HashEx
Blockchain Security

NFTY

smart contracts
audit report

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

2 Overview

HashEx was commissioned by the NFTY Labs team to perform an audit of their smart contract. The audit was conducted between September 21 and September 24, 2021.

The code located in @nftylabs/token GitHub repository was audited after the 90bca47 commit. The repository contains no tests for audited contracts. The code was provided without documentation besides README.md. The recheck was done after the e77ee0f commit.

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 improving the security posture of smart contracts, by remediating the issues that were identified.

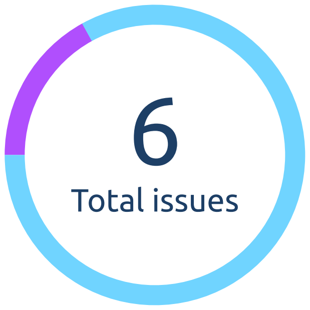
2.1 Summary

Project name	NFTY
URL	https://github.com/nftylabs/token/tree/90bca47217984c5a9a88071338044f0283e67fa9
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
NFTY	0xE1D7C7a4596B038CEd2A84bF65B8647271C53208

3 Found issues



■	Medium	1 (17%)
■	Low	5 (83%)

NFTY

ID	Title	Severity	Status
01	updateStreamRatios problem	■ Medium	Fixed
02	SafeMath	■ Low	Acknowledged
03	Gas optimization	■ Low	Partially fixed
04	Not enough events	■ Low	Partially fixed
05	Floating pragma	■ Low	Acknowledged
06	Very scarce documentation	■ Low	Fixed

4 Contracts

4.1 NFTY

4.1.1 Overview

The main contract, an implementation of the ERC20 standard. This token has a mechanism of minting tokens. There is a limited amount of tokens that are minted for every block. Owner of token sets 4 (or fewer) addresses that receive new tokens. All undistributed tokens are burned. Token has a cap on total supply. When the emission of the token reaches this amount, the process of minting more tokens is stopped.Also, the owner of the token can pause the emission of tokens and decrease the amount of tokens that are minted for one block.

4.1.2 Issues

01. updateStreamRatios problem

■ Medium ✓ Fixed

In the function updateStreamRatios() L1013 new percentages can be set for zero addresses (w1, w2, w3 and w4). If this happens, minting in function mineTo() L967 may fail because minting to zero address is not allowed.

Recommendation

We recommend adding similar logic like in function updateStreams() L1003.

02. SafeMath

▪ Low ⓘ Acknowledged

From 0.8.0 solidity there are embedded overflow and underflow checks in arithmetic operations and there is no need for using SafeMath library from OpenZeppelin. It is only an unnecessary waste of gas.

Recommendation

We recommend to remove SafeMath library from contract.

03. Gas optimization

▪ Low ◔ Partially fixed

- In the constructor, there is zero initialization of variable amountBurned. This is a waste of gas because all variables are initialized by zero.
- In the constructor, there is double initialization of variable lastBlockMined.

- In function mineTo() L967 there is a read of global variable lastBlockMined after write.
- In function mineTo() L967 there is a double read of the global variable auxContract.
- In function mineTo() L967 there is a read of the global variable amountBurned after its assignment.
- In function unpauseEmissions() L1027 there is a read after write of the global variable lastBlockMined
- In function decreaseEmissions() L1035 there is a read after write of the global variable amountMinedPerBlock

Recommendation

We recommend using local variables to store global values.

04. Not enough events

▪ Low ◎ Partially fixed

In the functions delegateAuth() L908, delegateVote() L914, updateStreams() L1003, updateStreamRatios() L1013 and setContract() L1042 there are no appropriate events.

Also is't a good practice to emit an Error event if execution fails in try/catch block. We recommend emittion such event in function mineTo() L967.

Recommendation

Add appropriate events

05. Floating pragma

■ Low ⓘ Acknowledged

Floating pragma is not recommended. It is better to use a fixed version.

Recommendation

Fix pragma version to avoid discrepancies in smart contracts behavior compiled with different Solidity versions (don't use ^ and use =).

06. Very scarce documentation

■ Low ✓ Fixed

Documentation provided by customer is very scarce.

Recommendation

Add documentation in NatSpec format for the contract code.

5. Conclusion

No critical or high severity issues were found. We strongly recommend adding tests with coverage of at least 90%, before the deployment to the mainnet.

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

Update: some issues were fixed after the update, statuses were updated.

6. 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.

7. Appendix B

- 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

8. Appendix C. Slither output

```
Compiled with solc

Number of lines: 1045 (+ 0 in dependencies, + 0 in tests)

Number of assembly lines: 0

Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 22

Number of informational issues: 20

Number of low issues: 4

Number of medium issues: 4

Number of high issues: 0

ERCs: ERC20

+-------------+-------------+-------+------------------+-------------+----------+

|    Name     | # functions | ERCS  |    ERC20 info    | Complex code | Features |

+-------------+-------------+-------+------------------+-------------+----------+

|  SafeMath   |     13      |       |                  |     No      |          |

| AuxContract |      8      |       |                  |     No      |          |

|    NFTY     |     50      | ERC20 |    No Minting    |    Yes      |          |
|           |           |       | Approve Race Cond. |            |         |

|           |           |       |                    |            |         |

+-------------+-------------+-------+--------------------+-------------+----------+
```

NFTY.getAmounts() (contracts/Contract_Unchanged.sol#920-958) uses a dangerous strict equality:

- tReward == 0 (contracts/Contract_Unchanged.sol#940)

NFTY.getAmounts() (contracts/Contract_Unchanged.sol#920-958) uses a dangerous strict equality:

- require(bool,string)(tReward == burnAmount.add(amt1).add(amt2).add(amt3).add(amt4),mismatch) (contracts/Contract_Unchanged.sol#956)

NFTY.mineTo(address) (contracts/Contract_Unchanged.sol#967-1000) uses a dangerous strict equality:

- amt2mine == 0 (contracts/Contract_Unchanged.sol#981)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

NFTY.getAmounts().burnAmount (contracts/Contract_Unchanged.sol#952) is a local variable never initialized

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

NFTY.updateStreamRatios(uint256,uint256,uint256,uint256) (contracts/

Contract_Unchanged.sol#1013-1016) should emit an event for:

- r1 = a1 (contracts/Contract_Unchanged.sol#1015)
- r2 = a2 (contracts/Contract_Unchanged.sol#1015)
- r3 = a3 (contracts/Contract_Unchanged.sol#1015)
- r4 = a4 (contracts/Contract_Unchanged.sol#1015)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

NFTY.updateStreams(address,address,address,address).a2 (contracts/Contract_Unchanged.sol#1003) lacks a zero-check on :

- w2 = a2 (contracts/Contract_Unchanged.sol#1005)

NFTY.updateStreams(address,address,address,address).a3 (contracts/Contract_Unchanged.sol#1003) lacks a zero-check on :

- w3 = a3 (contracts/Contract_Unchanged.sol#1005)

NFTY.updateStreams(address,address,address,address).a4 (contracts/Contract_Unchanged.sol#1003) lacks a zero-check on :

- w4 = a4 (contracts/Contract_Unchanged.sol#1005)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Different versions of Solidity is used:

- Version used: ['^0.8.0', '^0.8.2']

- ^0.8.0 (contracts/Contract_Unchanged.sol#61)

- ^0.8.0 (contracts/Contract_Unchanged.sol#145)

- ^0.8.0 (contracts/Contract_Unchanged.sol#173)

- ^0.8.0 (contracts/Contract_Unchanged.sol#199)

- ^0.8.0 (contracts/Contract_Unchanged.sol#555)

- ^0.8.0 (contracts/Contract_Unchanged.sol#627)

- ^0.8.2 (contracts/Contract_Unchanged.sol#852)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (contracts/Contract_Unchanged.sol#190-192) is never used and should be removed

SafeMath.div(uint256,uint256,string) (contracts/Contract_Unchanged.sol#814-823) is never used and should be removed

SafeMath.mod(uint256,uint256) (contracts/Contract_Unchanged.sol#774-776) is never used and should be removed

SafeMath.mod(uint256,uint256,string) (contracts/Contract_Unchanged.sol#840-849) is never used and should be removed

SafeMath.sub(uint256,uint256,string) (contracts/Contract_Unchanged.sol#791-800) is never used and should be removed

SafeMath.tryAdd(uint256,uint256) (contracts/Contract_Unchanged.sol#645-651) is never used and should be removed

SafeMath.tryDiv(uint256,uint256) (contracts/Contract_Unchanged.sol#687-692) is never used and should be removed

SafeMath.tryMod(uint256,uint256) (contracts/Contract_Unchanged.sol#699-704) is never used and should be removed

SafeMath.tryMul(uint256,uint256) (contracts/Contract_Unchanged.sol#670-680) is never used and should be removed

SafeMath.trySub(uint256,uint256) (contracts/Contract_Unchanged.sol#658-663) is never used and should be removed

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#61) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#145) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#173) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#199) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#555) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (contracts/Contract_Unchanged.sol#627) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.2 (contracts/Contract_Unchanged.sol#852) necessitates a version too

```
recent to be trusted. Consider deploying with 0.6.12/0.7.6

solc-0.8.7 is not recommended for deployment

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

AuxContract (contracts/Contract_Unchanged.sol#854-856) does not implement functions:

	- AuxContract.cron(address) (contracts/Contract_Unchanged.sol#855)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

name() should be declared external:

	- ERC20.name() (contracts/Contract_Unchanged.sol#255-257)

symbol() should be declared external:

	- ERC20.symbol() (contracts/Contract_Unchanged.sol#263-265)

decimals() should be declared external:

	- ERC20.decimals() (contracts/Contract_Unchanged.sol#280-282)

	- NFTY.decimals() (contracts/Contract_Unchanged.sol#889-891)

balanceOf(address) should be declared external:

	- ERC20.balanceOf(address) (contracts/Contract_Unchanged.sol#294-296)

transfer(address,uint256) should be declared external:

	- ERC20.transfer(address,uint256) (contracts/Contract_Unchanged.sol#306-309)
```

allowance(address,address) should be declared external:

- ERC20.allowance(address,address) (contracts/Contract_Unchanged.sol#314-316)

approve(address,uint256) should be declared external:

- ERC20.approve(address,uint256) (contracts/Contract_Unchanged.sol#325-328)

transferFrom(address,address,uint256) should be declared external:

- ERC20.transferFrom(address,address,uint256) (contracts/Contract_Unchanged.sol#343-357)

increaseAllowance(address,uint256) should be declared external:

- ERC20.increaseAllowance(address,uint256) (contracts/Contract_Unchanged.sol#371-374)

decreaseAllowance(address,uint256) should be declared external:

- ERC20.decreaseAllowance(address,uint256) (contracts/Contract_Unchanged.sol#390-398)

renounceOwnership() should be declared external:

- Ownable.renounceOwnership() (contracts/Contract_Unchanged.sol#603-605)

transferOwnership(address) should be declared external:

- Ownable.transferOwnership(address) (contracts/Contract_Unchanged.sol#611-614)

burn(uint256) should be declared external:

- NFTY.burn(uint256) (contracts/Contract_Unchanged.sol#894-897) delegateAuth(address) should be declared external:

- NFTY.delegateAuth(address) (contracts/Contract_Unchanged.sol#908-910)

delegateVote(address) should be declared external:

- NFTY.delegateVote(address) (contracts/Contract_Unchanged.sol#914-916)

mine() should be declared external:

- NFTY.mine() (contracts/Contract_Unchanged.sol#964-966)

updateStreams(address,address,address,address) should be declared external:

- NFTY.updateStreams(address,address,address,address) (contracts/Contract_Unchanged.sol#1003-1010)

updateStreamRatios(uint256,uint256,uint256,uint256) should be declared external:

- NFTY.updateStreamRatios(uint256,uint256,uint256,uint256) (contracts/Contract_Unchanged.sol#1013-1016)

pauseEmissions() should be declared external:

- NFTY.pauseEmissions() (contracts/Contract_Unchanged.sol#1020-1024)

unpauseEmissions() should be declared external:

- NFTY.unpauseEmissions() (contracts/Contract_Unchanged.sol#1027-1032)

decreaseEmissions(uint256) should be declared external:

- NFTY.decreaseEmissions(uint256) (contracts/Contract_Unchanged.sol#1035-1040)

setContract(address) should be declared external:

- NFTY.setContract(address) (contracts/Contract_Unchanged.sol#1042-1044)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

HashEx
Blockchain Security