HashEx
BLOCKCHAIN SECURITY

0xACID

smart contracts final audit report

April 2023

🌐 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 0xACID team to perform an audit of their smart contract. The audit was conducted between 13/04/2023 and 21/04/2023.

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.

Some of audited contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

The code is available at https://github.com/CERUS-Nodes/CERUS-Contracts GitHub repo and was audited after the commit b2ba6da.

Update. A recheck was done after the commit a2fd35c.

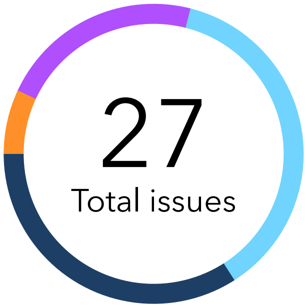
2.1 Summary

Project name	0xACID
URL	https://0xacid.com
Platform	Arbitrum Network
Language	Solidity

2.2 Contracts

Name	Address
MintableToken	
BaseToken	
esAcid	
Acid	
VesterV2	
RewardsDistributor	
RewardTracker	
StakingYieldPool	
AcidBondV4	
BondHelper	

3. Found issues



●	High	2 (7%)
●	Medium	6 (22%)
●	Low	10 (37%)
●	Info	9 (34%)

C1. MintableToken

ID	Severity	Title	Status
C1-01	● Medium	Overpowered privileged accounts	✓ Acknowledged
C1-02	● Low	Lack of events	✓ Acknowledged
C1-03	● Low	Missing error message	✓ Acknowledged

C2. BaseToken

ID	Severity	Title	Status
C2-01	● Medium	Overpowered privileged accounts	✓ Acknowledged
C2-02	● Info	Lack of events	✓ Acknowledged

C5. VesterV2

ID	Severity	Title	Status
C5-01	High	Centralization problem	Acknowledged
C5-02	Medium	Initialization problem	Resolved
C5-03	Medium	Wrong imports	Resolved
C5-04	Medium	Upgradeability problem	Acknowledged
C5-05	Info	Lack of events	Acknowledged

C6. RewardsDistributor

ID	Severity	Title	Status
C6-01	Low	Gas optimizations	Resolved
C6-02	Info	Lack of events	Acknowledged
C6-03	Info	Typos	Resolved

C7. RewardTracker

ID	Severity	Title	Status
C7-01	High	Centralization problem	Acknowledged
C7-02	Low	Gas optimizations	Resolved
C7-03	Low	Lack of validation	Acknowledged
C7-04	Info	Lack of events	Acknowledged

C7-05	● Info	Typos	✓ Resolved

C8. StakingYieldPool

ID	Severity	Title	Status
C8-01	● Medium	Upgradeability problem	✓ Acknowledged
C8-02	● Low	Gas optimizations	✓ Resolved
C8-03	● Low	Unused parameter	✓ Resolved
C8-04	● Info	Lack of events	✓ Acknowledged

C9. AcidBondV4

ID	Severity	Title	Status
C9-01	● Low	Lack of events	✓ Acknowledged
C9-02	● Low	Gas optimizations	✓ Resolved
C9-03	● Info	Typos	✓ Resolved
C9-04	● Info	Irrelevant and inconsistent comments	✓ Resolved

C10. BondHelper

ID	Severity	Title	Status
C10-01	● Low	Gas optimizations	✓ Resolved

4. Contracts

C1. MintableToken

Overview

An ERC-20 standard token with privileged addresses for minting and burning.

Issues

C1-01 Overpowered privileged accounts ● Medium ✓ Acknowledged

The `mint()` function is accessible for the owner-appointed list of addresses with optional individual mint caps.

The `burn()` function can be called by any privileged account from `isHander[]` list to burn user tokens ignoring ERC20 allowance.

```
function setHandler(address _handler, bool _status) external onlyOwner {
    isHandler[_handler] = _status;

function setMinter(address _minter, uint allowance) external onlyOwner {
    minter[_minter] = allowance;

function mint(address _account, uint256 _amount) external returns (uint) {
    if (minter[msg.sender] < type(uint).max) {
        if (_amount > minter[msg.sender]) {
            _amount = minter[msg.sender];
        }
        minter[msg.sender] -= _amount;
    }
    _mint(_account, _amount);
    return _amount;
}

function burn(address _account, uint256 _amount) external returns (uint) {
    require(isHandler[msg.sender], "!auth");
```

```
        _burn(_account, _amount);
        return _amount;
    }
```

Recommendation

The contract's ownership should be transferred to a Timelock-like contract or renounced completely. The list of minters and burners must be accessible to the public control.

C1-02 Lack of events

● Low ✓ Acknowledged

The functions `setHandler()` and `setMinter()` don’t emit events, which complicates off-chain tracking of important changes.

C1-03 Missing error message

● Low ✓ Acknowledged

There's no error message for minting a bigger amount than allowed (exceeding the cap).

```
    function mint(address _account, uint256 _amount) external returns (uint) {
        if (minter[msg.sender] < type(uint).max) {
            if (_amount > minter[msg.sender]) {
                _amount = minter[msg.sender];
            }
            minter[msg.sender] -= _amount;
        }
        _mint(_account, _amount);
        return _amount;
    }
```

C2. BaseToken

Overview

An ERC-20 standard token with privileged addresses for minting, burning, pausing, and transfers without spending allowance.

Issues

C2-01 Overpowered privileged accounts ● Medium ✓ Acknowledged

Transfers are pausable - the owner can switch the pause state by changing the `inPrivateTransferMode` variable.

Privileged accounts from the `isHandler[]` list can transfer tokens ignoring the ERC20 allowance.

```
    function setInPrivateTransferMode(bool _mode) external onlyOwner {
        inPrivateTransferMode = _mode;
    }

    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal override {
        if (!isHandler[msg.sender]) {
            ERC20._spendAllowance(owner, spender, amount);
        }
    }

    function _beforeTokenTransfer(
        address,
        address,
        uint256
    ) internal override virtual {
        if (inPrivateTransferMode) {
            require(
                isHandler[msg.sender] || minter[msg.sender] > 0,
                "BaseToken: msg.sender is not whitelisted"
            );
        }
    }
```

Recommendation

The contract's ownership should be transferred to a Timelock-like contract or renounced completely. The list of minters and burners must be accessible to public control.

C2-02 Lack of events

● Info ✓ Acknowledged

The function `setInPrivateTransferMode()` doesn't emit events, which complicates the off-chain tracking of important changes.

C3. esAcid

Overview

An ERC-20 standard token with privileged addresses for minting, burning, pausing, and transfers without spending allowance, inherited from the BaseToken contract.

C4. Acid

Overview

An ERC-20 standard token with privileged addresses for minting and burning, inherited from the MintableToken contract.

C5. VesterV2

Overview

An ERC-20 standard token with generally disabled transfers. User deposits esACID tokens and pair tokens (RewardTracker or sACID) to the contact. During the deposit process, the amount of pair tokens is calculated and is ensured to be locked in another contract. Acid token is used as reward token, being minted to VesterV2 contract when the `_claim()` function is called. Claiming also causes vACID and esACID tokens to be burnt.

VesterV2 contract is meant to be deployed via proxy.

Issues

C5-01 Centralization problem ● High ✓ Acknowledged

The owner can withdraw deposited funds by calling the `withdrawToken()` function.

```
    function withdrawToken(address _token, address _account, uint _amount) external
onlyOwner {
```

Recommendation

The owner's account must be secured, preferably by transferring the ownership to a contract with safety guards, or by renouncing ownership completely.

C5-02 Initialization problem ● Medium ✓ Resolved

The `lastVestingTime` is initialized by zero value and therefore `_getNextClaimableAmount()` wrongly calculates the claimable amount.

Recommendation

Initialize the contract with the current timestamp.

C5-03 Wrong imports ● Medium ✓ Resolved

Unknown import of BaseTokenUpgradeable.sol makes VesterV2 not compilable from the repo.

Recommendation

Include the BaseTokenUpgradeable contract in the repository.

C5-04 Upgradeability problem ● Medium ✓ Acknowledged

There are no storage gaps for the upgradeability of the proxy contract.

Recommendation

Any update must be thoroughly tested to not interfere with the existing storage layout.

C5-05 Lack of events ● Info ✓ Acknowledged

The functions `setVestDuration()`, `setPairMultiplier()`, and `setDisabled()` don't emit events, which complicates the off-chain tracking of important changes.

C6. RewardsDistributor

Overview

Linear distribution model contract. The token is minted for the RewardTracker contract when it calls the `distribute()` function.

Issues

C6-01 Gas optimizations ● Low ✓ Resolved

The variables `rewardToken` and `rewardTracker` should be declared as immutable.

C6-02 Lack of events ● Info ✓ Acknowledged

Lack of events in the governance function `updateLastDistributionTime()`.

C6-03 Typos ● Info ✓ Resolved

Typos reduce code readability. Typos in 'untill'.

C7. RewardTracker

Overview

An ERC-20 standard token inherited from the BaseToken contract. User deposits ACID to mint RewardTracker token (sACID). Rewards in form of esACID tokens are minted in the RewardDistributor contract, which also specifies the mint speed (esACID is minted linearly).

Users can use deposit with a lock to boost their rewards: locking effectively increases the deposited amount in rewards calculations.

Notifies external contracts for optional extra rewards.

Issues

C7-01 Centralization problem

● High ✓ Acknowledged

A malicious owner can set arbitrary addresses of distributor and extraRewards, which may break user interactions with the contract.

```
    function setRewardDistributor(IRewardDistributor _distributor) external onlyOwner {
        distributor = _distributor;
    }

    function addExtraReward(address _reward) external onlyOwner {
        require(_reward != address(0), "!reward setting");
        extraRewards.push(_reward);
    }
```

Recommendation

The owner's account must be secured, preferably by transferring the ownership to a contract with safety guards, or by renouncing ownership completely.

C7-02 Gas optimizations

● Low ✓ Resolved

1. The variables `acid` and `esAcid` should be declared as immutable.

2. Multiple reads from the storage of `extraRewards.length` in the `_claim()` function.

3. Unnecessary read of `boostedAmount[user]` in the `_updateBoostedAmount()` function.

C7-03 Lack of validation

● Low ✓ Acknowledged

There's no extra rewards length check. More than 256 could be added but in the `_updateExtraRewardStaked()` iteration the `uint8` index is used.

C7-04 Lack of events

● Info ✓ Acknowledged

Lack of events in governance functions `setRewardDistributor()`, `addExtraReward()`, and `updateBoostParamters()`.

C7-05 Typos

● Info ✓ Resolved

Typos reduce the code's readability. Typo in 'paramters'.

C8. StakingYieldPool

Overview

BasePool address updates users' staked amounts.

The manager role bearer sets the reward rate and duration.

Distributes reward tokens, which are deposited externally, the contract checks if it has enough rewards token when epy `notifyRewardAmount()` function is called.

StakingYieldPool contract is meant to be deployed via proxy.

Issues

C8-01 Upgradeability problem

● Medium ✓ Acknowledged

There are no storage gaps for the upgradeability of the proxy contract.

Recommendation

Any update must be thoroughly tested to not interfere with the existing storage layout.

C8-02 Gas optimizations ● Low ✓ Resolved

1. A possibly unnecessary read from the storage of `rewardPerTokenStored` in the `updateReward()` function.

2. No need to call `earned()` function in the `getReward()` since `updateReward()` syncs `rewards[_account]`.

C8-03 Unused parameter ● Low ✓ Resolved

The input parameter `newAmount` is not used in the `updateStaked()` function.

C8-04 Lack of events ● Info ✓ Acknowledged

Lack of events in the governance function `setManager()`.

C9. AcidBondV4

Overview

A fork of OlympusDAO BondDepositary. AcidBondV4 allows users to purchase ACID bonds with different markets. Terms and supported tokens are owner-controlled.

Issues

C9-01 Lack of events ● Low ✓ Acknowledged

Lack of events in governance functions setMinPrice(), setTuneEnaled(), setAdjustment(), setCapacity(), and setDebtDecayTerm().

C9-02 Gas optimizations ● Low ✓ Resolved

1. The Terms structure should be rearranged to be stored in 2 storage slots instead of 3.

2. The variables acid and treasury should be declared as immutable.

3. Multiple reads of the adjustments[_id] in the _decay() and nested functions.

4. Multiple reads of markets.length in the liveMarkets() function.

C9-03 Typos ● Info ✓ Resolved

Typos reduce the code's readability. Typos in 'decayedf', 'enaled'.

C9-04 Irrelevant and inconsistent comments ● Info ✓ Resolved

Comments remain from a deleted code from the fork on L374-378.

OHM token reference on L32 should be ACID.

Wrong comment on L147: max payout is not recalculated.

C10. BondHelper

Overview

A helper contract that interacts with external contracts. Some contracts are out of the scope of the current audit.

Issues

C10-01 Gas optimizations ● Low ✓ Resolved

The variables `weth`, `wstEth`, `rewardTracker`, `spNFT`, and `lpToken` should be declared as immutables.

5. Conclusion

2 high, 6 medium, 10 low severity issues were found during the audit. 2 medium, 6 low issues were resolved in the update.

The reviewed 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.

The audited contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

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

blog.hashex.org

linkedin

github

twitter

#HashEx
BLOCKCHAIN SECURITY