Elevate Staking

Security Assessment

February 26th, 2021

For :
Elevate DeFi

By :

Kun Zhao @ CertiK
kun.zhao@certik.org

Dan She @ CertiK
dan.she@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Elevate DeFi
Description	DeFi
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	2e60ae51ffa4520d0e827fe1037803deb529014c c3b4ac6f23087364d7c890d5419abbae6d62f580 a0cf249f52bc82941912e2176b4e94d799bd6904 2af66e101d4e711fc06568787d9e4ec155688a95

Audit Summary

Delivery Date	Feb. 26th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 18th, 2021 - Feb. 26th, 2021

Vulnerability Summary

Total Issues	12
Total Critical	0
Total Major	4
Total Minor	0
Total Informational	8

Executive Summary

This report has been prepared for **Elevate** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

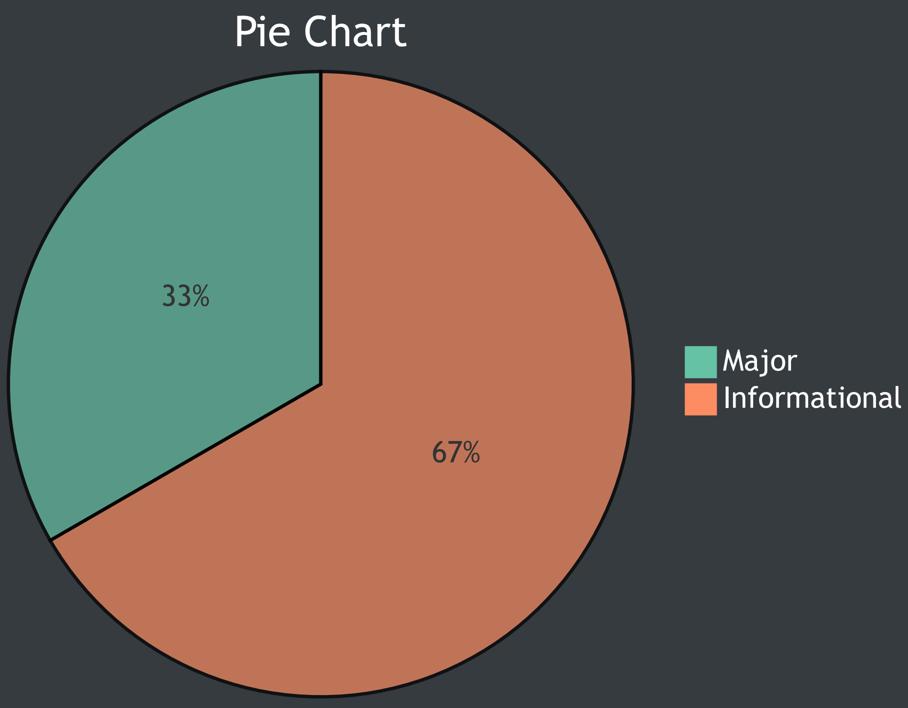
- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

There are a few depending injection contracts in current project:
`stakingToken` , `distributionToken` and `reflectiveTreasury` for contract **ReflectiveStake**;
`token` and `beneficiary` for contract **ReflectiveTreasury**.
They are not in the scope of this audit. We assume these contracts are valid and non-vulnerable actors, and implementing proper logic to collaborate with current project.

File in Scope

ID	Contract	SHA-256 Checksum
ITR	ITREASURY.sol	f68daa4bf757bcc84ce45044618cc72cc61621e052b1616fdec514bcd2b27f68
RFS	ReflectiveStake.sol	9a528764a5bfa17b2a0f85c3f7877bdf57fbda80c38bba29d2ba774783de6960
RFT	ReflectiveTreasury.sol	b3a65e8065207d807943eea41779f2296d6fcc4009c8c734b904d194d3f7ea83
TKP	TokenPool.sol	8d8049a91f5c92e09d08ed0f9fd8fb69c0bd8959fc6031a19fb9aa794f391df8

Findings



ID	Title	Type	Severity	Resolved
RFS-01	Inconsistent Coding Style	Coding Style	Informational	✓
RFS-02	Missing Error Message	Optimization	Informational	✓
RFS-03	Proper Usage of “public” And “external” Type	Optimization	Informational	✓
RFS-04	Redundant Arguments	Optimization	Informational	✓
RFS-05	Missing Checks For Reentrancy	Logic Issue	Major	✓
RFS-06	Logics of Staking	Logic Issue	Informational	
RFS-07	Code Simplicity	Optimization	Informational	
RFT-01	Proper Usage of “public” And “external” Type	Optimization	Informational	✓
RFT-02	Missing Checks For Reentrancy	Logic Issue	Major	✓
RFT-03	Code Simplicity	Optimization	Informational	
RFT-04	Centralization Risks	Logic Issue	Major	
TKP-01	Centralization Risks	Logic Issue	Major	

RFS-01: Inconsistent Coding Style

Type	Severity	Location
Coding Style	Informational	ReflectiveStake.sol: L63-64

Description:

Inconsistent coding style of checking positive integers.

```
        require(bonusPeriodSec_ != 0, 'TokenGeyser: bonus period is zero');
        require(initialSharesPerToken > 0, 'TokenGeyser: initialSharesPerToken is zero');
```

Recommendation:

We recommend changing line 63 to

```
        require(bonusPeriodSec_ > 0, 'TokenGeyser: bonus period is zero');
```

Alleviation:

The development team heeded our advice and resolved this issue in commit c3b4ac6f23087364d7c890d5419abbae6d62f580.

RFS-02: Missing Error Message

Type	Severity	Location
Optimization	Informational	ReflectiveStake.sol: L69

Description:

An error message is missing in the `require` call at line 69. Error messages could help function callers locate errors more efficiently.

Recommendation:

We recommend adding error messages for the check of `_unlockedPool.token() == _reflectiveTreasury.token()`.

Alleviation:

The development team heeded our advice and resolved this issue in commit c3b4ac6f23087364d7c890d5419abbae6d62f580.

RFS-03: Proper Usage of “public” And “external” Type

Type	Severity	Location
Optimization	Informational	ReflectiveStake.sol: L80, L132, L224, L229, L289 and L304

Description:

Functions defined at the aforementioned lines are declared as `public` while they are never called internally within the contract. Functions which are never called internally within the contract should have `external` visibility.

Recommendation:

We recommend changing the visibility of functions at the aforementioned lines to `external` .

Alleviation:

The development team heeded our advice and resolved this issue in commit c3b4ac6f23087364d7c890d5419abbae6d62f580.

RFS-04: Redundant Arguments

Type	Severity	Location
Optimization	Informational	ReflectiveStake.sol: L88

Description:

Private function `_stakeFor` is only used in function `stake`. Considering `staker` and `beneficiary` are both `msg.sender`, they can be removed from the argument list. In fact, `_stakeFor` is an unnecessary wrapped function.

Recommendation:

We recommend moving the implementations of `_stakeFor` to `stake`, replacing `staker` and `beneficiary` by `msg.sender` and removing `_stakeFor`.

Alleviation:

The development team heeded our advice and resolved this issue in commit c3b4ac6f23087364d7c890d5419abbae6d62f580.

RFS-05: Missing Checks For Reentrancy

Type	Severity	Location
Logic Issue	Major	ReflectiveStake.sol: L88, L136 and L251

Description:

Function `_stakeFor` , `unstake` and `updateAccounting` update states after external calls and thus are vulnerable to reentrancy attack.

Recommendation:

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation:

The development team heeded our advice and resolved this issue in commit c3b4ac6f23087364d7c890d5419abbae6d62f580 and 2af66e101d4e711fc06568787d9e4ec155688a95.

RFS-06: Logics of Staking

Type	Severity	Location
Logic Issue	Information	ReflectiveStake.sol: L95

Description:

The tokens being staked need to apply the same fee implementation as that in `_applyFee`. Otherwise applying fees when users stake has uncertain effects on users' stakes.
For example, if `_stakingPool.token` is a standard ERC20 token:

1. Initial state: `_stakingPool.balance` is 0, `totalStakingShares` is 0 and assume `_initialSharesPerToken` is 1. We assume `_stakingPool.balance` will not be changed by transactions other than staking and unstaking.
2. The first user A staked 1000 tokens. After applying fees, A's `stakingShares` is $1000 * 99\% * 1$ (line #99) = 990. Now `_stakingPool.balance` is 1000 and `totalStakingShares` is 990.
3. The second user B staked 1000 tokens. After applying fees, B's `stakingShares` is $990 * (1000 * 99\%)/1000$ (line #98) = 980. Now `_stakingPool.balance` is 2000 and `totalStakingShares` is 1970. Although user A and B staked the same amount of tokens, they have different shares.
4. After the third step (and after the lock time), user A is allowed to unstake $2000 * 990/1970$ (line #141) = 1005 tokens, which is even more than the amount A staked. The problem is the balance left is not enough for B to unstake at that moment.

In general, the staking process is like:

stake amount	mintedStakingShares	totalStakingShares	stakingPool
A	$0.99Ar$	$0.99Ar$	A
B	$0.99B(0.99r)$	$0.99Ar + 0.99^2Br$	$A + B$
C	$0.99C\frac{0.99A+0.99^2B}{A+B}r$	$0.99Ar + 0.99^2Br + 0.99^2Cr\frac{A+0.99B}{A+B}$	$A + B + C$

(r = _initialSharesPerToken, and assuming staking times are quite close)

The last depositor will always own less than his deposit amount at the moment when he deposits (his instant asset approximation is $\frac{[mintedStakingShares]*[stakingPool]}{[totalStakingShares]}$, which is less than the [stake amount] for non-first depositor).

Alleviation:

(**Elevate DeFi Team - Response**)
The token being staked has a built in transaction fee. The fee adjustment in the staking contract is meant to account for this so that it reflects the amount staked by the user accurately.

(**CertiK Team - Update**)

If `_stakingPool.token` has the same fee (1%) implementation as that in `_applyFee`, the example would become:

1. Initial state: `_stakingPool.balance` is 0, `totalStakingShares` is 0 and assume `_initialSharesPerToken` is 1. We assume `_stakingPool.balance` will not be changed by any transaction except for staking and unstaking.
2. The first user A staked 1000 tokens. After applying fees, A's `stakingShares` is $1000 * 99\% * 1$ (line #99) = 990. Now `_stakingPool.balance` is $1000 - 1000 * 1$ = 990 and `totalStakingShares` is 990.
3. The second user B staked 1000 tokens. After applying fees, B's `stakingShares` is $990 * (1000 * 99\%)/990$ (line #98) = 990. Now `_stakingPool.balance` is $990 + (1000 - 1000 * 1\%)$ = 1980 and `totalStakingShares` is 1980. User A and B have the same number of shares now.
4. After the third step (and after the lock time), user A is allowed to unstake $1980 * 990/1980$ (line #141) = 990 tokens, which is reasonable.

If it works in this way, the rule of staking would be fair to users.

RFS-07: Code Simplicity

Type	Severity	Location
Optimization	Informational	ReflectiveStake.sol: L126-129

Description:

The implementation of function `_applyFee` can be simplified for lower gas costs.

Recommendation:

We recommend simplying `_applyFee` from

```
function _applyFee(uint256 amount) internal pure virtual returns (uint256) {
    uint256 tFeeHalf = amount.div(200);
    uint256 tFee = tFeeHalf.mul(2);
    uint256 tTransferAmount = amount.sub(tFee);
    return tTransferAmount;
}
```

to

```
function _applyFee(uint256 amount) internal pure virtual returns (uint256) {
    return amount.sub(amount.div(100));
}
```

Alleviation:

(**Elevate DeFi Team - Response**)
This cannot be simplified as it needs to match exactly the logic from the token that is being staked.

RFT-01: Proper Usage of “public” And “external” Type

Type	Severity	Location
Optimization	Informational	ReflectiveTreasury.sol: L33, L40, L47 and L97

Description:

Functions defined at the aforementioned lines are declared as `public` while they are never called internally within the contract. Functions which are never called internally within the contract should have `external` visibility.

Recommendation:

We recommend changing the visibility of functions at the aforementioned lines to `external`.

Alleviation:

The development team heeded our advice and resolved this issue in commit a0cf249f52bc82941912e2176b4e94d799bd6904.

RFT-02: Missing Checks For Reentrancy

Type	Severity	Location
Logic Issue	Major	ReflectiveTreasury.sol: L61 and L80

Description:

Function `deposit` and `withdraw` update states after external calls and thus are vulnerable to reentrancy.

Recommendation:

We recommend using OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation:

The development team heeded our advice and resolved this issue in commit a0cf249f52bc82941912e2176b4e94d799bd6904.

RFT-03: Code Simplicity

Type	Severity	Location
Optimization	Informational	ReflectiveTreasury.sol: L70-75

Description:

The implementation of function `_applyFee` can be simplified for lower gas costs.

Recommendation:

We recommend simplying `_applyFee` from

```
function _applyFee(uint256 amount) internal pure virtual returns (uint256) {
    uint256 tFeeHalf = amount.div(200);
    uint256 tFee = tFeeHalf.mul(2);
    uint256 tTransferAmount = amount.sub(tFee);
    return tTransferAmount;
}
```

to

```
function _applyFee(uint256 amount) internal pure virtual returns (uint256) {
    return amount.sub(amount.div(100));
}
```

Alleviation:

(**Elevate DeFi Team - Response**)
This cannot be simplified as it needs to match exactly the logic from the token that is being staked.

RFT-04: Centralization risks

Type	Severity	Location
Logic Issue	Major	ReflectiveTreasury.sol: L80

Description:

Function `withdraw` at the aforementioned lines allows the owner to drain all tokens from contracts.

Recommendation:

We recommend the team to review the design and ensure minimum centralization risk. One of our recommendations is to remove the function `withdraw`.

Alleviation:

(**Elevate DeFi Team - Response**)
Contract ownership will be revoked once configured and shown to be working properly.

TKP-01: Centralization risks

Type	Severity	Location
Logic Issue	Major	TokenPool.sol: L23

Description:

Function `transfer` at the aforementioned lines allows the owner to drain all tokens from contracts.

Recommendation:

We recommend the team to review the design and ensure minimum centralization risk. One of our recommendations is to remove the function `transfer`.

Alleviation:

(**Elevate DeFi Team - Response**)
This contract is created by ReflectiveStake.sol, which has no possibility to call the transfer method outside of its defined scope.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

: Issue resolved

: Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

: Issue partially resolved. Not all instances of an issue was resolved.