SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

LORENZO

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Lorenzo-TGE
Version	v3
Type	Solidity
Dates	May 20 2025
Logs	Apr 16 2025; Apr 17 2025; May 20 2025

Vulnerability Summary

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

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	Cross chain replay	Medium	Business Logic	Resolved
2	The code design does not match the documentation	Medium	Business Logic	Resolved
3	Centralization risk	Low	Centralization	Mitigated
4	Missing check for redundant state updates in pauserole	Low	Data Validation	Resolved
5	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledge
6	Use of floating pragma	Informational	Configuration	Acknowledge

2.2 Notable Findings

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

1. Cross chain replay	
Severity: Medium	Category: Business Logic
Target: - contracts/airdrop/Airdrop.sol	

Description

In the `hardhat.config.ts` file, multiple chain rpc urls are configured. If the same `merkle root` is used on different chains, an attacker can use the same `leaf` node to claim airdrops on different chains.

contracts/airdrop/Airdrop.sol:L43 - L56

```
function airdrop(uint256 roundId, uint256 amount, bytes32[] calldata proof) external {
    require(roots[roundId] != bytes32(0), "root not set");

    bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(
        roundId, msg.sender, amount
    ))));
    bool verified = MerkleProof.verify(proof, roots[roundId], leaf);
    require(verified, "verify failed");
    require(!claimedAirdrops[leaf], "already claimed");
    claimedAirdrops[leaf] = true;

    SafeERC20.safeTransfer(IERC20(bankAddress), msg.sender, amount);
    emit AirdropClaimed(roundId, msg.sender, amount);
}
```

Recommendation

It is recommended to add the `chainId` variable during calculation

Status

The team has resolved this issue in commit 602fa6f.

2. The code design does not match the documentation	
Severity: Medium	Category: Business logic
Target: - contracts/tge/TgeContract.sol	

Description

According to the contract design document, the marketing role will divide 3% of the tokens into two parts and issue them once every 3 months. However, the current contract code will unlock all tokens while `claimStartTs` is reached. even if `ROLE_MARKETING_TGEUNLOCK_PERCENT` is changed to 0, the contract still does not conform to the claim cycle of 3 months as stated in the document, which results in the contract running inconsistently with the expectation.

contracts/tge/TgeContract.sol:L65 - L69

```
bytes32 public constant ROLE_MARKETING                   = keccak256("ROLE_MARKETING");
// BN marketing role
uint256 public constant ROLE_MARKETING_PERCENT           = 0.03e18; // 3%
uint256 public constant ROLE_MARKETING_CLIFF_MONTHS      = 0; // 0 months
uint256 public constant ROLE_MARKETING_RELEASE_MONTHS    = 6; // 0 months
uint256 public constant ROLE_MARKETING_TGEUNLOCK_PERCENT = 0.03e18; // 3%
```

Recommendation

It is recommended to change the parameter configuration and token unlocking logic according to the contract design document.

Status

The team has resolved this issue in commit da43472.

3. Centralization risk	
Severity: Low	Category: Centralization
Target: - contracts/airdrop/Airdrop.sol - contracts/tge/TgeContract.sol	

Description

There is a privileged account owner in the contract. The owner can set the `merkle root`, change the `bankToken` and pause a certain role.

If `owner’s` private key is compromised, an attacker can modify key variables to gain benefits for themselves

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

The team stated that they will transfer the owner address to a multi-sig account after deployment.

4. Missing check for redundant state updates in pauserole	
Severity: Low	Category: Data Validation
Target: - contracts/tge/TgeContract.sol	

Description

The `pauseRole()` function allows the contract owner to pause or unpause a specific role by updating the `pausedRoles` mapping and emitting a `RolePaused` event. However, it currently lacks a conditional check to ensure that the new state (paused) is different from the current state (`pausedRoles[role]`). As a result, the function may emit misleading events even when no actual state change has occurred.

contracts/tge/TgeContract.sol:L249 - L252

```
function pauseRole(bytes32 role, bool paused) external onlyOwner {
    pausedRoles[role] = paused;
    emit RolePaused(role, paused);
}
```

Recommendation

It is recommended to add a check on the input status.

Status

The team has resolved this issue in commit c03a8ba.

2.3 Informational Findings

5. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/tge/TgeContract.sol	

Description

The `TgeContract` contract inherits from the `OwnableUpgradeable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

6. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.28;
```

All contracts use a floating compiler version `^0.8.28`.

Using a floating pragma `^0.8.28`statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit ae5de9d :

File	SHA-1 hash
contracts/AbstractProxy.sol	4980cb6aca644018c7b0f48ae5a9909aacf4b44d
contracts/BankToken.sol	5687cc1724dcf4d3425444a75ebc31e98aa5d920
contracts/airdrop/Airdrop.sol	fa8ce8e6ebd1edccc1def888cfa6a0707f03084e
contracts/tge/Scheduler.sol	1c21165eea3f28b10835aa8d4581c261e0c9b635
contracts/tge/TgeContract.sol	630650cc0c9cdb53335f724d7a9e6fdea007ec6b

And we audited the files that introduced new features at address 0xb33ef3b9aecc0366c932848d67f381249b3ee762:

File	SHA-1 hash
contracts/tge//MarketingScheduler.sol	ffe6435424945948e2ca34d24bf47f7ef0c3cd89
contracts/tge//Scheduler.sol	f53a302659187b2ec21f7f6725179e7cdcf96b75
contracts/tge//TgeContract.sol	11c2068376b1d5a6435006cdc2587ad27673e69a
contracts/tge/TgeContractV2.sol	f127ee7631642abc83a852a84afd02d2b63f3418
contracts/tge/TgeContractV3.sol	34bd9b79b8a595b2131e71c891c7d0fc93f20374