BEOSIN
Blockchain Security

Bifrost (vtoken-minting && slp-v2)

Smart Contract Security Audit

No. 202509251121

Sep 25th, 2025

SECURING BLOCKCHAIN ECOSYSTEM

Contents

Summary of Audit Results

After auditing, 1 Medium-risk, 1 Low-risk and 2 Info items were identified in the Bifrost (vtoken-minting && slp-v2) project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Project Description:

This audit covered the vtoken-minting and slp-v2 pallets of the Bifrost project. Bifrost's non-custodial staking liquidity solution allows users to stake their tokens in exchange for vTokens (voucher tokens). The vtoken-minting pallet implements the minting, redemption, and locking logic for vTokens on the Bifrost platform. It manages users' asset flows and locks through multiple ledgers, including TokenPool, TokenUnlockLedger, TimeUnitUnlockLedger, and UserUnlockLedger, supporting both vToken minting and redemption while handling fee calculation, pool balances, and issuance updates. It also supports cross-chain and HyperBridge transfers. For ETH-type assets, unlock IDs are maintained separately, and redemption is processed according to time units to ensure assets are gradually released as planned.

The pallet also implements an incentive mechanism, minting rewards by locking users' vTokens and calculating rewards based on users' holdings of BbBNC tokens. The overall logic covers cross-chain operations, time locks, ledger management, and fee handling, ensuring asset security, proper locking, and accurate reward calculation, while providing events for on-chain monitoring and interaction with other modules.

The slp-v2 pallet is the upgraded version of Bifrost's Staking Liquidity Pool (SLP), supporting multiple staking protocols including Ethereum Staking and Astar Dapp Staking. It manages delegators, validators, and their ledgers through multi-layer storage structures, such as ConfigurationByStakingProtocol, DelegatorByStakingProtocolAndDelegatorIndex, and LedgerByStakingProtocolAndDelegator, recording staked assets, lock information, and reward status. The pallet provides functions for setting protocol configurations, adding/removing delegators and validators, updating ledgers, transferring assets cross-chain, handling XCM messages, and updating time units and token exchange rates, with an event system to record on-chain operations.

The code strictly enforces access control, allowing only governance or authorized staking protocol operators to perform critical actions such as asset transfers, time unit and exchange rate updates, and cross-chain staking tasks. It also handles protocol fee calculation, token pool management, and interaction with the vToken module, ensuring asset security, proper locking, and correct reward calculations, while supporting multi-chain and cross-module asset operations and event notifications.

1 Overview

1.1 Project Overview

Project Name	Bifrost (vtoken-minting && slp-v2)
Project Language	Rust
Platform	Polkadot
Code Base	https://github.com/bifrost-io/bifrost/tree/bifrost-v0.21.0 https://github.com/bifrost-io/bifrost/tree/audit-v2-vtoken-minting-1 https://github.com/bifrost-io/bifrost/tree/audit-v1-slp-1 https://github.com/bifrost-io/bifrost/tree/audit-v2-slp-1
Audit Scope	/pallets/vtoken-minting/src/lib.rs#L1201-L1281 /pallets/vtoken-minting/src/impls.rs /pallets/slp-v2/src/lib.rs#L382-L854 /pallets/slp-v2/src/common/impls.rs#L124-L198 /pallets/slp-v2/src/ethereum_staking/types.rs /pallets/slp-v2/src/ethereum_staking/impls.rs
Commit Hash	bifrost-v0.21.0 9de153188edac72ec4b6f815007b62012fadf205 audit-v2-vtoken-minting-1 806651cd16f2102a1cdbf2d4ca730358d088b8f1 944c7bfb50d51e7f71c90b272b3a3afe88037dca 4b3f86a61d6420a2affacabb23cb753a80e48d09 audit-v1-slp-1 5d54d969a685f8260323904119752e8da2b202a1 475660bad9b059cd5db4ac0f32776990b7232ee2 audit-v2-slp-1 bc068b1c06c824bcfdd662c63e2a3faef370856a

1.2 Audit Overview

Audit work duration: Sep 5, 2025 – Sep 19, 2025, Sep 25, 2025

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's Business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Bifrost-01	Inaccurate Transfer Amount in Function Logic	Medium	Fixed
Bifrost-02	Potential Saturating Operation Logic Deviation	Low	Fixed
Bifrost-03	Inaccurate Weight Calculation	Info	Fixed
Bifrost-04	Incorrect Event Emission for Invalid Ledger Types	Info	Fixed

Finding Details:

[Bifrost-01] Inaccurate Transfer Amount in Function Logic

Severity Level	Medium
Type	Business Security
Lines	pallets/slp-v2/src/common/impls.rs #L149, 195
Description	In the `do_transfer_to` function, if the `StakingProtocol` is `AstarDappStaking`, the function uses the entire balance of the entrance account as the transfer amount instead of the amount specified in the parameter. This behavior appears to be incorrect.

```
let entrance_account_free_balance =
    T::MultiCurrency::free_balance(currency_id, &entrance_account);
match staking_protocol {
    StakingProtocol::AstarDappStaking => {
        let dest_beneficiary_location = staking_protocol
            .get_dest_beneficiary_location::<T>(delegator.clone())
            .ok_or(Error::<T>::UnsupportedStakingProtocol)?;
        T::XcmTransfer::transfer(
            entrance_account.clone(),
            currency_id,
            entrance_account_free_balance,
            dest_beneficiary_location,
            WeightLimit::Unlimited,
        )
        .map_err(|_| Error::<T>::DerivativeAccountIdFailed)?;
    }
```

Additionally, the event should record amount rather than the full balance.

```
Self::deposit_event(Event::TransferTo {
    staking_protocol,
    from: entrance_account,
    to: delegator,
    amount: entrance_account_free_balance,
});
```

Recommendation	For the branch where the `StakingProtocol` is `AstarDappStaking`, it is recommended to first check whether the entrance_account balance is sufficient, requiring that entrance_account_free_balance is not less than

	amount, and then use amount for both the transfer amount and the event.
Status	**Fixed.** This issue has been fixed in commit a9e80047. The event is now triggered based on the actual transferred amount.

[Bifrost-02] Potential Saturating Operation Logic Deviation

Severity Level	Low
Type	Business Security
Lines	pallets/slp-v2/src/ethereum_staking/types.rs #L62-72
Description	In the `ethereum_staking` module, functions use saturating_accrue and saturating_reduce to update the total locked amount. While these saturating operations help prevent traditional overflow or underflow errors, they "silently" handle boundary cases (e.g., when reaching Balance::MAX or zero) by halting further increments or decrements instead of throwing an error. This behavior may cause inconsistencies between the locked amount and the intended business logic, potentially leading to ledger state deviations or increasing the complexity of auditing and traceability.
Recommendation	To prevent discrepancies between the locked amount and business logic, it is recommended to add explicit boundary checks before critical updates. For instance, prior to calling saturating_accrue or saturating_reduce, verify that the resulting value will not exceed Balance::MAX or drop below zero, and trigger an error or exception handling if it does. Additionally, recording events after ledger updates can facilitate on-chain auditing and traceability.
Status	**Fixed.** This issue has been fixed by adding overflow checks to ensure security.

```rust
pub fn add_lock_amount(&mut self, amount: Balance) {
    self.locked.saturating_accrue(amount);
}

/// Subtracts the specified amount of the total locked amount.
pub fn subtract_lock_amount(&mut self, amount: Balance) {
    self.locked.saturating_reduce(amount);
}
```

[Bifrost-03] Inaccurate Weight Calculation

Severity Level	Info
Type	General Vulnerability
Lines	pallets/vtoken-minting/src/lib.rs #L1226 pallets/slp-v2/src/lib.rs #L844

Description	The weight calculation for the set_v_currency_issuance function is inaccurate. The code currently specifies reads = 1, writes = 1, but in reality, `VtokenIssuance::<T>::mutate` performs one read and one write, and the subsequent call to `VtokenIssuance::<T>::get` adds an additional read. Therefore, the correct storage operations should be reads = 2, writes = 1. The current annotation underestimates the number of reads, which may lead to inaccurate on-chain resource consumption assessment.
	Additionally, the current `ethereum_staking` function incorrectly uses `astar_dapp_staking()` to estimate weight, which is unreasonable. Weight should instead be computed dynamically based on the actual operations or using a parameterized `WeightInfo` function.
	`#[pallet::weight(<T as Config>::WeightInfo::astar_dapp_staking())]`
	`pub fn ethereum_staking(`

Recommendation	For the `set_v_currency_issuance` function, it is recommended to update the weight annotation to `#[pallet::weight(T::DbWeight::get().reads_writes(2, 1))]` to accurately reflect the actual storage operations. For the `ethereum_staking` function, it is advised to implement a corresponding method in `WeightInfo` and use it in the call, ensuring the weight reflects the true operational cost.
Status	**Fixed.** This issue has been resolved. In the latest version of the code, a corresponding weight calculation has been added for the function to ensure accurate resource consumption estimation.

[Bifrost-04] Incorrect Event Emission for Invalid Ledger Types

Severity Level	Info
Type	Business Security
Lines	pallets/slp-v2/src/ethereum_staking/impls.rs #L29-56
Description	In the `do_ethereum_staking` function, ledgers that are not of the `EthereumStaking` type currently trigger the `EthereumStaking` event without raising an error. This behavior can create a mismatch between emitted events and the actual ledger state, potentially misleading front-end clients, monitoring tools, or on-chain observers. Such inconsistencies may result in incorrect UI displays, erroneous analytics, or misinterpreted on-chain data.

```
    pub fn do_ethereum_staking(
        delegator: Delegator<T::AccountId>,
        task: EthereumStaking,
    ) -> DispatchResultWithPostInfo {
        Self::ensure_delegator_exist(&ETHEREUM_STAKING, &delegator)?;
        LedgerByStakingProtocolAndDelegator::<T>::mutate(
            ETHEREUM_STAKING,
            delegator.clone(),
            |ledger| -> Result<(), Error<T>> {
                if let Some(Ledger::EthereumStaking(mut
pending_ledger)) = ledger.clone() {
                    match task {
                        EthereumStaking::Stake(amount) => {
                            pending_ledger.add_lock_amount(amount);
                        }
                        EthereumStaking::Unstake(amount) => {
                            if pending_ledger.locked < amount {
                                return
Err(Error::<T>::InvalidParameter);
                            }
                            pending_ledger.subtract_lock_amount(amount
);
                        }
                    }
                    *ledger =
Some(Ledger::EthereumStaking(pending_ledger));
                };
```

```
                Ok(())
            },
        )?;
        Self::deposit_event(Event::EthereumStaking { delegator,
task });
        Ok(().into())
    }
```

Recommendation	Ensure that the event is only emitted when the ledger is confirmed to be of the `EthereumStaking` type, and consider raising an error or skipping the operation otherwise. This will maintain consistency between ledger state and emitted events.
Status	**Fixed.**

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract Business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract Business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract Business system, individual Business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract Business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Deprecated Items
		Redundant Code
		Interface Specification
		Function Call Permissions
		Returned Value Security
2	General Vulnerability	Integer Overflow/Underflow
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Reentrancy
		Replay Attack
		Storage/State Inconsistency
		Upgrade & Storage Layout Risks
		Cross-chain / XCM Interface Consistency
		Weight / Resource Misestimation
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the Business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the Business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com