

CERUS

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. List of examined issue types	12

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 CERUS team to perform an audit of their smart contract. The audit was conducted between 2023-01-03 and 2023-01-07.

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 <https://github.com/CERUS-Nodes/CERUS-Contracts> GitHub repository and was audited after the commit [aa089c2](#).

Update. The CERUS team has responded to this report, the updated code is located in the same repository after the [63372d6](#) commit.

2.1 Summary

Project name	CERUS
URL	https://cerusnodes.io
Platform	Metis
Language	Solidity

2.2 Contracts

Name	Address
CERUSToken	
CERUSVesting	

3. Found issues



● Critical	1 (20%)
● Medium	1 (20%)
● Low	1 (20%)
● Info	2 (40%)

Cc3. CERUSToken

ID	Severity	Title	Status
Cc3laa	● Critical	Tokens can be burned from anyone	✓ Resolved
Cc3l7b	● Info	Tax distributor can be changed	✓ Resolved
Cc3ld6	● Info	Discrepancy in documentation and code behaviour	✓ Resolved

Ccf. CERUSVesting

ID	Severity	Title	Status
Ccf16a	● Medium	Claim is not guaranteed	⊙ Acknowledged
Ccf168	● Low	Gas optimizations	⚙️ Partially fixed

4. Contracts

Cc3. CERUSToken

Overview

An ERC-20 [standard](#) implementation with taxable transfers. Tax is applied with several conditions: the recipient has to be a contract with an implemented `token1()` method, tax recipient and tax percent must have non-zero values, and the spender must be tax-applicable, i.e. not included in the whitelist. The tax percent is updatable with a default value of 5%; the updated value can't exceed 5%.

Issues

Cc3laa Tokens can be burned from anyone

● Critical✓ Resolved

In the updated contract the `burn()` and `burnFrom()` functions were introduced. The latter can be used by an arbitrary caller to burn up to 20% of total supply from any account regardless the allowance.

```
function burnFrom(address _account, uint256 _amount) public override {
    require(totalBurned() + _amount <= MAX_BURNED, "burn: max burn!");
    _burn(_account, _amount);
}
```

Recommendation

Reduce current allowance for caller or remove the `burnFrom()` function.

Cc3l7b Tax distributor can be changed

● Info✓ Resolved

A tax from sales is sent to the distributor contract. Users should be aware that the distributor contract address can be changed as well as the distribution itself.

Cc3ld6 Discrepancy in documentation and code behaviour

● Info

✓ Resolved

The documentation states:

Given that it [TaxDistributor] is not set, when transfer is called, then it should revert with an error.

The actual behavior is when the TaxDistributor address is not set, the tax won't be taken.

Recommendation

Update documentation to conform to the actual behavior.

Ccf. CERUSVesting

Overview

A sale contract with a whitelist and fixed price. Purchased CERUS tokens are vested within a fixed number of claimable periods of 4 weeks long each.

Issues

Ccfl6a Claim is not guaranteed

● Medium

✓ Acknowledged

The total amount of sale tokens is written during the contract initialization but the actual balance is not checked. Thus, the `claim()` function may constantly fail for some or even every user since the contracts will try to transfer nonexistent tokens.

```
function initialize(  
    ...  
    uint256 _totalCerus  
) external {  
    ...  
    totalCerus = _totalCerus;
```



```
}  
  
function claim() external nonReentrant {  
    ...  
    IERC20(cerusToken).transfer(msg.sender, _claimableAmount);  
}
```

Recommendation

Ensure the **totalCerus** amount during the initialization either by checking the balance of the contract or by transferring them from the initializer.

Ccfl68 Gas optimizations

● Low

🔧 Partially fixed

1. The variables **treasury**, **investorInfo[].depositedAmount**, **totalVested**, and **cerusPrice** in the **deposit()** function are read from storage multiple times, which can be avoided by using local variables.
2. The variables **cerusToken**, **investorInfo[].claimedEpochs**, **investorInfo[].vestedAmount**, **totalEpochs**, and **claimStart** in the **claim()** function are read from storage multiple times, which can be avoided by using local variables.
3. The variable **isPublicSale** in the **withdrawUnsold()** function is read from storage multiple times, which can be avoided by using local variables.
4. The variable **claimStart** in the **currentEpoch()** function is read from storage multiple times, which can be avoided by using local variables.
5. In the case of a long whitelist, a Merkle tree pattern can be used to save gas.
6. The variables **initialized**, **totalEpochs**, and **epochLength** in the **currentEpoch()** function are read from storage multiple times, which can be avoided by using local variables. This issue was introduced in the code update.

5. Conclusion

1 critical, 1 medium, 1 low severity issues were found during the audit. 1 critical issue was 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.

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](https://t.me/hashex_manager)

 blog.hashex.org

 [linkedin](https://www.linkedin.com/company/hashex)

 [github](https://github.com/hashex)

 [twitter](https://twitter.com/hashex)

#HashEx
BLOCKCHAIN SECURITY