Security Assessment

RIGHTDOG

March 16th, 2022

Table of Contents

Summary

This report has been prepared for RIGHT to discover issues and vulnerabilities in the source code of the RightDog project 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 critical 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;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	RIGHT
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0x08d4e1d410316f807440cfd35805c8eaff3b53eb#code
Commit	

Audit Summary

Delivery Date	Aug 04, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	1	0	0
Medium	3	0	0	3	0	0
Minor	3	0	0	3	0	0
Informational	7	0	0	7	0	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
RD	Right.sol	d0ef6e4af501d3cba32cb68b1fd7fa59793432c30ba1d4771b2fcc7971c644f1

Review Notes

Overview

The RIGHT Protocol is a decentralized finance (DeFi) token deployed on the Binance smart chain (BSC). RIGHT employs two novel features in its protocol: static rewards for each user as well as an LP acquisition mechanism. The static reward (also known as reflection) and LP acquisition mechanisms function as follows:

Each RIGHT transaction is taxed two 2% fees totaling 4% of the transaction amount. The first fee is redistributed to all existing holders using a form of rebasing mechanism whilst the other 2% is accumulated internally until a sufficient amount of capital has been amassed to perform an LP acquisition. When this number is reached, the total tokens accumulated are split with half being converted to BNB and the total being supplied to the PancakeSwap contract as liquidity.

LP Acquisition

The LP acquisition mechanism can be indirectly triggered by any normal transaction of the token as all transfers evaluate the set of conditions that trigger the mechanism. The main conditions of the mechanism are whether the sender is different than the LP pair and whether the accumulation threshold has been reached. Should these conditions be satisfied, the `swapAndLiquify` function is invoked with the current contract's RIGHT balance.

The `swapAndLiquify` function splits the contract's balance into two halves properly accounting for any truncation that may occur. The first half is swapped to BNB via the PancakeSwap Router using the RIGHT-BNB pair and thus temporarily driving the price of the RIGHT token down. Afterwards, the resulting BNB balance along with the remaining RIGHT balance is supplied to the RIGHT-BNB liquidity pool as liquidity via the Router. The recipient of the LP units is defined as the current `owner` of the RIGHT contract, a characteristic outlined in more depth within finding 01.

Static Reward (Reflection)

Balances in the RIGHT token system are calculated in one of two ways. The first method, which most users should be familiar with, is a traditional fixed number of units being associated with a user's address. The second method, which is of interest to static rewards, represents a user's balance as a proportion of the total supply of the token. This method works similarly to how dynamic rebasing mechanisms work such as that of Ampleforth.

Whenever a taxed transaction occurs, the 2% meant to be re-distributed to token holders is deducted from the total "proportion" supply resulting in a user's percentage of total supply being increased. Within the system, not all users are integrated into this system and as such the 2% fee is rewarded to a subset of the

total users of the RIGHT token. The owner of the contract is able to include and exclude users from the dynamic balance system at will.

Privileged Functions

The contract contains the following privileged functions that are restricted by the `onlyOwner` modifier. They are used to modify the contract configurations and address attributes. We grouped these functions below:

Account management functions for inclusion and exclusion in the fee and reward system:

- function excludeFromReward(address account)
- function includeInReward(address account)
- function excludeFromFee(address account)
- function includeInFee(address account)

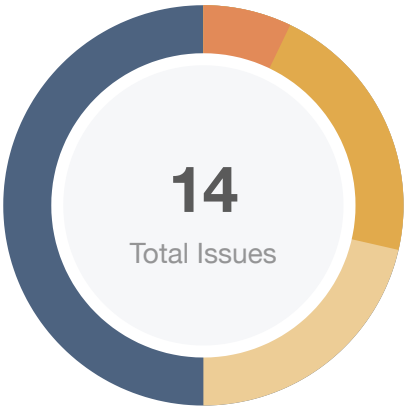
Modification of liquidation, tax and max transaction percents, and trading status of the system:

- function setTaxFeePercent(uint256 taxFee)
- function setLiquidityFeePercent(uint256 liquidityFee)
- function setMaxTxPercent(uint256 maxTxPercent)
- function enableTrading()

Toggle feature of the LP acquisition mechanism:

- function setSwapAndLiquifyEnabled(bool _enabled)

Findings



Critical	0 (0.00%)
Major	1 (7.14%)
Medium	3 (21.43%)
Minor	3 (21.43%)
Informational	7 (50.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
01	Centralized Risk in `addLiquidity`	**Centralization / Privilege**	**Major**	Acknowledged
02	Contract Gains Non-withdrawable BNB via the `swapAndLiquify` Function	Logical Issue	Medium	Acknowledged
03	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
04	Possible to Gain Ownership after Renouncing the Contract Ownership	**Logical Issue, Centralization / Privilege**	**Medium**	Acknowledged
05	Incorrect Error Message	Logical Issue	Minor	Acknowledged
06	Third Party Dependencies	Control Flow	Minor	Acknowledged
07	Potential Sandwich Attack	Volatile Code	Minor	Acknowledged
08	Typos in the Contract	Coding Style	Informational	Acknowledged
09	Return Value Not Handled	Volatile Code	Informational	Acknowledged
10	Redundant Code	Logical Issue	Informational	Acknowledged
11	Missing Event Emitting	Coding Style	Informational	Acknowledged
12	Unused Event	Coding Style	Informational	Acknowledged
13	Variable Could Be Declared As `constant`	Gas Optimization	Informational	Acknowledged

ID	Title	Category	Severity	Status
14	Function and Variable Naming Doesn't Match the Operating Environment	Coding Style	● Informational	ⓘ Acknowledged

01 | Centralized Risk in `ddLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	RIGHT.sol: 1048	ⓘ Acknowledged

Description

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function listed below with the `to` address specified as owner() for acquiring the generated LP tokens from the RIGHT-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.

```
1042  // add the liquidity
1043  uniswapV2Router.addLiquidityETH{value: ethAmount}(
1044      address(this),
1045      tokenAmount,
1046      0, // slippage is unavoidable
1047      0, // slippage is unavoidable
1048      owner(),
1049      block.timestamp
1050  );
```

Recommendation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[RIGHT Team]: The deployer regularly burns those LP tokens (send to zero address). So the risk is mitigated.

In fact, we have been taking advantage of this to actually move liquidity slowly from v1 to v2. As liquidity in v1 is locked forever (LP tokens burnt) we can't withdraw and move it to PancakeSwap v2. However, as new LP tokens are generated for the deployer, we remove that new liquidity and add it to v2.

The owner of the contract has been set to 0x9eAf318B9aAcDB6b8ae71d7465c2523DA1f1eBD6.
Relevant transactions can be checked on bscscan.

02 | Contract Gains Non-withdrawable BNB via the `wapAndLiquify` Function

Category	Severity	Location	Status
Logical Issue	Medium	Right.sol: 997	Acknowledged

Description

The `swapAndLiquify` function converts half of the contract `TokenBalance` RIGHT tokens to BNB. The other half of RIGHT tokens and part of the converted BNB are deposited into the RIGHT-BNB pool on PancakeSwap as liquidity. For every `swapAndLiquify` function call, a small amount of BNB is leftover in the contract. This is because the price of RIGHT drops after swapping the first half of RIGHT tokens into BNBs, and the other half of RIGHT 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 RIGHT token holders can be:

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

Alleviation

[RIGHT Team]: There is nothing we can do about it.

03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Medium	RIGHT.sol: 791, 800, 823, 827, 831, 835, 839, 845, 850	Acknowledged

Description

With the modifier `onlyOwner`, the 'owner' has the authority to call the following sensitive functions to change settings of the RIGHT contract:

- `RIGHT.excludeFromReward(address account)`
- `RIGHT.includeInReward(address account)`
- `RIGHT.excludeFromFee(address account)`
- `RIGHT.includeInFee(address account)`
- `RIGHT.setTaxFeePercent(uint256 taxFee)`
- `RIGHT.setLiquidityFeePercent(uint256 liquidityFee)`
- `RIGHT.setMaxTxPercent(uint256 maxTxPercent)`
- `RIGHT.setSwapAndLiquifyEnabled(bool _enabled)`
- `RIGHT.enableTrading()`

Any compromise to the `owner` account may allow the hacker to adversarially manipulate the settings of the contract.

Recommendation

We advise the client to carefully manage the 'owner' account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Timelock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

04 | Possible to Gain Ownership after Renouncing the Contract Ownership

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	**Medium**	RIGHT.sol: 418	Acknowledged

Description

An owner is possible to gain ownership of the contract even if he/she calls the function `renounceOwnership` to renounce the ownership. This can be achieved by performing the following operations:

1. Call `lock` to lock the contract. The variable `_previousOwner` is set to the current owner.
2. Call `unlock` to unlock the contract.
3. Call `renounceOwnership` to leave the contract without an owner.
4. Call `unlock` to regain ownership.

Recommendation

We advise updating/removing `lock` and `unlock` functions in the contract, or removing the `renounceOwnership` if such a privilege retains at the protocol level. If timelock functionality could be introduced, we recommend using the implementation of Compound finance as a reference.

Reference: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

Alleviation

N/A

05 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	Minor	RIGHT.sol: 801	Acknowledged

Description

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

Recommendation

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

Alleviation

N/A

06 | Third Party Dependencies

Category	Severity	Location	Status
Control Flow	Minor	RIGHT.sol: 691	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party PancakeSwap protocols. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third-party entities can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third-party entities can possibly create severe impacts, such as increasing fees of third-party entities, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of the RIGHT protocol requires the interaction PancakeSwap protocol for adding liquidity to the RIGHT-BNB pool and swapping tokens. We encourage the team to constantly monitor the statuses of those third-party entities to mitigate the side effects when unexpected activities are observed.

Alleviation

N/A

07 | Potential Sandwich Attack

Category	Severity	Location	Status
Volatile Code	Minor	RIGHT.sol: 1029~1035, 1043~1050	Acknowledged

Description

Potential sandwich attacks could happen if calling `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens` and `uniswapV2Router.addLiquidityETH` without setting restrictions on slippage.

For example, when we want to make a transaction of swapping 100 AToken for 1 ETH, an attacker could raise the price of ETH by adding AToken into the pool before the transaction so we might only get 0.1 ETH. After the transaction, the attacker would be able to withdraw more than he deposited because the total value of the pool increases by 0.9 ETH.

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Alleviation

N/A

08 | Typos in the Contract

Category	Severity	Location	Status
Coding Style	Informational	RIGHT.sol: 413, 679	Acknowledged

Description

The function name `geUnlockTime()` should be `getUnlockTime()`, and the variable name `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

Recommendation

We recommend correcting these typos in the contract.

Alleviation

N/A

09 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	Informational	RIGHT.sol: 1043~1050	Acknowledged

Description

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

```
1043        uniswapV2Router.addLiquidityETH{value: ethAmount}(
1044            address(this),
1045            tokenAmount,
1046            0, // slippage is unavoidable
1047            0, // slippage is unavoidable
1048            owner(),
1049            block.timestamp
1050        );
```

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

N/A

10 | Redundant Code

Category	Severity	Location	Status
Logical Issue	Informational	RIGHT.sol: 1063	Acknowledged

Description

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` can be included in `else` .

Recommendation

The following code can be removed:

```
1062  ... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
1063      _transferStandard(sender, recipient, amount);
1064  } ...
```

Alleviation

N/A

11 | Missing Event Emitting

Category	Severity	Location	Status
Coding Style	● Informational	RIGHT.sol: 831, 835, 839, 850, 823, 827	ⓘ Acknowledged

Description

In contract `RIGHT`, there are a bunch of functions that can change state variables. However, these functions do not emit events to pass the changes out of the chain. Some example functions are listed below:

- `RIGHT.excludeFromFee(address)`
- `RIGHT.includeInFee(address)`
- `RIGHT.setTaxFeePercent(uint256)`
- `RIGHT.setLiquidityFeePercent(uint256)`
- `RIGHT.setMaxTxPercent(uint256)`
- `RIGHT.enableTrading()`

Recommendation

We recommend declaring and emitting corresponding events for all the essential state variables that are possible to be changed during runtime.

Alleviation

N/A

12 | Unused Event

Category	Severity	Location	Status
Coding Style	Informational	RIGHT.sol: 674	Acknowledged

Description

`MinTokensBeforeSwapUpdated` event is declared but never used.

Recommendation

We recommend removing `MinTokensBeforeSwapUpdated` event.

Alleviation

N/A

13 | Variable Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	RIGHT.sol: 650, 654~656, 672	Acknowledged

Description

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

Recommendation

We recommend declaring those variables as `constant`.

Alleviation

N/A

14 | Function and Variable Naming Doesn't Match the Operating Environment

Category	Severity	Location	Status
Coding Style	Informational	RIGHT.sol	Acknowledged

Description

The RIGHT contract uses PancakeSwap for swapping and adding liquidity to the PancakeSwap pool, but naming it Uniswap. Function `RIGHT.swapTokensForEth(uint256)` swaps RIGHT token for BNB instead of ETH.

Recommendation

We recommend changing "Uniswap" and "ETH" to "PancakeSwap" and "BNB" in the contract respectively to match the operating environment and avoid confusion.

Alleviation

N/A

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.

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.

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.

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 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.

About

We are a group of freelancer blockchain developers and web3 services providers,
For smart contracts audit we check and verify the security, robustness, optimization 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.