HashEx
Blockchain Security

Floshin token

smart contracts
final audit report

December 2021


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.

2. Overview

HashEx was commissioned by the Floshin team to perform an audit of their smart contract. The audit was conducted between 2021-12-01 and 2021-12-03.

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 0x9A6Fed601a11290500F8D76153C33CC254E9F6D0.

2.1 Summary

Project name	Floshin token
URL	https://www.floshin.com
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
AntiBotBabyToken	`0x9A6Fed601a11290500F8D76153C33CC254E9F6D0`

BabyTokenDividendTracker	0x5279EdA97E2ADacA2c288b9CcEfB9a779d0281dA

3. Found issues

High	2 (13%)
Medium	6 (40%)
Low	4 (27%)
Informational	3 (20%)

AntiBotBabyToken

ID	Title	Severity	Status
01	Fees values are not limited	High	Resolved
02	Router update problem	High	Resolved
03	Swaps with 100% slippage	Medium	Acknowledged
04	External calls in transfers	Medium	Acknowledged
05	setSwapTokensAtAmount() has no internal checks on input value	Medium	Acknowledged
06	excludeFromDividends() is permanent	Medium	Acknowledged
07	Updating balances with try method	Medium	Acknowledged
08	addLiquidity() recipient	Medium	Acknowledged

09	updateDividendTracker() not excluding dead address	Low	Acknowledged
10	Gas optimizations	Low	Acknowledged
11	Inconsistent comment	Informational	Acknowledged

BabyTokenDividendTracker

ID	Title	Severity	Status
01	Gas optimizations	Low	Acknowledged
02	Lack of error messages	Low	Acknowledged
03	Denied transfers	Informational	Acknowledged
04	Event emits regardless success of transfer	Informational	Acknowledged

4. Contracts

4.1 AntiBotBabyToken

4.1.1 Overview

Implementation of ERC-20 token standard with fees on transfers.

4.1.2 Issues

01. Fees values are not limited

- High ✓ Resolved

setTokenRewardsFee(), setLiquiditFee() and setMarketingFee() functions update fees parameters without checking new values, so total fees may exceed 100%.

Recommendation

We recommend transferring token ownership to a proxy contract with limited fees in set functions.

02. Router update problem

- High ✓ Resolved

Setting a wrong or malicious router address could break the dividends distribution.

updateUniswapV2Router() function of AntiBotBABYTOKEN contract updates the uniswapV2Router variable and attempts creating a new pair with WETH() of that new

router. It's highly likely that the pair would've already been created at the moment of calling updateUniswapV2Router() and the transaction would be reverted.

Also, the new pair is not included in automatedMarketMakerPairs.

Recommendation

We recommend securing the token ownership with a proxy contract, e.g. Timelock with multisig admin.

03. Swaps with 100% slippage

- Medium ⓘ Acknowledged

swapTokensForEth() and swapTokensForCake() functions call router with 100% slippage and no deadline set. The transactions sent from this contract may be front-runned resulting in swaps with an undesired rate (sandwich attacks).

04. External calls in transfers

- Medium ⓘ Acknowledged

Every transfer makes an external call to the pinkAntiBot address if enableAntiBot is set to true. This contract is behind the proxy and out of scope. If this anti-bot contract malfunctioned, the AntiBotBabyToken token would have fully or partially blocked transfers.

Recommendation

External calls in vital functions should be wrapped in try/catch with emitting caught errors as events. We recommend disabling these external calls when the project enters the stable phase.

05. setSwapTokensAtAmount() has no internal checks on input value

- Medium ⓘ Acknowledged

The owner can update the swapTokensAtAmount variable with a wrong value. This may halt the distribution of the fees for a long period of time. Enabling back swaps and liquidity adding may lead to the token price wrecking if the contract's balance is comparable to a pair reserves.

06. excludeFromDividends() is permanent

- Medium ⓘ Acknowledged

excludeFromDividends() function of AntiBotBABYTOKEN contract calls the same name function of the BABYTOKENDividendTracker contract. Thus the arbitrary address could be restricted from taking the dividends as there’s no inclusion of a mistakenly excluded account.

07. Updating balances with try method

- Medium
- ⓘ Acknowledged

The _transfer() function of AntiBotBABYTOKEN calls for dividendTracker.setBalance() via try method which makes a successful transfer with unchanged balances of dividends tokens possible. The current dividendTracker implementation should not fail on setting balances. It must be noted that dividendTracker can be updated and in case the function setBalance() fails, discrepancies in token balances can take place.

Also, dividendTracker calls should catch returned errors and emit them in corresponding events.

08. addLiquidity() recipient

- Medium
- ⓘ Acknowledged

The addLiquidity() function calls for uniswapV2Router.addLiquidityETH() with the parameter of lp tokens recipient set to zero address. This locks the liquidity forever preventing any future liquidity migrations.

09. updateDividendTracker() not excluding dead address

- Low
- ⓘ Acknowledged

Function updateDividendTracker() does not exclude the “dead” address (0x000000000000000000000000000000000000dEaD) from dividends. If the “dead” address is not excluded it can lead to burning dividends.

10. Gas optimizations

- Low ⓘ Acknowledged

The _transfer() function performs 3 swaps instead of 2.

updateGasForProcessing(), isExcludedFromFees(), withdrawableDividendOf(), dividendTokenBalanceOf(), updateDividendTracker(), updateUniswapV2Router(), excludeMultipleAccountsFromFees(), and setAutomatedMarketMakerPair() functions could be declared external.

Excessive reads from storage in initialize() function.

11. Inconsistent comment

- Informational ⓘ Acknowledged

According to the comment in L103, the swapTokensAtAmount variable default value is 0.002% of total supply, but in reality, it's 0.0002%.

4.2 BabyTokenDividendTracker

4.2.1 Overview

Dividend tracker contract deployed from the token's constructor. Owned by 0x9A6Fed601a11290500F8D76153C33CC254E9F6D0 (AntiBotBabyToken contract).

4.2.2 Issues

01. Gas optimizations

- Low ⓘ Acknowledged

Redundant checking in L455: the statement is always true.

```
            if (gasLeft > newGasLeft) {
                gasUsed = gasUsed.add(gasLeft.sub(newGasLeft));
            }
```

BABYTOKENDividendTracker inherits the OwnableUpgradeable twice, once directly and once via DividendPayingToken.

IterableMapping: inserted[] not needed. IterableMapping library could save gas by getting rid of inserted[] mapping and using indexOf[] instead of it.

02. Lack of error messages

- Low ⓘ Acknowledged

Require statements in distributeCAKEDividends(), excludeFromDividends(), and SafeMathInt library functions lack error messages.

03. Denied transfers

- Informational ⓘ Acknowledged

All the transfers of DividendPayingToken are blocked which makes it a non-ERC20. It may be slightly confusing as many explorers will show DIVIDEND_TRACKER as an ERC20 token.

04. Event emits regardless success of transfer

- Informational ⓘ Acknowledged

The _withdrawDividendOfUser() function emits the DividendWithdrawn() event before checking the result of the transfer.

5. Conclusion

2 high severity issues were found and resolved by transferring the contract's ownership to the Timelock contract. The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on the code improving and preventing potential attacks.

By 2021-12-14 approximately 42% of Floshin's total supply and 81% of Floshin-WBNB LP supply were held by the PinkSale locker contract. Only ~3.5% of the total supply was held by the liquidity pair contract.

The reviewed code is available in the Binance Smart Chain:

0x9A6Fed601a11290500F8D76153C33CC254E9F6D0 Floshin token,

0x5279EdA97E2ADacA2c288b9CcEfB9a779d0281dA Dividend tracker.

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

 @hashexbot

 blog.hashex.org

 linkedin

 github

 twitter

HashEx
Blockchain Security