Astronaut: Smart Contract Audit

HashEx was commissioned by the Astronaut team to perform an audit of Astronaut smart contracts. The audit was conducted between March 23 and March 26, 2021.

The audited code is located in Astronaut's github repository. The audit was performed for astronauttoken commit 51da64f. There was limited documentation provided on astronaut-1.gitbook.io.

The purpose of this audit was to achieve the following:

● Identify potential security issues with smart contracts.

● Formally check the logic behind given smart contracts.
Information in this report should be used to understand the risk exposure of smart

contracts, and as a guide to improve the security posture of smart contracts by remediating the issues that were identified.
We found out that Astronaut token is a fork of Reflect.finance [1] custom token with an audit report available [2].

Contracts overview

`astronaut-contract/AstronautToken.sol`

Implementation of BEP20 token standard with custom functionality of auto-yield by burning tokens on transfers.

Found issues

#01 No safeguard for `_TAX_FEE` : Critical

Astronaut.sol L509 and L513 contains public `onlyOwner` functions that set `_TAX_FEE` to any value of `uint256`. This behavior is dangerous and must be mitigated by renouncing contract ownership as soon as possible. There are no getter functions nor events to check the current or previous values of `_TAX_FEE` variable.

#02 `excludeAccount()` abuse: High

Exclusion and inclusion of certain addresses into auto-yield mechanism may cause a confusing result for users. Including back the greater part of owner's balance would revert the rate (and users' balances) to the almost initial state of the time it was excluded. Moreover, the possible sale from excluded owner address will redistribute balances in profit of the owner in case of backwards including. We suggest removing `includeAccount()` function or lock exclusion/inclusion methods by renouncing ownership.

#03 BEP20 standard violation (restriction of zero amount transfer): High

Implementation of transfer() function does not allow to input zero amount as it's demanded in ERC-20 [3] and BEP-20 [4] standards. This issue may break the interaction with smart contracts that rely on full ERC20 support.

#04 `for()` loop in `getCurrentSupply()` : High

Mechanism of removing addresses from auto-yielding implies loop over excluded addresses for every transfer operation or balance inquiry. This may lead to extreme gas costs up to block gas limit and may be avoided only by the owner restricting the number

of excluded addresses. In an extreme situation with a large number of excluded addresses transaction gas may exceed maximum block gas size and all transfers will be effectively blocked.

#05 `updateBurnFee()` bug: Medium

`updateBurnFee()` function sets new value to `_TAX_FEE` instead of `_BURN_FEE`. Apparently the contract was designed with the intention of _BURN_FEE to be updated, but it is impossible.

#06 Low severity issues and recommendations: Low

1. excludeAccount() function contains hardcoded address of Uniswap Router02 [5] in Ethereum blockchain which make this requirement completely useless for current realization in BSC. Moreover, address hardcoding is not a good practice as third-party contracts could be deprecated/upgraded.
2. Maximum transfer amount is restricted for a non-owner user by `_MAX_TX_SIZE` variable. This variable is set to 10x of total supply which makes it virtually useless still gas consuming.
3. Private function `_getTaxFee()` is not used anywhere. It's visibility should be changed to external.
4. Function `transfer()` consists of an excessive number of conditional statements. Default condition is never reached.
5. `updateTaxFee()` and `updateBurnFee()` functions use unsafe multiplication, although the proposed safeguard will justify this.
6. Incorrect error message: must be "Account is already included".
7. Solidity version is not fixed but set to pragma ^0.6.0. Address library is OpenZeppelin
   v3.0.0 with ^0.6.2. Such inconsistencies may lead to compilation errors.
8. There are zero custom events besides IBEP20 interface. We suggest to implement such events for user actions as well as for changing values of crucial variables.
9. Defaults for `_TAX_FEE` and `_BURN_FEE` are 6% and 3%. Whitepaper claims "we will set it to 4% slippage which will do a 3% burn and 1% redistribution". Up to the block 6008624 we can't ensure that tax fee was updated.

#07 General recommendations: Low

1. Getters functions for contract constants may be set to pure and/or external.

   `name()`

   `symbol()`

   `decimals()`

   `_getMaxTxAmount()`

2. We suggest to add a `permit()` function to ERC20 tokens for gas savings on approvals.

3. Private function `_getMaxTxAmount()` is not used anywhere. It could be removed for the gas economy or made external.

4. `_GRANULARITY` constant can be used with 100x multiplier to save gas on calculations.

5. `_tFeeTotal` and `_tBurnTotal` variables are not in practical usage. The contract may be reworked to reduce gas costs.

6. Address library is not in use and may be removed.

7. We suggest adding unit tests for future development according to the roadmap of the Astronaut project.

8. Code style recommendations: `_TAX_FEE` and `_BURN_FEE` should be in mixedCase; omitted curly braces should be added for readability; four different cases of transfer function (L646, L656, L667, L678) should be merged in `_transfer()`.

Conclusion

Reviewed contract is deployed at 0x05B339B0A346bF01f851ddE47a5d485c34FE220c in BSC. The audited contract is a fork of Reflect.finance smart contract with some changes such as ability to change fees.

One critical and 3 high severity issues were found. Most of the issues can be mitigated by renouncing ownership of the contract.

Audit includes recommendations on the code improving and preventing potential attacks.