#HashEx
BLOCKCHAIN SECURITY

NFTY Bubbs

smart contracts
final audit report

September 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 NFTY team to perform an audit of their smart contract. The audit was conducted between 24/09/2022 and 27/09/2022.

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 at @NFTYNetwork/bubbs GitHub repository after the 7d9531d commit.

Update: the NFTY team has responded to this report. The updated code is located in the same repository after the cfff836 commit.

Update 2: a slightly modified smart contract with one additional unused constructor parameter has been deployed to the BSC network at address 0xf8Fda2E9C66276C9d6a85510AD7186c704C616F3. The change in the deployed contract does not impose any additional security risks.

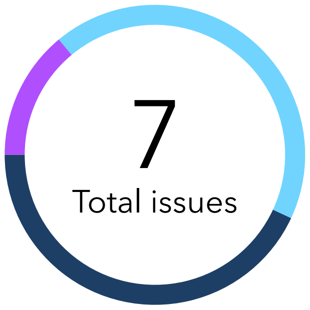
2.1 Summary

Project name	NFTY Bubbs
URL	https://nftynetwork.io
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
Bubbs	`0xf8Fda2E9C66276C9d6a85510AD7186c704C616F3`

3. Found issues



●	Medium	1 (14%)
●	Low	3 (43%)
●	Info	3 (43%)

C1. Bubbs

ID	Severity	Title	Status
C1-01	● Medium	Funds may get locked on the contract	Partially fixed
C1-02	● Low	Lack of error message	Acknowledged
C1-03	● Low	No events	Acknowledged
C1-04	● Low	Gas optimization	Partially fixed
C1-05	● Info	Royalties percentage may exceed 100%	Acknowledged
C1-06	● Info	Wallet may achieve max mint limit during presale stage	Acknowledged
C1-07	● Info	Documentation incompliance	Acknowledged

4. Contracts

C1. Bubbs

Overview

Extended 721A implementation of EIP-721 standard with presale and EIP-2981 royalty support. Uncommonly to the classic 'real-world' royalty realization, when the NFT creator or owner is benefited, the same address set by the owner is rewarded for all token IDs. The reward percentage is also set by the owner. Token issuance is payable functionality for users and free for the owner. Base token URI can be adjusted.

Issues

C1-01 Funds may get locked on the contract ● Medium Partially fixed

Functions `withdraw()`, `mint()`, and `presaleMint()` have a redundant `payable` modifier, although payments for minting methods are accepted only in ERC20 tokens, and withdrawal operation does not suppose native currency transfer at all. Accidentally sent cryptocurrency will stay locked on the contract.

```
function presaleMint(uint32 count, bytes32[] calldata proof)
  external
  payable
  preMintChecks(count)
{
  ...
  payableToken.transferFrom(msg.sender, address(this), cost * count);
  _safeMint(msg.sender, count);
}

function mint(uint256 count) external payable preMintChecks(count) {
  require(open == true, "Mint not open");
  payableToken.transferFrom(msg.sender, address(this), cost * count);
  _safeMint(msg.sender, count);
}

function withdraw() public payable onlyOwner {
  payableToken.transfer(msg.sender, payableToken.balanceOf(address(this)));
}
```

Recommendation

Consider removing all payable modifiers.

Update

`withdraw()` function has been updated, allowing the owner to collect native balance of the contract:

```
  function withdraw() external payable onlyOwner {
    uint256 balance = address(this).balance;
    if (balance > 0) {
      Address.sendValue(payable(msg.sender), balance);
    }

    payableToken.transfer(msg.sender, payableToken.balanceOf(address(this)));
  }
```

C1-02 Lack of error message ● Low ✓ Acknowledged

If the first requirement in the `airdrop()` function is not satisfied, the transaction will be reverted with an empty message. The absence of a transaction failure explanation may be confusing and will complicate problem investigation.

```
require(_recipients.length == _amount.length);
```

C1-03 No events ● Low ✓ Acknowledged

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in the `updateToken()`, `updatePresale()`, and `updateReveal()` functions.

C1-04 Gas optimization ● Low Partially fixed

a. `updatePresale()`, `updateReveal()`, `updateRoyalties()`, `withdraw()`, `airdrop()`, `numberMintedOfOwner()` functions should be external;

b. Two `if` clauses can be replaced with one `if-else` clause in `updateReveal()`;

c. `maxSupply` should be marked as `immutable`;

d. Multiple storage reads caused by `supply()` and `totalSupply()` calls in `airdrop()`.

Recommendation

d. State variables reads can be reduced with the following code snippet:

```
uint256 _supply = supply();
uint256 _totalSupply = totalSupply();
for (uint256 i = 0; i < _amount.length; i++) {
  _supply += _amount[i];
  require(_supply <= _totalSupply, "reached max supply");
  _safeMint(_recipients[i], _amount[i]);
}
```

Update

`a` and `d` have been fixed.

C1-05 Royalties percentage may exceed 100% ● Info Acknowledged

Royalties' percentage is not validated for excess of 100% during modification. This may lead to token sell failures on NFT markets; however, token transfers won't be affected.

C1-06 Wallet may achieve max mint limit during presale stage

● Info ✓ Acknowledged

Max per wallet limit on token issuing can be achieved before the main sale stage starts. It might be reasonable to guarantee a wallet has some available tokens for minting after the presale. Verification of MerkleProof doesn't limit the number of minting calls unless the total number of minted tokens isn't greater than the per wallet limit value.

C1-07 Documentation incompliance

● Info ✓ Acknowledged

NatSpec description of the ERC721A contract (and the ERC721Enumerable standard extension) from which the token is derived claims, that `totalSupply()` 'returns the total number of tokens in existence'. But the overridden function realization returns the max possible mint amount minus burned amount.

```
function totalSupply() public view override returns (uint256) {
    return maxSupply - _burnCounter;
}
```

5. Conclusion

1 medium, 3 low severity issues were found during the audit. No issues were resolved in the update.

We recommend adding documentation as well as unit and functional tests for all contracts.

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