#HashEx
BLOCKCHAIN SECURITY

BurpToken

smart contracts final audit report

October 2021

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 CoinBurp team to perform an audit of changes in their smart contracts since the last audit in July 2021 [1]. The audit was conducted between September 21 and October 26, 2021.

The code located in the GitHub repository @coinburp/coinburp-contracts was audited after commit 1e0901.

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	BurpToken
URL	https://burptoken.com
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
RaffleAdminAccessControl	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/AccessControl/RaffleAdminAccessControl.sol
RarityRegisterAdminAccessControl	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/AccessControl/RarityRegisterAdminAccessControl.sol
Raffle	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/Raffle/Raffle.sol
RaffleAccessControl	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/Raffle/RaffleAccessControl.sol
ERC1155PrizeWrapper	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/Raffle/ERC1155PrizeWrapper.sol
NFTRarityRegister	https://github.com/coinburp/coinburp-contracts/blob/1e09012c9dbd4dc77f9b2b837df7de2d2a5a87e3/contracts/NFTRarityRegister/NFTRarityRegister.sol

3. Found issues

High	1 (9%)
Medium	3 (27%)
Low	6 (55%)
Info	1 (9%)

C24. RaffleAdminAccessControl

ID	Severity	Title	Status
C24I53	Medium	onlyManager excessive requirements	Resolved
C24I54	Low	Constructor parameters	Resolved
C24Ib3	Low	Duplicating functions	Acknowledged

C25. RarityRegisterAdminAccessControl

ID	Severity	Title	Status
C25I55	Medium	onlyManager excessive requirements	Resolved
C25I56	Low	Constructor parameters	Resolved
C25Ib2	Low	Duplicating functions	Acknowledged

C27. Raffle

ID	Severity	Title	Status
C27Iea	High	claimPrize() re-entrancy	Acknowledged
C27I57	Low	RaffleTicket address	Resolved
C27I58	Low	AccessControl setup is doubled	Resolved
C27Ieb	Info	Locked tokens	Acknowledged

C47. RaffleAccessControl

ID	Severity	Title	Status
C47Ia8	Medium	onlyMinter excessive requirements	Resolved

4. Contracts

C24. RaffleAdminAccessControl

Overview

AccessControl contract based on OpenZeppelin contract [2].

Issues

C24I53 onlyManager excessive requirements

Medium Resolved

The `onlyManager()` modifier is used in the `addRaffleManager()`, `removeRaffleManager()`, `addPrizeManager()` and `removePrizeManager()` functions in addition to the requirements from AccessControl's `grantRole()` and `revokeRole()` functions. Thus, these functions need to be called by the DEFAUL_ADMIN_ROLE bearer who also has to grant the manager role to himself.

Recommendation

Setup proper AccessControl model with role's admin. Don't forget that 0x00 DEFAULT_ADMIN_ROLE is its own admin.

Update

The issue was fixed. The onlyManager() modifier was replaced with onlyAdmin().

C24I54 Constructor parameters

Low Resolved

The `raffleOwner` input parameter of the constructor is required to be non-zero but never used nor even stored.

The `_managerOf` input parameter of the constructor is used only for events and couldn't be changed.

Update

The issue was fixed. raffleOwner became a default admin.

C24Ib3 Duplicating functions

Low Acknowledged

Public functions `isRaffleManager()`, `isPrizeManager()`, `addRaffleManager()`, `addPrizeManager()`, `renounceRaffleManager()`, `renouncePrizeManager()`, `removeRaffleManager()` and `removePrizeManager()` don't implement additional functionality over standard functions from AccessControl contract, except for emitting custom events. But these events aren't reliable as the same results of granting or revoking roles can be achieved by calling `grantRole()`, `revokeRole()` and `hasRole()` public functions.

C25. RarityRegisterAdminAccessControl

Overview

AccessControl contract based on OpenZeppelin contract [2].

Issues

C25I55 onlyManager excessive requirements

Medium Resolved

The `onlyManager()` modifier is used in `addRaffleManager()`, `removeRaffleManager()`, `addPrizeManager()` and `removePrizeManager()` functions in addition to the requirements from AccessControl's `grantRole()` and `revokeRole()` functions. Thus these functions need to be called by the DEFAUL_ADMIN_ROLE bearer who also has to grant the manager role to himself.

Recommendation

Setup proper AccessControl model with the role's admin. Don't forget that 0x00 DEFAULT_ADMIN_ROLE is its own admin.

Update

The issue was fixed. onlyManager() modifier was replaced with onlyAdmin(). RARITY_REGISTER_MANAGER_ROLE was removed.

C25I56 Constructor parameters

Low Resolved

The `owner` input parameter of the constructor is required to be non-zero but never used nor even stored.

The `_managerOf` input parameter of the constructor is set to `address(this)`, used only for events and couldn't be changed.

Update

The issue was fixed. The owner became a default admin.

C25Ib2 Duplicating functions

Low Acknowledged

Public functions `isRarityManager()`, `addRarityManager()`, `renounceRarityManager()` and `removeRarityManager()` don't implement additional functionality over standard functions from AccessControl contract except for emitting custom events. But these events aren't reliable as the same results of granting or revoking roles can be achieved by calling `grantRole()`, `revokeRole()` and `hasRole()` public functions.

C27. Raffle

Overview

Lottery contract randomized by Chainlink VRF.

Issues

C27Iea claimPrize() re-entrancy

High Acknowledged

Winner of single raffle with ERC1155 prize can claim all the prize tokens with the same tokenId in different transactions or with single call to `claimPrize()` with re-entrancy from ERC1155 `safeTransferFrom()`. The `safeTransferFrom()` function invokes `onERC1155BatchReceived()` hook which can be used for reentrancy attack.

Recommendation

Add ReentrancyGuard and check the status of `raffleInfo[raffleIndex].prizes[prizeIndex].claimed` before claiming in the claimPrize

Update

As the Raffle contract is already deployed, an ERC1155PrizeWrapper.sol contract was created to wrap ERC1155 prizes to ERC721 tokens and use them as prizes.

C27I57 RaffleTicket address

Low Resolved

RaffleTicket address should be named in mixedCase to conform with Solidity naming conventions. It should also be declared immutable as it's set only in the constructor.

Update

The issue was fixed. Variable was modified according recommendations.

C27I58 AccessControl setup is doubled

Low Resolved

`_setupRole()` initializes twice: in RaffleAdminAccessControl and then in Raffle constructor.

Update

The issue was fixed. Initialization in the Raffle contract was removed.

C27Ieb Locked tokens

Info Acknowledged

`onERC721Received()`, `onERC1155BatchReceived()`, `onERC1155Received()` and `supportsInterface()` functions of Raffle contract could lead to locked third-party ERC721 and ERC1155 tokens.

C47. RaffleAccessControl

Overview

AccessControl contract based on OpenZeppelin contract [2].

Issues

C47Ia8 onlyMinter excessive requirements

Medium Resolved

The `onlyMinter()` modifier is used in `addMinter()` function in L54 in addition to the requirements from AccessControl's `grantRole()`. Thus that function needs to be called by the DEFAUL_ADMIN_ROLE bearer who also has to grant the manager role to himself.

Recommendation

Setup proper AccessControl model with role's admin. Don't forget that 0x00 DEFAULT_ADMIN_ROLE is its own admin.

Update

The `onlyMinter()` modifier was changed to onlyAdmin() which checks the same conditions and `grantRole()` and can be omitted for gas savings.

C62. ERC1155PrizeWrapper

Overview

A token wrapper contract for making ERC1155 tokens compatible with ERC721 Raffle.

Issues

No issues were found.

C26. NFTRarityRegister

Overview

Contract for NFT registration.

5. Conclusion

One high severity issue was found regarding reentrancy in Raffle with ERC1155 tokens in some cases. A wrapper contract was implemented to wrap ERC1155 to ERC721 tokens to use in the Raffle. The contracts are 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.

Audit includes recommendations on the code improving 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

8. References

1. CoinBurp audit by HashEx
2. AccessControl by OpenZeppelin

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY