasmtime

Version: 0.27.0

Title: Multiple Vulnerabilities in Wasmtime

Date: 2021-09-17

ID: RUSTSEC-2021-0110

URL: https://rustsec.org/advisories/RUSTSEC-2021-0110

Solution: Upgrade to >=0.30.0

Solution

Upgrade the corresponding library to the resolved version.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
BCA002111300001	SlowMist Security Team	2021.11.15 - 2021.11.30	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 1 high risk, 2 low risk, 2 suggestion vulnerabilities.All other findings were fixed.

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