#HashEx
BLOCKCHAIN SECURITY

InVaria2222

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 InVarFinance team to perform an audit of their smart contract. The audit was conducted between 19/08/2022 and 23/08/2022.

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 is available at the @InVarFinance/invaria2222 GitHub repository and was audited after the commit 81b14fe1 (audited file InVaria-staking.sol).

Update: the InVarFinance team has responded to this report. The updated code is located in the GitHub repository after the commit 6b5c42d.

Update 2: The audited InVariaStaking contract with additional events was deployed to Ethereum chain at address 0x10a92B12Da3DEE9a3916Dbaa8F0e141a75F07126.

2.1 Summary

Project name	InVaria2222
URL	https://app.invar.finance/
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
InVariaStaking	0x10a92B12Da3DEE9a3916Dbaa8F0e141a75F07126

3. Found issues

High	1 (11%)
Low	6 (67%)
Info	2 (22%)

C1. InVariaStaking

ID	Severity	Title	Status
C1-01	High	Exaggerated owner's rights	Resolved
C1-02	Low	Lack of events	Partially fixed
C1-03	Low	Lack validation of input parameters	Partially fixed
C1-04	Low	Division before multiplication	Acknowledged
C1-05	Low	Floating pragma	Acknowledged
C1-06	Low	Gas optimization	Partially fixed
C1-07	Low	Left to burn amount possible miscalculation	Open
C1-08	Info	Lack of documentation	Acknowledged
C1-09	Info	No guaranteed rewards	Acknowledged

4. Contracts

C1. InVariaStaking

Overview

The contract allows staking ERC1155 tokens with id1. Users can get rewards in USDC tokens during the staking period. Also, users are able to burn their tokens a year later after staking. As a reward for burning, users will get compensation in USDC tokens.

Issues

C1-01 Exaggerated owner's rights

High Resolved

a. The contract owner has the ability to change the reward token (USDC) at any time using the `setAddress()` function. If the owner accidentally or maliciously changes the `USDC` address, users will receive rewards in completely different tokens that are worth nothing;

b. The contract owner has the ability to change staking token at any time using the `setAddress()` function. If the owner accidentally or maliciously change `InVariaNFT` and `InVariaNFTBurn` addresses, users' NFTS will be locked in the contract and won't be available for burning.

Recommendation

a. Make `USDC` variable constant;

b. Make `InVariaNFT` and `InVariaNFTBurn` variables constant.

C1-02 Lack of events

Low Partially fixed

The functions `setAddress()`, `stakeNFT()`, `unStake()`, `withDraw()`, `BurnNFT()` don't emit events, which complicates the tracking of important off-chain changes.

C1-03 Lack validation of input parameters

Low Partially fixed

The contract constructor and the `setAddress()` function don't check the addresses for a non-zero value.

We recommend adding validation to prevent incorrect initialization and behavior of the contract.

C1-04 Division before multiplication

Low Acknowledged

Division before multiplication can, in some narrow cases, cause calculation errors in integer math. In this contract, division before multiplication is performed on L182 (`StakingReward_Balance()` function).

Make sure the calculations work as expected.

C1-05 Floating pragma

Low Acknowledged

A general recommendation is that pragma should be fixed to the version that you are intending to deploy your contracts with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in future solidity releases.

C1-06 Gas optimization

Low Partially fixed

a. The state variables `AprByMin`, `BurnReturn`, `YearInSeconds`, `WithDrawAddress` can be declared as constant to save gas;

b. The state variable `WithDrawAddress` is never used in the contract and can be removed, to save gas on deployment;

c. The `USDC_Balance()`, `BurnNftInfo()` functions can be declared as `external` to save gas;

d. The variables `staketime`, `unstaketime` and `isUnstake` of the `StakingInfo` structure can be

packed into one slot (by casting time variables to `uint64`);

e. The variables `locktime` and `isBurn` of the `BurningInfo` structure can be packed into one slot (by casting time variable to `uint64`);

f. The values `stakingInfo[stakingAddress].length` (L109, L143, L176) and `burningInfo[msg.sender].length` (L148) can be stored in a local variable to save gas on every loop step;

g. `InVariaNFT` and `InVariaNFTBurn` can be presented by one variable if `BurnFunction` interface is derived from `IERC1155`.

C1-07 Left to burn amount possible miscalculation

Low Open

NFTs in a `burningInfo` instance can be burned with `BurnNFT()` when the records' `lockTime` parameter is strictly less than `block.timestamp`. However, while calculating the NFTs amount available for burning in `BurnNftInfo()`, NFTs of records with `lockTime` equal to `block.timestamp` are also summed. This may affect the contracts with burned amount dependent logic. The issue was introduced in the update.

C1-08 Lack of documentation

Info Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

C1-09 No guaranteed rewards

Info Acknowledged

It is assumed that users will receive rewards for staking and burning tokens. But there is no guarantee that there are enough rewards on the balance of the contract at any time, since the source of the rewards is not explicitly known. Thus, before starting to use the contract, users must make sure that the contract has enough rewards for both staking and burning. Also, they

should be informed that the contract owner can withdraw `USDC` tokens before the rewards distribution ends.

5. Conclusion

1 high, 2 medium, 5 low, and 1 informational severity issues were found.

1 high severity issue has been resolved in the update. 3 low severity issues have been partially resolved.

The reviewed contract is highly dependent on the owner's account. As stated in the project description, users can earn rewards by staking, and can also burn NFT after the lock-up period and return the initial mint cost. But taking into account the implementation of the smart contract, the payment of **any reward (including for burning NFT) is not guaranteed**. And also, the owner can withdraw rewards from the contract at any time. Users using the project have to trust the owner and that the owner's account is properly secured.

We strongly suggest adding functional tests for the contract to cover all edge cases.

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