SALUS SECURITY

OCT 2025

CODE SECURITY ASSESSMENT

LORENZO

Overview

Project Summary

- Name: Lorenzo - OTFVault
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Lorenzo-Protocol/OTF-Contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Lorenzo - OTFVault
Version	v1
Type	Solidity
Dates	Oct 14 2025
Logs	Oct 14 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	6
Total	8

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Incomplete pause coverage allows state changes during emergencies	Low	Business logic	Pending
2	Missing events for functions that change critical state	Low	Logging	Pending
3	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Pending
4	Inconsistent native-asset validation between deposit paths enables misconfiguration	Informational	Inconsistency	Pending
5	Lack of License Specification	Informational	Configuration	Pending
6	Mapping declaration lacks named parameters for improved readability	Informational	Code Quality	Pending
7	Function Visibility Overly Permissive	Informational	Code Quality	Pending
8	Gas optimization	Informational	Gas Optimization	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Incomplete pause coverage allows state changes during emergencies

Severity: Low	Category: Business logic
Target: - contracts/vaults/BnbPlusVault.sol	

Description

The pause guard (`whenNotPaused`) protects `confirmShare()` and `refundShare()` but not `directDeposit()`, `mint()`, `burn()`, and `burnFrom()`. During a paused state, privileged actors can still mint/burn shares or execute direct deposits (which also transfers native funds to the custody wallet), defeating the expected “global freeze.”

Recommendation

It is recommended to apply `whenNotPaused` to `directDeposit()`, `mint()`, `burn()`, and `burnFrom()`.

2. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - contracts/vaults/BnbPlusVault.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `BnbPlusVault` contract, some events are lacking in the privileged setter functions (e.g. `setUnderlying()`, `setSigner()`).

Recommendation

It is recommended to emit events for critical state changes.

2.3 Informational Findings

3. Lack of Security Contact Information for Responsible Disclosure

Severity: Informational	Category: Configuration
Target: - contracts/vaults/BnbPlusVault.sol	

Description

The contract `BnbPlusVault` does not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

4. Inconsistent native-asset validation between deposit paths enables misconfiguration

Severity: Informational	Category: Inconsistency

Target:
- contracts/vaults/BnbPlusVault.sol

Description

The `onDepositUnderlying()` function hard-checks `underlyingToken == NATIVE_TOKEN` (the sentinel 0xEeee…), while `directDeposit()` checks `underlyingToken == underlying`. The manager can set `underlying` to any non-zero address (`setUnderlying()` only requires `_underlying != address(0)`). if `underlying` is mis-set, there is an inconsistency across the two deposit flows.

contracts/vaults/BnbPlusVault.sol:L135-L159

```
// to handle the depoisted underlying asset
function onDepositUnderlying(address from, address underlyingToken, uint256
underlyingAmount) internal override returns (uint256 sharesAmount, uint256 unitNav)
{
    require(underlyingToken == NATIVE_TOKEN, "only native token");
    require(msg.value == underlyingAmount, "not pay enough value");
    require(underlyingAmount >= minDepositAmount, "deposit amount too small");
    ...
}
```

contracts/vaults/BnbPlusVault.sol:L284-L304

```
function directDeposit(address underlyingToken, uint256 underlyingAmount) public payable
onlyDepositor {
    // WARN: only support deposit bnb to the vault
    require(underlyingToken == underlying, "only native token deposit");
    ...
}
```

Recommendation

Make both deposit paths enforce the same rule and prevent misconfiguration.

5. Lack of License Specification

Severity: Informational	Category: Configuration
Target: - contracts/vaults/BnbPlusVault.sol	

Description

The contract `BnbPlusVault` does not specify a license or specify an incorrect one (e.g., // SPDX-License-Identifier: UNLICENSED).

Recommendation

It is recommended to explicitly specify a license (e.g., MIT or GPL-3.0) in all contracts and interfaces and maintain consistency across the codebase. This establishes clear legal usage terms, prevents misuse, and encourages collaboration.

6. Mapping declaration lacks named parameters for improved readability

Severity: Informational	Category: Code Quality
Target: - contracts/vaults/BnbPlusVault.sol	

Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In the `BnbPlusVault` contract, the mappings are declared without named parameters, making the purpose of the key and value less obvious upon inspection.
contracts/vaults/BnbPlusVault.sol:L42-L54

```
mapping(address => mapping(uint128 => uint256)) public pendingShares; // user -> period
-> pending shares
mapping(address => bool) public signers; // signer -> is signer
mapping(bytes32 => bool) public confirmedTx; // txHash -> is confirmed
mapping(bytes32 => bool) public refundedTx; // txHash -> is refunded

// record the confirmed shares of the user of a period,
// because the user can not request withdraw the confirmed shares in the same period,
// the confirmed shares must stay in the vault for at least next period
mapping(address => mapping(uint128 => uint256)) public confirmedShares; // user ->
period -> confirmed shares

// add roles for minter and burner for chainlink ccip @Sep. 3, 2025
// mint role:  0x01, burner role: 0x02, depositor role: 0x04
mapping(address => uint256) public roles; // account -> role
```

Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

7. Function Visibility Overly Permissive

Severity: Informational	Category: Code Quality

Target:
- contracts/vaults/BnbPlusVault.sol

Description

Some functions in the contract are declared with broader visibility than necessary (e.g., public instead of external). Overly permissive visibility can expose internal logic to unintended callers, increase the attack surface, and slightly increase gas costs. Functions intended to be called externally only do not need public visibility and could be declared external to better communicate intended usage and optimize gas.
In the `BnbPlusVault` contract, some functions declared with broader visibility than necessary (e.g. `setUnderlying()`, `setSigner()`, `setRole()`, `confirmShare()`, `refundShare()`, `mint()`, `burn()`, `burnFrom()`, `directDeposit()`).

Recommendation

Consider changing the visibility of the above function.

8. Gas optimization

Severity: Informational	Category: Gas Optimization
Target: - contracts/vaults/BnbPlusVault.sol	

Description

Throughout the codebase, these contracts use <if (condition) revert("error message") statements and require(condition, "error message") statements>. Since Solidity version 0.8.26, require statements support custom errors, which are more gas-efficient and improve code clarity. Initially, this feature was only available through the IR pipeline, but starting from Solidity 0.8.27, it is also supported in the legacy pipeline.

In several parts of the codebase, loops use the postfix increment operator (i++) for iteration. While functionally correct, this introduces an unnecessary step of storing the original value before incrementing, compared to using the prefix increment operator (++i). As the return value of the expression is ignored, this results in extra gas costs. In scenarios where loops run frequently, this overhead may accumulate to significant gas usage.

Since Solidity 0.8.0, arithmetic operations include automatic overflow/underflow checks, which add gas overhead to loop increments. The common workaround is to manually use unchecked { ++i; } inside the loop. Starting with Solidity 0.8.22, the compiler automatically emits an “unchecked” increment for for-loop counters when specific conditions hold, so developers no longer need to write unchecked by hand; this optimization even applies when the general optimizer is disabled (and can be explicitly turned off via Standard JSON).

Recommendation

Consider optimizing gas usage and improving code readability by replacing if-revert and require(condition, "error message") statements with require(condition, CustomError()), preferring the prefix increment operator (++i) over the postfix increment operator (i++) in loop iterations, and wrapping the counter increment in an unchecked block.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 2c61bad:

File	SHA-1 hash
contracts/vaults/BnbPlusVault.sol	3f4ffbfef5c3c2e428503dbf66b1008bede280ba