#HashEx
BLOCKCHAIN SECURITY

NFC Market

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 NFC Market team to perform an audit of their smart contract. The audit was conducted between 29/09/2022 and 03/10/2022.

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 was provided in NFCPortal.sol file with 541772db867b24032ced7e58be1027d8 MD5 sum.

Update: the NFC Market team has responded to this report. The updated code is located in the @Kabyno44/solidity-nfcyborgs GitHub repository after the 333ab7e commit. The same code is deployed to Cronos networks at address 0xA6dC7630EA856c63Ce5380B2a4a6079ec34d8a23.

2.1 Summary

Project name	NFC Market
URL	https://nfc.market
Platform	Cronos Network
Language	Solidity

2.2 Contracts

Name	Address
NFCWrappedToken (NFCReplicaToken)	
NFCPortal	

3. Found issues

Medium	1 (20%)
Low	3 (60%)
Info	1 (20%)

C1. NFCWrappedToken (NFCReplicaToken)

ID	Severity	Title	Status
C1-01	Low	Gas optimization	Resolved

C2. NFCPortal

ID	Severity	Title	Status
C2-01	Medium	Lack of validation	Resolved
C2-02	Low	Lack of events	Resolved
C2-03	Low	Gas optimization	Resolved
C2-04	Info	Fee calculation	Partially fixed

4. Contracts

C1. NFCWrappedToken (NFCReplicaToken)

Overview

An implementation of the EIP-20 token standard built on the ERC20Burnable extension from OpenZeppelin. NFCReplicaToken is mintable by the owner. The token is deployed in the NFCPortal contract constructor, hence minting is allowed only by the contract.

Issues

C1-01 Gas optimization

Low Resolved

The `mint()` function can be declared as `external` to save gas.

C2. NFCPortal

Overview

The contract allows depositing NFT tokens for `10**18` `NFCReplicaToken` tokens and exchanging NFT tokens for previously deposited ones. The received `NFCReplicaToken` tokens can be burned in exchange for any deposited NFT. All tokens available on the platform have the same value, i.e. any deposited token can be changed to any others from the list of supported or to the amount of`NFCReplicaToken` tokens issued to the NFT depositors without surcharges. All operations are carried out with commissions paid by `feeToken`.

Issues

C2-01 Lack of validation

Medium Resolved

The contract owner can accidentally set the `startTimestamp` variable with an unreachable value (for example `block.timestamp + 100 years`) at any time using the `setStartTimestamp()` method. In this case, all operations will be blocked for anyone except the owner.

```
function depositNFTsForToken(uint256[] calldata _tokenIds) public nonReentrant {
    if(block.timestamp < startTimestamp){
        require (_msgSender() == owner(), "Portal not open for users");
    }
    ...
}

function burnTokenForNFTs(uint256[] calldata _tokenIds) public nonReentrant {
    if(block.timestamp < startTimestamp){
        require (_msgSender() == owner(), "Portal not open for users");
    }
    ...
}

function depositNFTsForNFTs(uint256[] calldata _tokenIds, uint256[] calldata
_tokenIdsWanted) public nonReentrant {
    if(block.timestamp < startTimestamp){
        require (_msgSender() == owner(), "Portal not open for users");
    }
    ...
}

function setStartTimestamp(uint256 _startTimestamp) public onlyOwner {
    require (block.timestamp < startTimestamp, "already started");
    startTimestamp = _startTimestamp;
}
```

Recommendation

Consider adding validation to the contract `constructor()` and `setStartTimestamp()` function, to prevent blocking contract functionality.

C2-02 Lack of events

Low Resolved

Governance functions `setExchangeFee()`, `setExchangeWallet()`, `setArtistWallet()`, `setStartTimestamp()`, `addTokenIDsAuthorized()` should emit corresponding events. It also complicates the tracking of important off-chain changes.

C2-03 Gas optimization

Low Resolved

a. The state variables `token`, `nft`, `isAllTokenIDsAuthorized` should be marked as `immutable`;

b. The state variables `feeToken`, `masterchef` can be declared as `constant`;

c. All `public` functions except `getArtistFee()` should have `external` visibility.

C2-04 Fee calculation

Info Partially fixed

The fee for the exchange of tokens is calculated using the `masterchef` contract on L98, L130, and L164. It is recommended that users who perform the exchange approve only the exact size of the fee. Otherwise, if the user approves the `type(unit).max` value, the `masterchef` contract owner can change the fees using the front-run pattern. As a result, the user may pay a much higher fee than expected.

Also, consider adding a view function to get the current fee amount from the `masterchef` contract.

Update

A fee view function was added. Now users can see the charged amount for one NFT deposit, withdrawal, or swap operation. Regard, the total fee depends on the length of the passed `_tokenIds` parameter to exchange functions.

```
function getUserFeeAmount(address _account) public returns (uint256) {
    return masterchef.getUserFee(_account);
}
```

5. Conclusion

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

We strongly suggest 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