#HashEx
BLOCKCHAIN SECURITY

Doggo

smart contracts final audit report

September 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Doggo team to perform an audit of their smart contract. The audit was conducted between 04/09/2023 and 06/09/2023.

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 for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at 0x890952cFe56dc0D68119e007F3F1D1AfB746E8c1 in the BNB Smart Chain.

Update. A recheck was done for the code deployed to the BNB Smart Chain at 0x67B81015c0c608A50A4d3d4148a2FA59a8075f40.

Update 2. A second recheck was done for the code deployed to the BNB Smart Chain at 0x369ecb308640C6839a675d161ea4D557200494C2.

Update 3. A third recheck was done for the code deployed to the BNB Smart Chain at 0xB93E3077eeCeA50F9814a8DBD5E1cD0A25d9013F.

2.1 Summary

Project name	Doggo
URL	https://doggo.games/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
DOGGO	0x369ecb308640C6839a675d161ea4D557200494C2

3. Found issues

High	2 (33%)
Medium	1 (17%)
Low	2 (33%)
Info	1 (17%)

Cf7. DOGGO

ID	Severity	Title	Status
Cf7I46	High	Errors in transfer tax	Resolved
Cf7I45	High	Owner's exaggerated rights	Partially fixed
Cf7Idf	Medium	Incorrect tax calculation in getFeeAmount function for whitelisted recipients	Open
Cf7I43	Low	Gas optimizations	Resolved
Cf7I62	Low	Lack of input validation	Resolved
Cf7I44	Info	Lack of events	Resolved

4. Contracts

Cf7. DOGGO

Overview

An ERC-20 standard token with owner governed capped mint and transfer tax.

Issues

Cf7I46 Errors in transfer tax

High Resolved

The `_tax_burn` flag is used to burn transfer fee instead of collecting in the `_taxRecipient` address. If `_tax_burn` is set to true, the sender's balance is reduced by `amount+tax` and recipient's balance is increased by `amount-tax`, so one tax amount is burnt correctly with event emitted, and the second tax amount is burnt silently.

The second problem with `_tax_burn` is that if it's enabled, the burn `amount` is burnt not from sender address, but from `msg.sender`. This breaks interaction with other contracts calling `transferFrom()` function. In some situations, `transferFrom()` calls may be reverted if the caller doesn't have tokens to be burnt.

The third problem is lack of `Transfer()` event in the `else` section of tax transfer to the `_taxRecipient`. Lack of transfer event usually results in explorer errors and not accurate balance tracking.

```
  function _transfer(address sender, address recipient, uint256 amount) internal {
    ...
    _balances[sender] = _balances[sender].sub(amount, "BEP20: transfer amount exceeds
balance");
    uint256 feeAmount = amount.mul(__tax).div(100000);
    if(_tax_burn) {
      _burn(_msgSender(), feeAmount);
    } else {
```

```
      _balances[__taxRecipient] = _balances[__taxRecipient].add(feeAmount);
      emit TaxTransfer(sender, __taxRecipient, feeAmount);
    }
    uint256 _amount = amount.sub(feeAmount);
    _balances[recipient] = _balances[recipient].add(_amount);
    ...
  }
```

Recommendation

Disable `_tax_burn` flag.

Replace `_burn(_msgSender(), feeAmount)` with `emit Transfer(sender, address(0), feeAmount)` in the `if(_tax_burn)` section in code updates.

Add `Transfer()` event for the tax transfer to the tax recipient.

Update

The first and second points were fixed with the update, effectively reducing severity of the issue. The last issue regarding missing `Transfer()` event remains.

This issue was marked as high severity one, and has been mitigated to low severity with code update.

Cf7145 Owner's exaggerated rights

High Partially fixed

There's no safety limit for tax fee percent. Setting it over 100% will result in transfer fail due to math underflow.

```
function setTax(uint256 tax) external{
  require(isPartner(msg.sender), 'setAddress: require isPartner(msg.sender)');
  _tax = tax;
}
```

Token can be minted by 3 privileged accounts (owner, admin, partner). Two of them have no public getters. As of 05/09/2023 all privileged accounts are EOAs and total supply is about

50% of mintable cap.

```
  function mint(address _to, uint256 _amount) external onlyPartner {
    _mint(_to, _amount);
  }

   modifier onlyPartner() {
       require(_owner == msg.sender || _admin == msg.sender || _partner == msg.sender,
'Ownable: caller is not the owner');
       _;
   }
```

In the updated code the tax whitelist was introduced. The `_transfer()` function checks both sender and recipient for membership in this list. Due to inefficiency of the `existsFreeTaxWhiteList()` function, the block gas limit may be exceeded if there's too much whitelisted addresses in tax whitelist. Any of 3 privileged accounts can permanently break all transfers as the `removeFreeTaxWhiteList()` requires much more gas than the `addFreeTaxWhiteList()`.

Recommendation

Secure privileged accounts using Timelock and Multisig contracts.

Add safety checks in case of future updates.

Update

Tax percent has been limited to 2% at most.

Block gas limit issue has been emerged.

Update 2

The introduced in the update possible block gas limit issue was fixed in the second update.

Cf7Idf Incorrect tax calculation in getFeeAmount function for whitelisted recipients

Medium Open

There is a discrepancy between the tax logic implemented in the `_transfer()` function and the `getFeeAmount()` function. The `_transfer()` function considers tax exemptions for both senders and receivers. However, the `getFeeAmount()` function only checks the sender's status in the whitelist, neglecting the receiver's potential exemption. This oversight may lead to inaccurate fee calculations, where an incorrect non-zero tax is returned even if the receiver is exempt.

```
function _transfer(address sender, address recipient, uint256 amount) internal {
  ...

  if(_freeTaxWhiteList[sender] || _freeTaxWhiteList[recipient]){
      __tax = 0;
  }
  ...
}
```

```
function getFeeAmount(address sender, uint256 amount) external view returns(uint256){

    uint256 __tax = _tax;
    if(_freeTaxWhiteList[sender]){
      __tax = 0;
    }

    uint256 feeAmount = 0;
    if(__tax > 0 && amount > 0){

      feeAmount = (amount * __tax) / 100000;

    }

    return feeAmount;
}
```

Recommendation

Check if the receiver is included in _freeTaxWhiteList and return 0.

Cf7I43 Gas optimizations

Low Resolved

1. The `MaxSupply` variable should be declared immutable.

2. The `SafeMath` library is outdated for chosen compiler version as Solidity after 0.8 includes over- and underflowing checks.

3. In the updated code the tax whitelist was introduced. The `removeFreeTaxWhiteList()` and `existsFreeTaxWhiteList()` functions are extremely inefficient. The `_freeTaxWhiteList[]` variable should be a mapping instead of array.

Cf7I62 Lack of input validation

Low Resolved

The function `setTaxRecipient()` doesn't check if the new tax recipient is a zero address.

```
function setTaxRecipient(address taxRecipient) external{
  require(isPartner(msg.sender), 'setAddress: require isPartner(msg.sender)');
  _taxRecipient = taxRecipient;
}
```

This is problematic because transferring funds to a zero address is equivalent to burning them, making them unrecoverable. This could lead to unexpected behaviors and financial loss.

Cf7I44 Lack of events

Info Resolved

The functions `setTaxBurn()`, `setTax()`, `setTaxRecipient()` don't emit events, which complicates the tracking of important off-chain changes.

5. Conclusion

2 high, 1 medium, 2 low severity issues were found during the audit. 1 high, 2 low issues were resolved in the update.

The reviewed contracts are dependent on the list of privileged accounts.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY