SECURITY AUDIT REPORT

Elevate-DeFi

DATE

02 January 2025

PREPARED BY

0xTeam.

WEB3 AUDITS

info@0xTeam.Space

0xTeamSpace

OxTeamSpace

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	21 Dec 2024	D.Aditya K.Bob	Initial Audit Report
2.0	28 Dec 2024	D.Aditya K.Bob	Second Audit Report
3.0	02 Jan 2025	D.Aditya K.Bob	Final Audit Report

0xTeam conducted a comprehensive Security Audit on the Elevate GoLangs to ensure the overall code quality, security, and correctness. The review focused on ensuring that the code functions as intended, identifying potential vulnerabilities, and safeguarding the integrity of Elevate's operations against possible attacks.

Report Structure

The report is divided into two primary sections:

1. **Executive Summary** : Provides a high-level overview of the audit findings.
2. **Technical Analysis** : Offers a detailed examination of the GoLang code.

Note :
The analysis is static and exclusively focused on the GoLang code. The information provided in this report should be utilised to understand the security, quality, and expected behaviour of the code.

1.0 Disclaimer

This is a summary of our audit findings based on our analysis, following industry best practices as of the date of this report.However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. The audit focuses on GoLang coding practices and any issues found in the code, as detailed in this report. For a complete understanding of our analysis, you should read the full report. We have made every effort to conduct a thorough analysis, but it's important to note that you should not rely solely on this report and cannot make claims against us based on its contents. We strongly advise you to perform your own independent checks before making any decisions. Please read the disclaimer below for more information.

DISCLAIMER: By reading this report, you agree to the terms outlined in this disclaimer. If you do not agree, please stop reading immediately and delete any copies you have. This report is for informational purposes only and does not constitute investment advice. You should not rely on the report or its content, and 0xTeam and its affiliates (including all associated companies, employees, and representatives) are not responsible for any reliance on this report.The report is provided "as is" without any guarantees. 0xTeam excludes all warranties, conditions, or terms, including those implied by law, regarding quality, fitness for a purpose, and use of reasonable care. Except where prohibited by law, 0xTeam is not liable for any type of loss or damage, including direct, indirect, special, or consequential damages, arising from the use or inability to use this report. The findings are solely based on the GoLang code provided to us.

2.0 Executive Summary

2.1 Overview

0xTeam has meticulously audited the Elevate GoLang project from 01 Oct 2024 to 14 Oct 2024. The primary objective of this audit was to assess the security, functionality, and reliability of the Elevate's before their deployment on the blockchain. The audit focused on identifying potential vulnerabilities, evaluating the contract's adherence to best practices, and providing recommendations to mitigate any identified risks. The comprehensive analysis conducted during this period ensures that the Elevate is robust and secure, offering a reliable environment for its users.

2.2 Scope

The scope of this audit involved a thorough analysis of the Elevate GoLang, focusing on evaluating its quality, rigorously assessing its security, and carefully verifying the correctness of the code to ensure it functions as intended without any vulnerabilities.

Files in Examination:

Code Language	Golang
In-Scope	• /elevated/proofs-DeFi

OUT-OF-SCOPE: External GoLang code, other imported code.

2.3 Audit Summary

Name	Verified	Audited	Vulnerabilities
Elevate	Yes	Yes	Refer Section 5.0

2.4 Vulnerability Summary

● High	● Medium	● Low	● Informational
0	2	1	0

● High ● Medium ● Low ● Informational

2.5 Recommendation Summary

Severity

Issues	● High	● Medium	● Low	● Informational
Open	0	3	2	0
Resolved		2	2	
Acknowledged		1		
Partially Resolved				

- **Open**: Unresolved security vulnerabilities requiring resolution.
- **Resolved**: Previously identified vulnerabilities that have been fixed.
- **Acknowledged**: Identified vulnerabilities noted but not yet resolved.
- **Partially Resolved**: Risks mitigated but not fully resolved.

3.0 Checked Vulnerabilities

We examined GoLang for widely recognized and specific vulnerabilities. Below are some of the common vulnerabilities considered.

Category	Check Items
Source Code Review	➔ Reentrancy Vulnerabilities ➔ Ownership Control ➔ Time-Based Dependencies ➔ Gas Usage in Loops ➔ Transaction Sequence Dependencies ➔ Style Guide Compliance ➔ EIP Standard Compliance ➔ External Call Verification ➔ Mathematical Checks ➔ Type Safety ➔ Visibility Settings ➔ Deployment Accuracy ➔ Repository Consistency
Functional Testing	➔ Business Logic Validation ➔ Feature Verification ➔ Access Control and Authorization ➔ Escrow Security ➔ Token Supply Management ➔ Asset Protection ➔ User Balance Integrity ➔ Data Reliability ➔ Emergency Shutdown Mechanism

4.0 Techniques , Methods & Tools Used

The following techniques, methods, and tools were used to review all the Elevate

- **Structural Analysis:**
  This involves examining the overall design and architecture of the Elevate. We ensure that the code is logically organised, scalable, and follows industry best practices. This step is crucial for identifying potential structural issues that could lead to vulnerabilities or maintenance challenges in the future.

- **Static Analysis:**
  Static analysis is conducted using automated tools to scan the contract's codebase for common vulnerabilities and security risks without executing the code. This process helps identify issues such as reentrancy, arithmetic errors, and potential denial-of-service (DOS) vulnerabilities early on, allowing for quick remediation.

- **Code Review / Manual Analysis:**
  A manual, in-depth review of Elevate's code is performed to verify the logic and ensure it matches the intended functionality as described in the project's documentation. During this phase, we also confirm the findings from the static analysis and check for any additional issues that may not have been detected by automated tools.

- **Dynamic Analysis:**
  Dynamic analysis involves executing the Elevate in various controlled environments to observe its behaviour under different conditions. This step includes running comprehensive test cases, performing unit tests, and monitoring gas consumption to ensure the code operates efficiently and securely in real-world scenarios.

Note: The following values for "Severity" mean:

- High: Direct and severe impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal impact on the funds or the protocol's main functionality.
- Informational: Suggestions related to good coding practices and gas efficiency.

5.0 Technical Analysis

Medium

Issue#1 [Resolved]

[M-1] Freezing of Validator Bond Delegations

Severity

Medium

Issue Description

A flaw in the interaction between **tokenized shares, validator bond shares, and delegation mechanisms** within the staking module allows an attacker to **manipulate ValidatorBondShares**, potentially disrupting other delegators.

The issue arises because:

1. **Tokenized shares cannot be marked as validator bond**, but it is still possible to **transfer tokens to a delegator with an existing validator bond delegation**.

2. The **RedeemTokensForShares function** currently permits validator bond delegations but **fails to update ValidatorBondShares** accordingly.

Exploitation Scenario

- A malicious delegator can delegate a small amount and mark it as ValidatorBond.
- They then convert this into a large number of LSM tokens, redeem them, and undelegate the funds.
- This artificially decreases the validator's ValidatorBondShares.
- As a result, other delegators with ValidatorBond delegations to the same validator may become unable to undelegate their funds, since SafelyDecreaseValidatorBond would fail due to insufficient ValidatorBondShares.
- This attack requires minimal capital but can cause significant disruptions in the staking process.

Issue Impact

Freezing of Validator Bond Delegations:

- Delegators with validator bond delegations **may be unable to undelegate their funds** due to insufficient ValidatorBondShares.
- This disrupts the normal **staking and unstaking process**, affecting liquidity for affected delegators.

Manipulation of ValidatorBondShares:

- A malicious actor can **artificially reduce** a validator's ValidatorBondShares with minimal capital.
- This impacts other delegators who rely on validator bond delegations, creating an unfair staking environment.

Potential Staking Disruptions:

- The exploit could **destabilize validator operations**, leading to staking inefficiencies or unintended penalties.
- Affected delegators may **lose confidence in the staking mechanism**, reducing overall participation.

Economic and Security Risks:

- Attackers can **intentionally disrupt validator bonding**, causing potential harm to the network's integrity.
- If widely exploited, this could **discourage users from participating in validator bonding**, weakening the staking security model.

Recommended Mitigation Steps

Enhance Redemption Validation in RedeemTokensForShares:

- Implement a **check to determine if the destination delegation is a validator bond** before allowing redemption.
- Either **block the redemption process entirely** or **ensure that ValidatorBondShares are correctly updated** to reflect the changes.

.

Issue#2 [Acknowledged]

[M-2] Lack of Liquid Staking Accounting in CreateValidator

Severity

Medium

Issue Description

The **CreateValidator function** does not incorporate the **liquid staking accounting mechanisms** present in the **Delegate function**.

In the **Delegate function**, specific checks and updates are performed when a **liquid staking provider initiates a delegation**:

- Staked tokens are converted into an **equivalent number of shares** in the validator.
- The **global liquid stake** and **validator liquid shares** are **updated accordingly** to maintain correct accounting.

However, during **validator creation**, when the initial stake is self-delegated:

- **No such checks or updates occur** for liquid staking.
- This results in **inconsistent tracking**, as the **global liquid staking cap and per-validator cap remain unchanged**.
- This creates a potential discrepancy between the actual **liquid stake in the system** and what is recorded, leading to potential **accounting and governance issues**.

Issue Impact

Inconsistent Liquid Staking Limits:

- The global and per-validator liquid staking caps do not reflect the actual liquid stake amounts.
- This could lead to violations of staking constraints, potentially affecting network security.

Potential Exploits in Liquid Staking Systems:

- Validators could circumvent staking caps, enabling over-exposure of liquid stake beyond intended limits.
- This could lead to imbalanced validator participation and unintended dominance by certain validators.

Discrepancies in Staking Accounting:

- The network may fail to enforce liquid staking limits properly, allowing

incorrect stake distributions.
- This could lead to unexpected behavior in governance, slashing mechanisms, or stake-weighted decisions.

Recommended Mitigation Steps

Integrate Liquid Staking Bookkeeping into CreateValidator:

- Ensure that **the same accounting checks and updates from Delegate** are applied when creating a validator.
- Convert the **initial self-stake into shares properly** and **update the global and per-validator liquid staking caps accordingly**.

Issue#3 [Resolved]

[M-3] Inconsistencies in Slash Redelegation

Severity

Medium

Issue Description

A flaw in the **SlashRedelegation** function leads to inconsistencies when handling **redelegated stakes involving a validator bond**.

In the **Cosmos SDK**, when a validator is slashed for an infraction:

- **Penalties apply to both direct delegations and redelegations**.
- **Redelegated tokens are penalized** by **unbonding an equivalent slash amount**.

Issues Identified:

1. **Validator Bond Shares Not Adjusted:**
    - If the redelegated stake involves a **validator bond**, the **validatorBondShare** should also be reduced.
    - Since the validatorBond represents the **validator's self-staked commitment**, penalties must also **impact this bond** when slashing occurs.
    - **Failing to adjust validatorBondShare** leaves the **validator's total staked shares artificially inflated**.
2. **LiquidShares Not Updated After Slashing:**
    - If the **delegatorAddress** is a **liquid staker**, the **Unbond function** is triggered with `sharesToUnbond`.
    - This **reduces the redelegated balance** but **does not update LiquidShares** in the validator's record.
    - As a result, the **validator's liquid stake remains artificially high**, leading to **incorrect staking calculations**.

```
func (k Keeper) SlashRedelegation(ctx context.Context, srcValidator types.Validator,
    |→ redelegation types.Redelegation,
    infractionHeight int64, slashFactor math.LegacyDec,
) (totalSlashAmount math.Int, err error) {
    [...]
    tokensToBurn, err := k.Unbond(ctx, delegatorAddress, valDstAddr, sharesToUnbond)
    if err != nil {
             return math.ZeroInt(), err
    }
     [...]
}
```

Issue Impact

Validator Bond Inflation:

- Validators may retain more self-staked shares than they should, leading to an overstated commitment.
- This weakens the security model, as validators appear to be staking more than they actually are.

Liquid Staking Inconsistencies:

- Slashed liquid stake is not properly reflected, making staking data unreliable.
- This could allow incorrect governance weight calculations and distort stake-based rewards.

Potential for Exploitation:

- A validator could retain an artificially high stake post-slashing, potentially avoiding penalties they should incur.
- Liquid stakers may not properly account for slashed amounts, leading to incorrect economic balances.

Recommended Mitigation Steps

Reduce validatorBondShare When Slashing Occurs

- Ensure that **when a validator bond is involved in a redelegation**, slashing appropriately reduces **validatorBondShare**.

Update LiquidShares When Unbonding Liquid Stake

- Modify `SlashRedelegation` to correctly **adjust the LiquidShares value** in the validator’s records when slashed.

Low

Issue#4 [Resolved]

[L-1] Improper Storage Utilization

Severity

Low

Issue Description

msg_server::TokenizeShares does not ensure that the share is not zero. It is possible for the amount to be positive while the share is zero, resulting in the creation and storage of records without any actual shares.

```
func (k msgServer) TokenizeShares(goCtx context.Context, msg
*types.MsgTokenizeShares)
    |→ (*types.MsgTokenizeSharesResponse, error) {
    [...]
    shares, err := k.ValidateUnbondAmount(
                ctx, delegatorAddress, valAddr, msg.Amount.Amount,
        )
        if err != nil {
            return nil, err
        }
   [...]
}
```

Recommended Mitigation Steps

Add a check after the call to **ValidateUnbondAmount** to ensure that **shares** is greater than zero.

Issue#5 [Resolved]

[L-2] Insufficient Error Handling

Severity

Low

Issue Description

ValidatorBond , **RedeemTokensForShares** , and **UnbondValidator** in **msg_server** lack sufficient error handling for certain function calls (**SetDelegation** , **bondedTokensToNotBonded** , and **jailValidator** , respectively). Missing error checks may affect delegation states in the staking module, leading to inconsistencies in the bonded and not-bonded pools and impacting validator management.

Recommended Mitigation Steps

Utilize a linter, such as **errcheck** , which checks for unchecked errors in Go code. **errcheck** helps detect cases where error handling is missing.

6.0 Auditing Approach and Methodologies Applied

The GoLang Code Base was audited using a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of the code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analysing the overall architecture of the GoLang Code Base and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behaviour and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the GoLang community and industry experts. This ensures that the GoLang Code Base is secure, reliable, and efficient.

Our audit team followed OWASP and (GoLang) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve Elevate's security and performance.

Throughout the audit of the GoLangs, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behaviour. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the GoLang was secure, reliable, and optimised for performance.

6.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the GoLang Code Bases to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analysed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the GoLang was secure and reliable.

7.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the Elevate Project and methods for exploiting them. 0xTeam recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while 0xTeam considers the major security vulnerabilities of the analysed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Elevate GoLang Code Base described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change. 0xTeam makes no undertaking to supplement or update this report based on changed circumstances or facts of which 0xTeam becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that 0xTeam use certain software or hardware products manufactured or maintained by other vendors. 0xTeam bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, 0xTeam does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by 0xTeam for the exclusive benefit of Elevate and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between 0xTeam and Elevate governs the disclosure of this report to all other parties including product vendors and suppliers.