SALUS SECURITY

SEP 2025

CODE SECURITY ASSESSMENT

ZEROBASE

Overview

Project Summary

- Name: ZEROBASE - ZBT
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/ZeroBase-Pro/Economics
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ZEROBASE - ZBT
Version	v2
Type	Solidity
Dates	Sep 17 2025
Logs	Sep 17 2025; Sep 17 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	4
Total	6

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 **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. The restrictions on SUPPLY_CAPD may be breached. 6

2. Centralization risk with initial token distribution 7

2.3 Informational Findings 8

3. Incomplete Docstrings 8

4. Mapping Declaration Lacks Named Parameters for Improved Readability 9

5. Custom Errors in require Statements 10

6. Hardcoded Addresses 11

Appendix **12**

Appendix 1 - Files in Scope 12

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	The restrictions on SUPPLY_CAPD may be breached.	Medium	Business Logic	Resolved
2	Centralization risk with initial token distribution	Low	Centralization	Acknowledged
3	Incomplete Docstrings	Informational	Code Quality	Acknowledged
4	Mapping Declaration Lacks Named Parameters for Improved Readability	Informational	Code Quality	Acknowledged
5	Custom Errors in require Statements	Informational	Gas Optimization	Acknowledged
6	Hardcoded Addresses	Informational	Code Quality	Acknowledged

2.2 Notable Findings

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

1. The restrictions on SUPPLY_CAPD may be breached.	
Severity: Medium	Category: Business logic
Target: - contracts/ZEROBASE.sol	

Description

contracts/ZEROBASE.sol:L90 - L94

```
function mint(address _to, uint256 _amount) external {
    require(totalSupply() + _amount <= SUPPLY_CAP);
    require(isMinter[msg.sender]);
    _mint(_to, _amount);
}
```

Within the `ZEROBASE` contract, a `SUPPLY_CAP` exists to limit the maximum on-chain token supply. However, this restriction currently resides solely within the `mint()` function, meaning tokens may bypass this limit during cross-chain transfers. For example:

1. When 900 million tokens are cross-chained from BSC to the Ethereum chain, BSC's `totalsupply` will decrease by 900 million.

2. The `minter` subsequently mints 900M tokens via the `mint()` function. At this point, the `totalsupply` still does not exceed the `SUPPLY_CAP` limit.

3. Bridging the 900M tokens back from Ethereum to BSC increases the `totalsupply` to 900M + 900M = 1800M, exceeding the `SUPPLY_CAP` limit.

Recommendation

It is recommended that the contract logic be amended so that cross-chain transactions do not affect the `SUPPLY_CAP` restriction.

Status

The team has resolved this issue in commit be1bedd.

2. Centralization risk with initial token distribution	
Severity: Low	Category: Centralization
Target: - contracts/ZEROBASE.sol	

Description

contracts/ZEROBASE.sol:L33 - L48

```
constructor(
    address _owner
) OFT("ZEROBASE", "ZBT", LZ_POINT, _owner) Ownable(_owner) {
    isWhitelisted[_owner] = true;
    isMinter[_owner] = true;
    uint8 id;

    assembly {
        id := chainid()
    }

    if(id == 56){
        SUPPLY_CAP = 1_000_000_000 * 10 ** 18;
        _mint(_owner, SUPPLY_CAP * 90 / 100);
    }
}
```

When the contract is deployed, `$ZKB` is sent to one account. This account then has full control over the token distribution. If it is an EOA account, any compromise of its private key could drastically affect the project – for example, attackers could manipulate the price of `$ZKB` on the DEX if they gain access to the private key.

Recommendation

It is recommended to transfer tokens to a multi-sig account and promote transparency by providing a breakdown of the intended initial token distribution in a public location.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Incomplete Docstrings	
Severity: Informational	Category: Code Quality
Target: - contracts/ZEROBASE.sol	

Description

In the `ZEROBASE` contract ,multiple instances of incomplete docstrings were identified across the codebase. Several functions/events do not document all of their parameters or return values. This reduces readability and maintainability of the code, and increases the effort required by developers, auditors, or integrators to fully understand the contract logic. When such functions are part of the contract's public API, incomplete documentation may further hinder external usage and integration.

Recommendation

It is recommended to provide complete documentation for all public functions and events, including details on their parameters and return values. Following the Ethereum Natural Specification Format (NatSpec) is highly encouraged to ensure consistency, readability, and usability. Comprehensive documentation improves long-term maintainability and facilitates both auditing and integration processes.

Status

This issue has been acknowledged by the team.

4. Mapping Declaration Lacks Named Parameters for Improved Readability	
Severity: Informational	Category: Code Quality
Target: - contracts/ZEROBASE.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 `` `ZEROBASE.sol` `` , the mappings `` `isWhitelisted` `` and `` `isMinter` `` are declared without named parameters, making the purpose of the key and value less obvious upon inspection.
contracts/ZEROBASE.sol:L20 - L21

```
mapping(address => bool) public isWhitelisted;
mapping(address => bool) public isMinter;
```

Recommendation

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

Status

This issue has been acknowledged by the team.

5. Custom Errors in require Statements	
Severity: Informational	Category: Gas Optimization
Target: - contracts/ZEROBASE.sol	

Description

In the `ZEROBASE` contract , the contract uses <if (condition) revert() statements and require(condition) 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.

Recommendation

Consider replacing all if-revert statements and require(condition) statements with require(condition, CustomError()) to improve readability and reduce gas consumption.

Status

This issue has been acknowledged by the team.

6. Hardcoded Addresses

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

Description

In the `ZEROBASE` contract, multiple instances of hardcoded addresses were identified:

contracts/ZEROBASE.sol:L20 - L21

```
address public constant LZ_POINT = 0x1a44076050125825900e736c501f859c50fE728c;
address public constant MULTISIG = address(0xffff);
```

Declare hard-coded addresses as immutable variables and initialize them via constructor parameters. This ensures consistency across deployments on different networks and reduces the need to redeploy contracts due to incorrect hard-coded addresses.

Recommendation

Consider declaring hardcoded addresses as immutable variables.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit cdb2a73:

File	SHA-1 hash
src/ZEROBASE.sol	f17139a51206a4fc74d521fc77f78a2e43d38c2f