HashEx
BLOCKCHAIN SECURITY

PASS: InVariant

smart contracts
final audit report

January 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer 3

2. Overview 4

3. Found issues 6

4. Contracts 7

5. Conclusion 10

Appendix A. Issues severity classification 11

Appendix B. Issue status description 12

Appendix C. List of examined issue types 13

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). HashEx has exclusive rights to publish the results of this audit on company's web and social sites.

2. Overview

HashEx was commissioned by the Invaria2222 team to perform an audit of their smart contract. The audit was conducted between 2023-01-20 and 2023-01-25.

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 https://github.com/InVarFinance/invar-pass repository and was audited after the commit b1d5a08. A recheck was done after the commit 51d7044.

Update. The audited code was deployed to Ethereum Network at address 0x0e4f563103a6c7b624e3017958a9823c7e7dd4f9.

2.1 Summary

Project name	PASS: InVariant
URL	https://app.invar.finance/invaria2222
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
InVarPass	0x0e4f563103a6c7b624e3017958a9823c7e7dd4f9
IPass	
IPassConstants	

3. Found issues

5
Total issues

●	Low	4 (80%)
●	Info	1 (20%)

C1. InVarPass

ID	Severity	Title	Status
C1-01	● Low	Lack of input validation	✓ Resolved
C1-02	● Low	Gas optimizations	✓ Partially fixed
C1-03	● Low	Lack of events	✓ Resolved
C1-04	● Low	Possible transaction fail without a reason	✓ Resolved
C1-05	● Info	Unconventional naming	✓ Resolved

4. Contracts

C1. InVarPass

Overview

An implementation of the ERC721 token standard built on ERC721Enumerable extension by OpenZeppelin. Supports 4 types of minting: free mint and whitelist mint are available for whitelisted users (in separate Merkle trees), public mint is limited to less than 3 tokens per address, premium mint allows converting 2 pre-selected (with separate Merkle tree) tokens to a new token with ID from premium range.

Issues

C1-01 Lack of input validation ● Low ✓ Resolved

Input parameters of governance functions (including constructor) aren't validated. For example, the start of premium token enumeration could be set lower than `MAX_SUPPLY`, meaning premium tokens would be mintable with public mint.

Recommendation

We recommend adding reasonable requirements for input data of `setBaseUri()`, `setPremium()`, `setMaxSupply()`, and constructor section.

C1-02 Gas optimizations ● Low ✓ Partially fixed

1. The variables `trees.freemintMerkleRoot`, `trees.whitelistMerkleRoot`, `mintRecords[msg.sender].publicMinted` are read multiple times from storage in the `freeMint()`, `whitelistMint()`, `publicMint()` functions. Local variables may save gas on reading.
2. Unchecked math could be used in the `_refundIfOver()` function inside the `if(msg.value > _price)` block.

C1-03 Lack of events ● Low ✓ Resolved

Governance functions `setSaleConfig()`, `setMerkleRoot()`, `setBaseUri()`, `setPremium()`, and `setMaxSupply()` don’t emit events, which complicates off-chain tracking of important changes.

C1-04 Possible transaction fail without a reason ● Low ✓ Resolved

A mismatch of array lengths is possible in the `premiumMint()` function. Input parameters `_proofs` and `_tokens` must be at least 2 in length, but there's no check for that. Failing without reason transactions significantly complicate debugging.

Update

A check

```
if (_proofs.length != _tokens.length) revert LengthMismatch();
```

was added.

C1-05 Unconventional naming ● Info ✓ Resolved

The `MAX_SUPPLY` variable is not a constant one, it can be updated with the `setMaxSupply()` function. This is a contradiction to Solidity naming conventions.

Update

Developers removed the possibility to change the variable in the code update.

C2. IPass

Overview

Interface for InVarPass contract. No issues were found.

C3. IPassConstants

Overview

Abstract contract containing constants for InVarPass contract. No issues were found.

5. Conclusion

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

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.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. 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