#HashEx
BLOCKCHAIN SECURITY

Olympix NFT

smart contracts final audit report

August 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 Olympix team to perform an audit of their smart contract. The audit was conducted between 10/08/2022 and 12/08/2022.

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 the NFT_worldcup_foundationAddress.sol file with MD5 sum of 28761c39c71fbf98546e129b4080dbba.

Update: the Olympix team has responded to this report. The updated contract is available at 0x156C63a7b83da42Bd106A722bc08D09d1A7235C1 in the Ethereum mainnet.

2.1 Summary

Project name	Olympix NFT
URL	https://olympix.io
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
NFT	0x156C63a7b83da42Bd106A722bc08D09d1A7235C1

3. Found issues

Low	3 (50%)
Info	3 (50%)

C1. NFT

ID	Severity	Title	Status
C1-01	Low	Gas optimization	Resolved
C1-02	Low	Extra paid ETH is not returned	Resolved
C1-03	Low	Lack of validation of input parameters	Resolved
C1-04	Info	Lack of error messages	Resolved
C1-05	Info	Lack of events	Resolved
C1-06	Info	Floating Pragma	Resolved

4. Contracts

C1. NFT

Overview

Implementation of the ERC-721 standard built on ERC721Enumerable extension from OpenZeppelin library. Supports owner-governed URI updates and pausable sale with 3 whitelists for discount minting. The contract has been renamed to OlympixBall_NFT in the update.

Issues

C1-01 Gas optimization

Low Resolved

1. Since the `maxSupply` variable does not change in the contract, it can be declared a `const` to save gas.

2. The `withdraw()`, `withdrawFoundation()` functions are declared `payable`, but work with the ETH sent by the user is not carried out in the function.

3. Multiple or unnecessary reads of storage variables in:

- function `mint()`: `freeMintCount`, `maxFreeMintAmount`, `ogDiscountMintCount`, `maxOgMintAmount`, `discountMintCount`, `maxWlMintAmount`, `discountStartHeight` variables;- function `withdrawFoundation()`: `foundationAddress` variable;- function `setFreeWhitelistBySelf()`: `selfSetupFreeMintCurrentCount` variable;- function `setOgDiscountWhitelistBySelf()`: `selfSetupOgMintCurrentCount` variable;- function `setDiscountWhitelistBySelf()`: `selfSetupWlMintCurrentCount` variable

You can save some gas by creating a separate variable for some of the fields, or by accessing the `memory` copy of the variables.

4. The `mint()`, `walletOfOwner()`, `tokenURI()`, `setCost()`, `setMaxMintAmount()`, `setBaseURI()`,

`setBaseExtension()`, `pause()`, `setFreeWhitelistUser()`, `setDiscountWhitelistUser()`, `setOgDiscountWhitelistUser()`, `setFreeWhitelistUserList()`, `setDiscountWhitelistUserList()`, `setOgDiscountWhitelistUserList()`, `setOgDiscountCost()`, `setDiscountCost()`, `setStartHeight()`, `setFreeStartHeight()`, `setDiscountStartHeight()`, `setMaxFreeMintAmount()`, `setMaxOgMintAmount()`, `setMaxWlMintAmount()`, `removeDiscountWhitelistUser()`, `setFoundationAddress()`, `removeFreeWhitelistUser()`, `removeOgDiscountWhitelistUser()`, `setFreeWhitelistBySelf()`, `setOgDiscountWhitelistBySelf()`, `setDiscountWhitelistBySelf()`, `resetSelfWhiteListParam()`, `setSelfWhiteListParam()`, `withdraw()`, `withdrawFoundation()`, `setFreeWhitelistBySelf()`, `setOgDiscountWhitelistBySelf()`, `setDiscountWhitelistBySelf()`, `resetSelfWhiteListParam()`, `setSelfWhiteListParam()` functions can be declared as `external` to save gas.

C1-02 Extra paid ETH is not returned

Low Resolved

In the `mint()` payable function, if the user transfers more ETH than necessary, the difference will not be returned to the balance of `msg.sender`.

```
function mint(address _to, uint256 _mintAmount) public payable {
    ...
    require(msg.value >= cost * _mintAmount, " eth not  enough ");
    ...
}
```

C1-03 Lack of validation of input parameters

Low Resolved

Governance functions that update the contract's parameters lack sanity checks for new values against zeroes and extremely high values. For example, `selfSetupFreeMintMaxLimit*maxFreeMintAmount` must not exceed the `maxSupply`, otherwise, the whole supply would be minted for free.

C1-04 Lack of error messages

Info Resolved

We recommend using the `require()` statement returning an error message to simplify the debugging, see L69-72, L236, L244.

C1-05 Lack of events

Info Resolved

Many set and remove functions don't emit events. For example: `setCost()`, `setMaxMintAmount()`, `setBaseURI()`, `setBaseExtension()`, `pause()`, `setFreeWhitelistUser()`, `setDiscountWhitelistUser()`, `setOgDiscountWhitelistUser()`, `setFreeWhitelistUserList()`, `setDiscountWhitelistUserList()`, `setOgDiscountWhitelistUserList()`, `setOgDiscountCost()`, `setDiscountCost()`, `setStartHeight()`, `setFreeStartHeight()`, `setDiscountStartHeight()`, `setMaxFreeMintAmount()`, `setMaxOgMintAmount()`, `setMaxWlMintAmount()`, `removeDiscountWhitelistUser()`, `setFoundationAddress()`, `removeFreeWhitelistUser()`, `removeOgDiscountWhitelistUser()`, `setFreeWhitelistBySelf()`, `setOgDiscountWhitelistBySelf()`, `setDiscountWhitelistBySelf()`, `resetSelfWhiteListParam()`, `setSelfWhiteListParam()`.

C1-06 Floating Pragma

Info Resolved

The contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

5. Conclusion

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

The reviewed contract is 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 code improvement and the prevention of potential attacks. We recommend adding unit and functional tests with coverage of at least 90% with any updates in the future.

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