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	2	1	0
Resolved		2	1	
Acknowledged				
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] Unchecked Fee Manipulation in x/liquidity Module

Severity

Medium

Issue Description

A critical weakness in the **x/liquidity module** arises from the **lack of restrictions on fee parameter modifications**. Currently, there are **no enforced limits on fee values or frequency of updates**, allowing potential **exploitation by privileged roles**.

This vulnerability **enables fee adjustments that could be leveraged for financial gain**, including:

- **Front-Running High-Value Trades:** Attackers monitoring the mempool can **increase fees just before a large trade executes**, capturing a portion of its value.
- **Unfair Profit Distribution:** The **PairCreatorSwapFeeRatio** can be adjusted unfairly, favoring the pair creator over other liquidity providers.
- **Bait-and-Switch Tactics:** Users may **commit assets under low fees** only to be subjected to **unexpectedly high fees after execution**.

Issue Impact

- **Financial Loss for Traders:** Unexpected fee increases can lead to **substantial or complete financial losses**.
- **Liquidity Provider Windfall:** Liquidity providers can **manipulate fees during peak activity** to extract unfair profits.
- **Erosion of User Trust:** Fee exploitation would **severely damage platform credibility** and deter participation.

While **admin access controls** (e.g., CreatePair and UpdatePairSwapFee) provide some security, they **do not mitigate the core issue of unrestricted fee adjustments**. Furthermore, **centralized control over fees** introduces additional risks.

Recommended Mitigation Steps

1 **Enforce Fee Limits:**

- Establish **strict upper and lower bounds** for `SwapFeeRate` and `PairCreatorSwapFeeRatio` within validation functions.
- Implement **dynamic fee limits** to adjust parameters based on market conditions while preventing excessive manipulation.

2 **Introduce Rate Limiting or Timelocks:**

- **Restrict how often fees can be changed** by enforcing **minimum time intervals** between modifications.
- Alternatively, **introduce time delays** between the proposal and execution of fee changes, ensuring traders and liquidity providers have time to react.

3 **Decentralized Governance Mechanism:**

- Implement a **governance model** where **token holders can propose and vote on fee adjustments**, ensuring community-driven oversight.
- Reduce reliance on **centralized admin controls**, which may introduce bias or manipulation risks.

4 **Enhance Transparency:**

- Provide **detailed and publicly accessible documentation** outlining **fee structures, modification rules, and governance processes**.
- **Proactively notify the community** before implementing fee changes, ensuring all participants are aware and can plan accordingly.

Issue#2 [Resolved]

[M-2] Centralized Control and Elevated Admin Privileges

Severity

Medium

Issue Description

The project currently faces a **security risk due to centralized control and elevated admin privileges**.

Centralized Control:

A limited group of **AdminAccounts** holds **unrestricted authority** over the system, allowing them to:

- Modify **asset permissions**, effectively controlling access.
- **Whitelist accounts** for bank transfers, bypassing existing restrictions.
- **Assign new admin accounts**, further reinforcing centralized control.

Additionally, these **admin accounts** possess **privileged execution rights** across multiple modules, including but not limited to:

- **x/airdrop:** CreateNewCampaigns
- **x/coinfactory:** Mint, Burn, ForceTransferAssets
- **x/did:** CreateDidDocuments
- **x/guard:** UpdateAccountPrivileges (Single, Batch, Grouped), UpdateGuardTransferCoins
- **x/liquidity:** CreatePair, UpdatePairSwapFee, CreatePool, CreateRangedPool
- **x/lpfarm:** CreatePrivatePlan
- **x/token:** UpdateRestrictedCollectionNftImage (Single, Grouped), UpdateGuardSoulBondNftImage
- **x/txfees:** CreateFeeToken, UpdateFeeToken, DeleteFeeToken

This is **not an exhaustive list** but represents key areas where **admin control is disproportionately high**.

Elevated Admin Privileges:

The **CheckHasAuthz function** is implemented across multiple modules, granting **admin accounts broader privileges than regular users**, introducing significant risks.

Issue Impact

Single Point of Failure: If an **admin account is compromised**, the attacker could:

- **Manipulate permissions** to gain unauthorized access to critical resources.
- **Whitelist malicious accounts**, enabling fraudulent transactions.
- **Escalate privileges** by adding more compromised accounts as admins.

Unauthorized Actions: While designed for **administrative convenience**, these elevated privileges can lead to:

- **Unintentional or malicious misuse**, potentially resulting in **a full system compromise**.
- **Unauthorized fund transfers** or **asset mismanagement**, leading to **financial losses**.

Root Cause Analysis:

The issue arises from the **existing system architecture**, where admin accounts are granted **broad control and decision-making power**. While this approach simplifies development and maintenance, it **conflicts with security best practices** and increases exposure to **centralization risks**.

Recommended Mitigation Steps

Decentralized Control:

- Implement a multi-signature (multisig) model requiring consensus for sensitive administrative actions.
- Establish separate consensus rules for adding new admins, reducing unilateral decision-making risks.
- Introduce role-based access control (RBAC) with multi-signature approvals for critical functions.
- Isolate approval processes for whitelisting accounts, ensuring checks and balances.

Enhance Authorization Checks:

- Strengthen authorization logic for admin actions, requiring additional validation before execution.
- Log and audit admin activities to maintain accountability and detect potential misuse.

Low Severity

Issue#3 [Resolved]

[L-1] Weak Input Validation and Absence of FeeToken Liveness Checks

Severity
Low

Issue Description

The FeeToken mechanism presents two major weaknesses that could lead to disruptions in the fee payment process:

1. **Insufficient Input Validation:**
   - The CreateFeeToken and UpdateFeeToken functions do not adequately validate whether the pairId exists or if the denom is valid.
   - This allows attackers to create FeeTokens associated with invalid or illiquid pools, leading to potential transaction failures and system instability.
2. **Lack of FeeToken Liveness Checks:**
   - The system does not verify whether a FeeToken remains valid at the time of transaction execution.
   - This means transactions can use FeeTokens linked to pools that have become illiquid or significantly devalued, causing unexpected behavior.

Issue Impact

- **Denial of Service (DoS):** Attackers could exploit this weakness by spamming transactions with invalid FeeTokens, overwhelming the network and disrupting legitimate transactions.
- **Unexpected Fee Fluctuations:** Since there are no liveness checks, a FeeToken could be linked to a pool that experiences extreme price changes, leading to unpredictable transaction fees.
- **Increased Attack Surface:** Given the lower degree of decentralization in the system, the cost of executing such attacks is relatively low, making them more feasible.

Recommended Mitigation Steps

Enhance Input Validation:

- Modify CreateFeeToken and UpdateFeeToken to validate:
  - That the specified pairId exists in the LiquidityModule.
  - That the denom is a valid and operational token.

Implement FeeToken Liveness Checks:

- Introduce an `active: bool` field in the FeeToken structure.
- In the BeginBlocker of x/txfees, verify the operational status of FeeToken-associated pools by checking liquidity levels and functionality.
- Deactivate FeeTokens linked to non-operational or depleted pools.

Enhanced CreateFeeToken Validation:

```
func (k Keeper) CreateFeeToken(ctx sdk.Context, msg *types.MsgCreateFeeToken) error
{
   // Validate that the Pair ID exists
   if !k.liquidityKeeper.HasPair(ctx, msg.PairId) {
       return sdkerrors.Wrapf(types.ErrInvalidPairID, "pair ID %d does not exist",
msg.PairId)
   }

   // Validate that the Denom exists
   if !k.bankKeeper.HasSupply(ctx, msg.Denom) {
       return sdkerrors.Wrapf(types.ErrInvalidDenom, "denom %s does not exist",
msg.Denom)
   }

   feeToken := types.FeeToken{
       Denom:  msg.Denom,
       PairId: msg.PairId,
       Active: true, // New liveness field
   }

   k.SetFeeToken(ctx, feeToken)
   return nil
}
```

Enhanced UpdateFeeToken Validation:

```
func (k Keeper) UpdateFeeToken(ctx sdk.Context, msg *types.MsgUpdateFeeToken) error
{
   // Ensure the FeeToken exists
   feeToken, found := k.GetFeeToken(ctx, msg.Denom)
```

```
    if !found {
        return sdkerrors.Wrapf(types.ErrFeeTokenNotFound, "FeeToken %s does not
exist", msg.Denom)
    }

    // Validate that the new Pair ID is valid
    if !k.liquidityKeeper.HasPair(ctx, msg.PairId) {
        return sdkerrors.Wrapf(types.ErrInvalidPairID, "pair ID %d does not exist",
msg.PairId)
    }

    feeToken.PairId = msg.PairId
    k.SetFeeToken(ctx, feeToken)
    return nil
}
```

New Liveness Check in BeginBlocker

```
func (k Keeper) BeginBlocker(ctx sdk.Context) {
    feeTokens := k.GetAllFeeTokens(ctx)
    for _, feeToken := range feeTokens {
        // Check if the associated liquidity pool is operational
        if !k.liquidityKeeper.IsPoolActive(ctx, feeToken.PairId) {
            feeToken.Active = false
            k.SetFeeToken(ctx, feeToken)
        }
    }
}
```

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.