#HashEx
BLOCKCHAIN SECURITY

Polarys Vesting

smart contracts final audit report

October 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 PolarysDAC team to perform an audit of their smart contract. The audit was conducted between 18/10/2022 and 19/10/2022.

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 is available in the @PolarysDAC/polarys-contracts public GitHub repository after the commit eeda6d3.

Update: the PolarysDAC team has responded to this report. The updated code is located in the @PolarysDAC/polarys-contracts GitHub repository after the ed0416c commit.

2.1 Summary

Project name	Polarys Vesting
URL	https://www.polarys.io/
Platform	Metis
Language	Solidity

2.2 Contracts

Name	Address
PolarToken	
PolarDepositContract	
TokenVesting	

3. Found issues

High	2 (25%)
Medium	1 (13%)
Low	2 (25%)
Info	3 (37%)

C1. PolarToken

ID	Severity	Title	Status
C1-01	High	Pausing	Acknowledged

C2. PolarDepositContract

ID	Severity	Title	Status
C2-01	High	Replay of depositToken()	Resolved
C2-02	Info	Sale status parameter	Acknowledged

C3. TokenVesting

ID	Severity	Title	Status
C3-01	Medium	Unreleased tokens after revoking	Resolved

C3-02	Low	Function lacks validation of input parameters	Partially fixed
C3-03	Low	Gas optimization	Resolved
C3-04	Info	Typos	Resolved
C3-05	Info	Commented line of code	Resolved

4. Contracts

C1. PolarToken

Overview

The ERC20 token with additional functionalities: sale status, pausable and burnable.

Issues

C1-01 Pausing

High Acknowledged

The contract owner has the ability to pause all transfers using the `pause()` function. This may lead to the users who bought tokens not being able to dispose of them or, for example, not being able to withdraw them from the TokenVesting contract (using `release()` function).

Also, a problem may arise if the owner's private key is compromised.

Recommendation

Without documentation, it is impossible to determine how necessary such (pause) functionality is. Perhaps this is only needed at certain stages of the sale.

Consider limiting the pause feature to the owner at any time, or removing it altogether.

C2. PolarDepositContract

Overview

The contract allows depositing ERC20 tokens with specifying the quantity of NFT tokens. Only users with the Deposit role and with a special signature can make a deposit. Such signatures are provided by third-party applications.

Users who have admin permission can withdraw deposited ERC20 tokens to any address.

According to the developer, the contract allows accepting payment for NFT in other chains. After payment, the contract emits events that must be processed by third-party applications. Based on these events, users will be able to receive NFT tokens on other networks.

At the same time, the audit team did not audit third-party applications.

Issues

C2-01 Replay of depositToken()

High Resolved

A user who has a signature that allows the function `depositToken()` to be executed can replay his call several times until the deadline arrives. This will emit multiple events with the same parameters.

```
    function depositToken(
        uint256 quantity,
        uint256 amount,
        uint256 deadline,
        uint256 status,
        bytes calldata signature,
        bytes32[] calldata merkleProof
    ) external {
        ...
        require(_verify(_hash(_msgSender(), quantity, amount, deadline, status),
signature), "Invalid signature");
        ...
        require(MerkleProof.verify(merkleProof, merkleRoot,
keccak256(abi.encodePacked(_msgSender()))), 'MerkleDistributor: Invalid proof.');
        ...
    }
```

Recommendation

We recommend verifying the signature against reuse. For example, using a nonce for each account.

C2-02 Sale status parameter

● Info ✓ Acknowledged

Without documentation, it is impossible to speak about the correct usage of the `status` parameter in the `depositToken()` function. Perhaps this value should not be a function parameter and should be taken from the PolarToken contract using the `getSaleStatus()` function to get the actual sale status.

Recommendation

We recommend updating the documentation and making sure the `depositToken()` function works as intended.

Update

According to the developer PolarDepositContract and PolarToken will be deployed on different chains. Therefore, the status parameter will be determined using a third-party application.

C3. TokenVesting

Overview

The contract is used to distribute (vest) tokens to certain users.

Contract admins with `VESTING_ROLE` can create and revoke PolarysToken vestings for any user. The admin can revoke the vesting at any time. At the same time, tokens for which the release time has not yet come will be returned to the owner of the contract (to an address controlled by him).

Issues

C3-01 Unreleased tokens after revoking

Medium Resolved

The contract owner can revoke specific vesting using the `revoke()` function. At the same time, a portion of the tokens (`unreleased` amount) can be temporarily blocked on the contract until the owner creates new vesting for this unreleased amount.

Problems may also occur when the owner's private key is compromised. In this case, the malicious user with owner rights will be able to cancel user vestings and create vestings for controlled addresses for the `unreleased` amounts of the tokens.

Recommendation

1. Make sure the function works as expected. Regard burning or sending the unreleased token amounts back to the treasury (owner) during revocation.

2. Consider transferring ownership to a multi-sig contract to prevent the immediate consequences of a single private key being compromised.

C3-02 Function lacks validation of input parameters

Low Partially fixed

The function `createVestingSchedule()` does not validate the values of the `start` and `_cliffDuration` variables.

Consider adding upper limits for them.

C3-03 Gas optimization

Low Resolved

a. The `createVestingSchedule()`, `computeReleasableAmount()`, `revoke()` functions can be declared as external to save gas.

b. The `vestingSchedule.beneficiary` variable reads twice in the `release()` function. Its value can be stored in a local variable.

c. Check for revoke on L125, L148 can be implemented after declaring the `vestingSchedule` variable. It can save gas.

C3-04 Typos

Info Resolved

Typos reduce the code readability:L16 'BeneficiayrOrOwner' should be replaced with 'BeneficiaryOrOwner'

C3-05 Commented line of code

Info Resolved

There is a commented line of code on L35. This may indicate that the code is incomplete.

5. Conclusion

2 high, 1 medium, 2 low severity issues were found during the audit. 1 high, 1 medium, 1 low issues were resolved in the update.

The contract PolarDepositContract is highly dependent on third-party applications. The contract PolarToken is highly dependent on the owner. Users using the project have to trust the owner (as well as the project team) and be sure that the third-party applications work properly.

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.

Note: no third-party application has been audited by the audit team.

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