Security Assessment

ELONGATE

May 10th, 2021

Summary

This report has been prepared for ElonGate smart contracts, 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 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.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	ELONGATE
Platform	BSC
Language	Solidity
Codebase	https://github.com/ElonGate-creator/ElonGateToken
Commits	ca392d3c8b4c1aca356e7cab9ccfd1b4dbc629b7

Audit Summary

Delivery Date	May 10, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	ElonGateToken

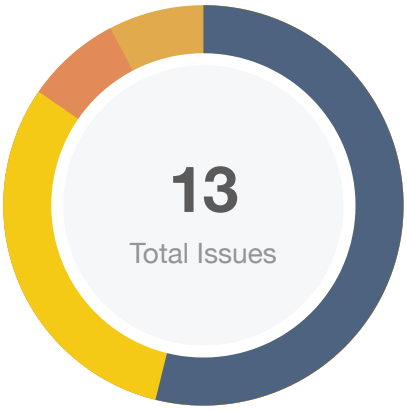
Vulnerability Summary

Total Issues	13
Critical	0
Major	1
Medium	1
Minor	4
Informational	7
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
EGT	elongate.sol	246b339f72e02135baa9d6bd0090b4701b34dd98f0fc01cb819112cab756a5b9

Findings



Critical	**0** (0.00%)
Major	**1** (7.69%)
Medium	**1** (7.69%)
Minor	**4** (30.77%)
Informational	**7** (53.85%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
EGT-01	Incorrect Error Message	Logical Issue	Minor	Acknowledged
EGT-02	Redundant Code	Logical Issue	Informational	Acknowledged
EGT-03	Contract gains non-withdrawable BNB via the `swapAndLiquify()` function	Logical Issue	Medium	Acknowledged
EGT-04	Centralized Risk in `addLiquidity()`	**Centralization / Privilege**	**Major**	**Partially Resolved**
EGT-05	Variable Could Be Declared as `constant`	Gas Optimization	Informational	Acknowledged
EGT-06	Return Value Not Handled	Volatile Code	Informational	Acknowledged
EGT-07	Third Party Dependencies	Control Flow	Minor	Acknowledged
EGT-08	Missing Event Emitting	Coding Style	Informational	Acknowledged
EGT-09	Function and variable naming doesn't match the operating environment	Coding Style	Informational	Acknowledged
EGT-10	Privileged Ownerships	**Centralization / Privilege**	**Minor**	**Partially Resolved**

ID	Title	Category	Severity	Status
EGT-11	Typos in Code and Comments	Coding Style	Informational	Acknowledged
EGT-12	Purpose of Function `deliver()`	Control Flow	Informational	Acknowledged
EGT-13	Unprotected/Recoverable Ownership	**Logical Issue, Centralization / Privilege**	**Minor**	**Acknowledged**

EGT-01 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	Minor	elongate.sol: 854~855	Acknowledged

Description

The error message of the `require` statement, `require(!_isExcluded[account], "Account is already excluded");` does not describe the error correctly.

Recommendation

The message "Account is already excluded" can be changed to "Account is not excluded" .

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-02 | Redundant Code

Category	Severity	Location	Status
Logical Issue	Informational	elongate.sol: 854~855	Acknowledged

Description

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` could be included in `else {...}`.

Recommendation

The following code can be removed:

```
... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient, amount);
} ...
```

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-03 | Contract gains non-withdrawable BNB via the `swapAndLiquify()` function

Category	Severity	Location	Status
Logical Issue	Medium	elongate.sol: 1042~1043	Acknowledged

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 to add a `withdraw` function in the contract to withdraw BNB. Other approaches that benefit the ElonGate token holders can be:

- Distribute BNB to ElonGate token holders proportional to the amount of token they hold.
- Use leftover BNB to buy back ElonGate tokens from the market to increase the price of ElonGate.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

[ElonGate Team]: There is no Withdraw function in the contract to extract these leftover BNBs. Creating a withdraw function in the contract also opens up the possibility of malicious use if not handled properly. The code for the ELONGATE token is Open Source and was not written by the team. Smart Contracts deployed on the blockchain cannot be edited. As it is impossible to update the code directly, it will be left unchanged. The ELONGATE team is confident that this issue has no impact on the security or sustainability of the token.

EGT-04 | Centralized Risk in `addLiquidity()`

Category	Severity	Location	Status
Centralization / Privilege	Major	elongate.sol: 1093	Partially Resolved

Description

```
// add the liquidity
uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    owner(),
    block.timestamp
);
```

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, over time 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, such as simply withdrawing a portion of the LP tokens as an incentive for the `_owner` of the ElonGate project. In any case, we strongly recommend any central addresses of the system (i.e. the `_owner`) to be replaced by smart-contract based addresses with enhanced security features and decentralization in mind.

Indicatively, here are some solutions:

1. Time-lock with reasonable latency for community awareness on privileged operations;
2. Multisig with community-voted 3rd-party independent co-signers;
3. DAO or Governance module increasing transparency and community involvement;

Alleviation

[ElonGate Team]: ELONGATE's business model is fundamentally based on extracting from the unlocked portion of the liquidity pool (LP) to facilitate transparent charitable donations. The majority of the LP tokens are locked in DX Sale for 1-5 years, as an assurance to the community. As with any business with routine operational costs, access to some capital is required. ELONGATE is charity- first, so operations are lean and fully transparent. As an additional risk mitigation, the leadership team is fully de-anonymized. When LP tokens are extracted from the unlocked portion of the liquidity pool, the amount is sent to various business wallets to meet our operational needs. These wallet addresses are displayed on our website and anyone can see the full list of transactions. Any LP tokens not needed are locked away in DX Sale. At the time of writing, 80% of the LP Tokens are locked in DX Sale. Here is a list of transactions showing the Locked LPs. This information is also available on our website. We will continue to lock excessive LP tokens away and share the results with the community.

1. https://dxsale.app/app/pages/dxlockview?id=0&add=0x56662A77D203EA79956fAC4a798836D2B4B9170B&type=lplo ck&chain=BSC
2. https://dxsale.app/app/pages/dxlockview?id=1&add=0x56662A77D203EA79956fAC4a798836D2B4B9170B&type=lplo ck&chain=BSC
3. https://dxsale.app/app/pages/dxlockview?id=2&add=0x56662A77D203EA79956fAC4a798836D2B4B9170B&type=lplo ck&chain=BSC
4. https://dxsale.app/app/pages/dxlockview?id=3&add=0x56662A77D203EA79956fAC4a798836D2B4B9170B&type=lplo ck&chain=BSC
5. https://dxsale.app/app/pages/dxlockview?id=4&add=0x56662A77D203EA79956fAC4a798836D2B4B9170B&type=lplo ck&chain=BSC

EGT-05 | Variable Could Be Declared as `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	elongate.sol: 707~709(ElonGate), 703~704(ElonGate), 724~725(ElonGate)	Acknowledged

Description

Variables `_tTotal`, `numTokensSellToAddToLiquidity`, `_name`, `_symbol` and `_decimals` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We recommend declaring those variables as `constant`.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-06 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	Informational	elongate.sol: 1088~1089	Acknowledged

Description

The return values of function `addLiquidityETH` are not properly handled.

```
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-07 | Third Party Dependencies

Category	Severity	Location	Status
Control Flow	Minor	elongate.sol	Acknowledged

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 PancakeSwap protocol for adding liquidity to ElonGate-BNB pool and swap tokens. We encourage the team to constantly monitor the statuses of those third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[ElonGate Team]: ELONGATE conducts routine risk assessments. If any unauthorized 3rd party protocol exploits or destabilizes ELONGATE, the team will alert the community and react swiftly to fix the issue.

EGT-08 | Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	elongate.sol: 817~818, 843~844, 853~854, 865~866, 876~877, 880~881, 884~885, 888~889, 892~893, 898~899, 906~907, 949~950, 969~970, 979~980, 1119~1120, 1128~1129, 1138~1139	Acknowledged

Description

In contract `ElonGate`, there are a bunch of functions can change state variables. However, these function do not emit event to pass the changes out of chain.

Recommendation

Recommend emitting events, for all the essential state variables that are possible to be changed during runtime.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-09 | Function and variable naming doesn't match the operating environment

Category	Severity	Location	Status
Coding Style	Informational	elongate.sol: 1~7	Acknowledged

Description

Based on the comments at the very beginning of the contract, we assumed that the contract are to be deployed on Binance Smart Chain.

The ElonGate contract uses Pancakeswap for swapping and add liquidity to Pancakeswap pool, but naming it Uniswap. Function `swapTokensForEth(uint256 tokenAmount)` swaps ElonGate token for BNB instead of ETH.

Recommendation

Change "Uniswap" and "ETH" to "Pancakeswap" and "BNB" in the contract respectively to match the operating environment and avoid confusion.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-10 | Privileged Ownerships

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	elongate.sol	**Partially Resolved**

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`

without obtaining the consensus of the community.

Recommendation

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

Alleviation

[ElonGate Team]: Given the trust of the community, the privileged ownership aspect of the smart contract allows ELONGATE to be flexible with certain aspects of the implementation in the long-term. Over the course of many years, the community may determine that a tax fee change or LP redirection is required, so the leadership team will have the ability to execute on that. Therefore, there will be no transfer of ownership but as an extra layer of security, the Leadership team will implement MultiSig and this will be completed as soon as possible. This means that the private keys will be split between the Leadership team. It will require 2/3 private keys to authorise an action on the contract or to do a transaction.

EGT-11 | Typos in Code and Comments

Category	Severity	Location	Status
Coding Style	Informational	elongate.sol: 903~904, 731~732	Acknowledged

Description

There are several typos in the code and comments.

1. `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

```
event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );
```

2. `recieve` should be `receive` and `swaping` should be `swapping` in the line of comment `//to recieve ETH from uniswapV2Router when swaping`.

Recommendation

Recommend correcting all typos in the contract.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-12 | Purpose of Function `deliver()`

Category	Severity	Location	Status
Control Flow	Informational	elongate.sol: 817~818	Acknowledged

Description

The function `deliver()` can be called by anyone. It accepts an `uint256` number parameter `tAmount`. The function reduces the ElonGate token balance of the caller by `rAmount`, where `tAmount` reduces the transaction fee. Then the function adds `tAmount` to variable `_tFeeTotal`, which represents the contract's total transaction fee. We would like to learn more about the purpose and use cases of this functionality.

Alleviation

The team acknowledged the findings and given that the deployed contract cannot be edited, decided to retain the code base unchanged.

EGT-13 | Unprotected/Recoverable Ownership

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	**Minor**	elongate.sol: 459~473(Owner)	**Acknowledged**

Description

Function `lock()` would set the owner to `address(0)` and save teh 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 would do harm to `transferOwnership()` and `renounceOwnership()`. For example, if the current owner, let's say `0xA`, do the following operations:

1. Lock the contract by calling `lock()`. Then `_previousOwner` has value `0xA` saved.
2. Unlock the contract by calling `unlock()`, such that `transferOwnership()` and `renounceOwnership()` can be called.
3. Call `transferOwnership()` or `renounceOwnership()` to assigned `owner` to be a new address or zero address.
4. The previous owner, `0xA`, can call function `unlock()` again to regain the ownership.

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 timelock functionality, the contract of Compound TimeLock can be used as a reference.

Alleviation

[ElonGate Team]: As per the reasons identified in EGT-10, ownership will not be transferred or renounced. Therefore this function will never be called as it directly affects our Business Model.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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 in-storage 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 but rather 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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content string of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.