Crunch

smart contracts
final audit report

November 2022

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 Crunch-Network team to perform an audit of their smart contract. The audit was conducted between 01/11/2022 and 02/11/2022.

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 in the @Crunch-Network/crunch-token public GitHub repository after the commit 2b96592.

Update: the Crunch-Network team has responded to this report. The updated code is located in the @Crunch-Network/crunch-token GitHub repository after the e5c8203 commit.

The same code with minor modifications is deployed to Polygon network at address 0xf98b792C6A21cCD8A8dfe87463F13a81030c09B4.

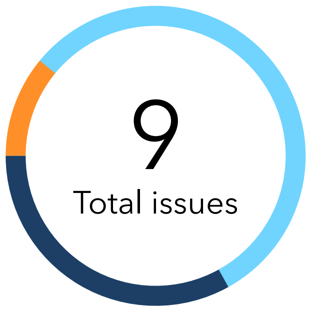
2.1 Summary

Project name	Crunch
URL	https://www.crunchnetwork.io/
Platform	Polygon Network
Language	Solidity

2.2 Contracts

Name	Address
CrunchToken	0xf98b792C6A21cCD8A8dfe87463F13a81030c09B4

3. Found issues



●	High	1 (11%)
●	Low	5 (56%)
●	Info	3 (33%)

C1. CrunchToken

ID	Severity	Title	Status
C1-01	High	Blocked transfers	Resolved
C1-02	Low	Fails with adding liquidity	Resolved
C1-03	Low	Re-entrancy with swapBack() function	Acknowledged
C1-04	Low	Few events	Resolved
C1-05	Low	Functions lacks validation of input parameters	Resolved
C1-06	Low	Gas optimization	Partially fixed
C1-07	Info	Floating pragma	Acknowledged
C1-08	Info	Missing functionality for multiple variables	Resolved
C1-09	Info	Lack of documentation (NatSpec)	Acknowledged

4. Contracts

C1. CrunchToken

Overview

The ERC20-like token with transfer fees. Some of these fees are aimed at maintaining liquidity in a pair of this and native tokens.

Issues

C1-01 Blocked transfers — High — Resolved

The `_transfer()` function adds a new recipient to the `holders` array every time if it was not there. In this case, the check for the presence of a user in the array occurs in a loop. In case the number of users becomes huge, this check can be broken due to the block gas limit. This will result in all transfers being blocked.

Moreover, the `holders` array functionality is not used anywhere else and there is not even a view function to get holders. (Also, the contract lacks the functionality to remove a holder from the `holders` array, and the `holders` functionality looks incomplete.)

Recommendation

Check that saving the holders to an array is really necessary. If necessary, an EnumerableSet should probably be used for this purpose.

C1-02 Fails with adding liquidity — Low — Resolved

When adding liquidity to a pair, an error may occur due to the fact that while transferring tokens to a pair, the `swapBack()` function is called. This feature also adds liquidity and refreshes balances and reserves on the pair.

So, after updating the balances of the pair (in the call to the `swapBack()` function), the router will again transfer tokens to the pair and try to call the mint function on the pair. This call may fail with "`INSUFFICIENT_LIQUIDITY_MINTED`" error.

Recommendation

We recommend adding an additional check `!automatedMarketMakerPairs[to]` on L318-L323, to prevent `swapBack()` when transferring tokens to a pair.

C1-03 Re-entrancy with swapBack() function

Low | Acknowledged

The `swapBack()` function converts the token balance into native tokens and then transfers native tokens to `devWallet`, `marketingWallet` using `call()` function. The `call()` function forwards all gas further without specifying the amount of it. If there are contracts behind the wallet addresses, this allows making a re-entrancy. In this case, the transaction can be carried out without paying commissions.

Recommendation

We recommend limiting the amount of gas transferred in call functions. This can be done by adding a global variable as well as a function to change it. Also, such a function must have upper and lower bounds for the `swapBack()` function to work correctly.

C1-04 Few events

Low | Resolved

The functions `updateSwapTokensAtAmount()`, `updateMaxAmount()`, `updateSwapEnabled()`, `updateBuyFees()`, `updateSellFees()`, `swapBack()` don't emit events, which complicates the tracking of important changes.

C1-05 Functions lacks validation of input parameters

Low | Resolved

The contract constructor and the `updateMarketingWallet()`and `updateDevWallet()` functions do not check the address parameters against the non-zero value.

C1-06 Gas optimization

Low | Partially fixed

a. The state variable `uniswapV2Router` , `uniswapV2Pair` can be declared as `immutable` to save gas.

b. The functions `setAutomatedMarketMakerPair()`, `isExcludedFromFees()` can be declared external to save gas.

c. The state variable `buyBackWallet` is never initialized and never used in the contract code. Consider removing the variable or adding functionality to it.

d. The sum of the input parameters of the `updateBuyFees()` and `updateSellFees()` functions should be checked before assigning them to state variables.

e. The `sellTotalFees` and `buyTotalFees` storage variables are read multiple times in the `_transfer()` function. Instead, use local variables to save gas.

f. The BoughtEarly, UpdateUniswapV2Router events are never used in the contract code.

g. `buyTotalFees` in `updateBuyFees()` should be the sum of the local variable in order to save gas, same for `sellTotalFees` in `updateSellFees`.

h. The `swapEnabled` bool storage variable should be packed with `swapping` bool variable.

Update

The `holders` variable is no longer used.

One of the `deltaEthBalance` and `ethBalance` variables in `swapBack()` function is redundant since they have the same value.

Consider removing unused variables.

C1-07 Floating pragma

● Info ⊘ Acknowledged

A general recommendation is that pragma should be fixed to the version that you are intending to deploy your contracts with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in future solidity releases.

C1-08 Missing functionality for multiple variables

● Info ⊘ Resolved

The contract has declared `maxTransactionAmount`, `maxWallet`, `_isExcludedMaxTransactionAmount` variables. These variables can only be updated. But at the same time, they are not involved in the functionality of the contract in any way.

Recommendation

Consider removing variables or adding functionality to them.

C1-09 Lack of documentation (NatSpec)

● Info ⊘ Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

5. Conclusion

1 high, 5 low severity issues were found during the audit. 1 high, 3 low issues were resolved in the update.

We strongly suggest adding unit tests for the contract.

This audit includes recommendations on improving the code and preventing 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