PROOF

ELONGATE TOKEN SMART CONTRACT SECURITY AUDIT REPORT

Disclaimer

This is a limited report of findings based on an analysis of industry best practices as of the date of this report regarding cybersecurity vulnerabilities and issues in smart contract frameworks and algorithms, the details of which are detailed in this report. stated in the report. To get the full picture of our analysis, it's important to read the full report. Although we have conducted our analysis and have done our best to prepare this report, you should not rely on this report and cannot claim against us based on what it does or does not say or how it was produced. It is important to do your own research before making any decisions. This is explained in more detail in the following disclaimer. Please be sure to read to the end.

Security analytics are based solely on smart contracts. Application or process security not checked. Product code not reviewed.

Table of Contents

Executive Summary

Objectives

Proof Audit, carried out an audit of ELONGATE TOKEN, specifically their BEP20 token. The project is based on the BNB Chain Network. We reviewed documentation which helped with understanding the functions of their code. Our findings in the audit ranged from minor to critical.

Project Info

Audited project

ELONGATE TOKEN

Deployer Address

0x6FDC71505C02bFc1b4302F7a9821eFFC1fFa2cec

Contract Address

0x2A9718defF471f3Bb91FA0ECEAB14154F150a385

Blockchain

BNB Chain

Project website:

https://www.elongate.cc/

Methodology

During the audit process, we inspected the repository thoroughly, using a line-by-line code read through to review vulnerabilities, quality of the code and adherence to best practices and specifications. We used Computer-Aided Verification to support the audit process.

Our auditing process is as follows:

1. **Code Review:**
   A review of the scope, specifications and documentation provided to ensure an in depth understanding of the purpose and functionality of the relevant smart contracts.
2. **Automated Analysis:**
   A series of reviews carried out with the use of automated tools. These reviews serve as a basis for further manual analysis and provide relevant visualizations of the code.
3. **Testing & Manual Review of Code:**
   Test coverage analysis and a line-by-line read through of the project code in order to identify vulnerabilities, errors and weaknesses in code quality.
4. **Specification Comparison:**
   A review of the code against the specifications provided to ensure that the code operates as is intended.
5. **Best Practices Review:**
   A review of the smart contracts to identify potential improvements in effectiveness, efficiency, and maintainability, with a focus on adherence to industry best practices.

Scope

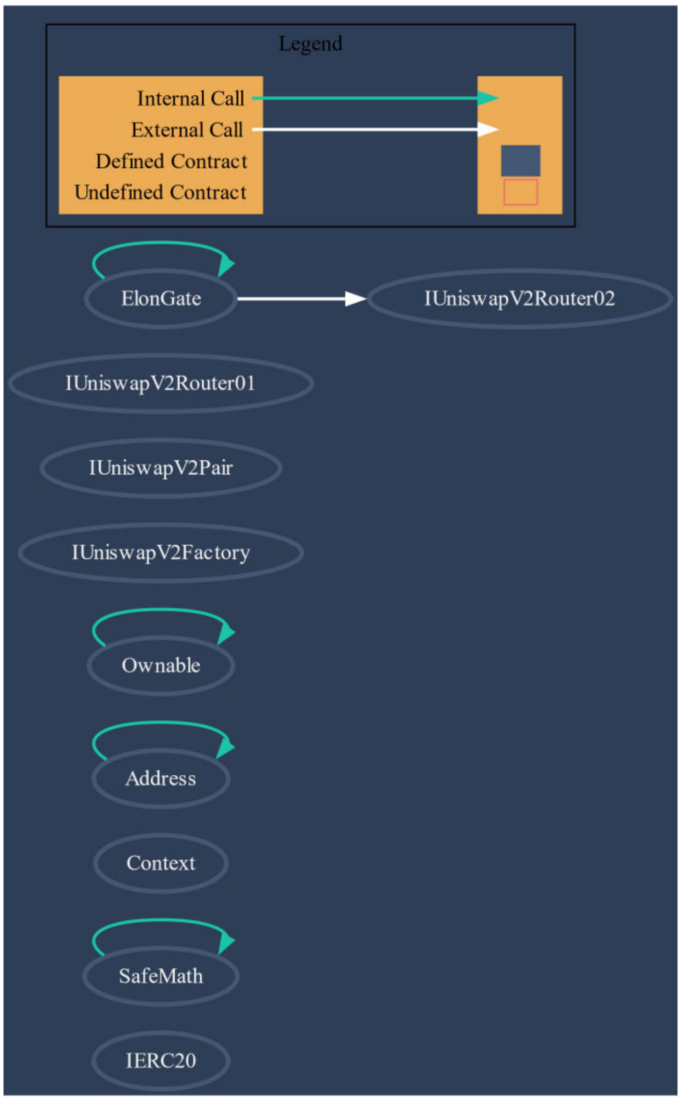
The contracts audited are from the ElonGate-creator/ElonGateToken git repository. The audit is based on the commit 'Create elongate.sol' from 26/03/2021.

The audited contracts are:

elongatetoken.sol

The scope of the audit is limited to these files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

elongate.sol Interaction Graph



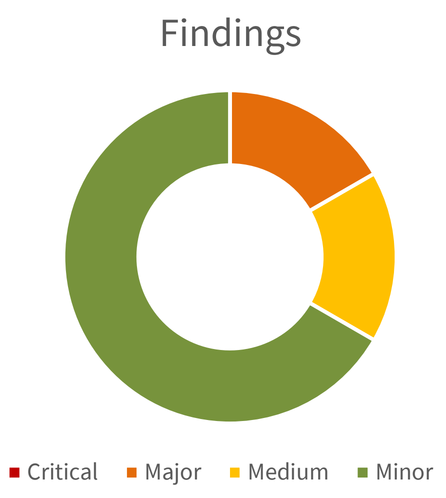
Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found **0** critical issue, **1** Major issues, **1** medium issues, and **3** minor issues.



Security Issues

ID	Title	Severity	Status
01	Centralized Risk in addLiquidity()	Major	**Acknowledged**
02	Contract gains non-withdrawable BNB via the swapAndLiquify() function	Medium	**Acknowledged**
03	Third Party Dependencies	Minor	**Acknowledged**
04	Privileged Ownerships	Minor	**Acknowledged**
05	Unprotected/Recoverable Ownership	Minor	**Acknowledged**

Findings

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them, or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

N/A

Major Severity Issues

Centralized Risk in addLiquidity()

Description: The addLiquidity function calls the uniswapV2Router.addLiquidityETH function with the to address specified as owner() for acquiring the generated LP tokens from the ElonGate-BNB pool. As a result, overtime the _owner address will accumulate a significant portion of LP tokens. If the _owner is an EOA(Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation: We advise the to address of the uniswapV2Router.addLiquidityETH function call to be replaced by the contract's address itself and to provide access to the underlying LP tokens via dedicated business-oriented functions.

Status: **Acknowledged**

Medium Severity Issues

Contract gains non-withdrawable BNB via the swapAndLiquify()

Description: The swapAndLiquify function converts half of the contractTokenBalance ElonGate tokens to BNB. The other half of ElonGate tokens and part of the converted BNB are deposited into the ElonGate-BNB pool on pancakeswap as liquidity. For every swapAndLiquify function call, a small amount of BNB leftover in the contract. This is because the price of ElonGate drops after swapping the first half of ElonGate tokens into BNBs, and the other half of ElonGate tokens require less than the converted BNB to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those BNB, and they will be locked in the contract forever.

Recommendation: It's not ideal that more and more BNB are locked into the contract over time. The simplest solution is toadd a withdraw function in the contract to withdraw BNB.

Status: **Acknowledged**

Minor Severity Issues

Third Party Dependencies

Description: The contract is serving as the underlying entity to interact with third party PancakeSwap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen.

Recommendation: We understand that the business logic of the ElonGate protocol requires the interaction PancakeSwapprotocol for adding liquidity to ElonGate-BNB pool and swap tokens. We encourage the team to constantlymonitor the statuses of those third parties to mitigate the side effects when unexpected activities areobserved.

Status: **Acknowledged**

Privileged Ownerships

Description: The owner of contract ElonGate has the permission to:

1. change the address that can receive LP tokens,
2. lock the contract,
3. exclude/include addresses from rewards/fees,
4. set taxFee , liquidityFee and _maxTxAmount ,
5. enable swapAndLiquifyEnabled

Recommendation: Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governingprocedure and let the community monitor in respect of transparency considerations.

Status: **Acknowledged**

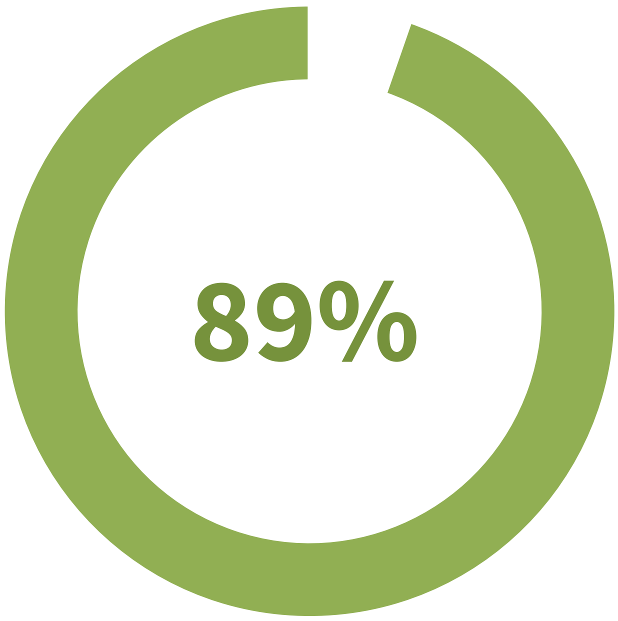
Unprotected/Recoverable Ownership

Description: Function lock() would set the owner to address(0) and save the current owner address to a state variable _previousOwner . Function unlock() can only be triggered by the previous owner and set owner back to address of _previousOwner .However, this lock and unlock feature could do harm to transferOwnership() and renounceOwnership()

Recommendation: Recommend removing lock and unlock functions in the contract, since these two function are not used in the contract. If there are uses cases for the time lock functionality, the contract of Compound TimeLockcan be used as a reference

Status: **Acknowledged**

Security Rating



Based on Vulnerabilities Found