#HashEx
BLOCKCHAIN SECURITY

EVEDEX

smart contracts final audit report

August 2025

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 EVEDEX team to perform an audit of their smart contracts. The audit was conducted between 17/06/2025 and 20/06/2025.

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 0xc7dEfc896E1Dd9D98AEa402d928C78c0FD1dc36C (Multicall3), 0x77075c627e51145d54e4EDD54Afa169DA7ff8A17 (Treasury proxy) with implementation at 0xbe0a8FB9F5943f547d0A3DC06eE5e478C76Dda52 (TreasuryV1), 0x026968b5cED079ECCD6CC78f35a5Dfddc13F9Af8 (Vault proxy) with implementation at 0x0507389B4cbB043E6d0137B70c40Fc2BE483919B (VaultV2), 0x0a9591c64Fd9e8C1f9A81DB1B668a5f211b5735A (CashbackVault proxy) with implementation at 0x86dC076BC83044EEbb99a03015B9e130D72e6197 (CashbackVaultV1), 0xEB71A9c02c1F68A9D7004e74e17c7d62C3291Bf8 (GovernorMultisig), 0x21F69c75d757164137044b235663FeC8beC404E5 (Storage) in the Eventum mainnet and at 0x66208B370a58812533EC7aa6971B9702B1e58cC8 (EHMarket proxy) with implementation at 0xc9bf97Efb7E33699904e80aF7c3Ee730770440ed (EHMarketV2) in the Eventum testnet.

The code is also available in the @evedex-official/solidity-contracts GitHub repo after 192ba94 commit and in the @evedex-official/exchange-ehmarket repo after the 6a1b184 commit.

Update. The EVEDEX team has responded to this report. The updated code is available in the same repositories after the commit 0a05b0e and f2186e2. Cashback vault implementation was deployed to 0x6282f3821615A894012126DF8Bb78bBD9251B849. EHMarketV2 proxy

was deployed to 0x1DC14e4261eCd7747Cbf6D2C8538a73371405D76 and it's implementation to 0x80AF2ee95f5e40251f2C759a04908a21E3272038 in the Eventum Mainnet.

Update 2. An incremental audit of the EHMarket contract was conducted between 25/09/2025 and 27/09/2025.

The audited code was provided via EVEDEX private GitLab repository. SHA256 hashes of audited files:

`630d118e489c9c9522e729e9cc5a2cbfef21bcc17d34cb1b871f809b2e7d143d` EHMarketV3.sol.

2.1 Summary

Project name	EVEDEX
URL	https://evedex.com/
Platform	Eventum Network
Language	Solidity
Centralization level	High
Centralization risk	Low

2.2 Contracts

Name	Address
EHMarketV2	`0x1DC14e4261eCd7747Cbf6D2C8538a73371405D76`
VaultV2	`0x026968b5cED079ECCD6CC78f35a5Dfddc13F9Af8`

CashbackVaultV1	0x0a9591c64Fd9e8C1f9A81DB1B668a5f211b5735A
TreasuryV1	0x77075c627e51145d54e4EDD54Afa169DA7ff8A17
Storage	0x21F69c75d757164137044b235663FeC8beC404E5
GovernorMultisig	0xEB71A9c02c1F68A9D7004e74e17c7d62C3291Bf8
Multiownable	0xEB71A9c02c1F68A9D7004e74e17c7d62C3291Bf8
Multicall3	0xc7dEfc896E1Dd9D98AEa402d928C78c0FD1dc36C

3. Project centralization risks

The contracts Treasury, Vault, CashbackVault, and Storage are highly centralized. The owner can transfer out funds and update crucial parameters, that may break interaction between contracts of the project.

We recommend securing the ownership with a multisig contract.

EVEDEX team comment

EVEDEX is committed to a structured decentralization roadmap comprised of three phases. Presently, the platform operates in Phase 1: client assets are held exclusively in an autonomous smart-contract vault, and no exchange-controlled user hot wallets exist. Launching at this interim decentralization level enables us to deliver a competitive product to market rapidly while maintaining on-chain custody.

Phase 2 will introduce contracts that autonomously execute position liquidations and facilitate non-custodial withdrawals. Development on these modules is ongoing; preliminary code and architecture are publicly available in our corporate repository (https://github.com/evedex-official/exchange-contracts).

In Phase 3, administrative privileges over all core contracts will transition to DAO governance, completing the decentralization process and eliminating centralized control vectors. This phased approach ensures progressive risk reduction and transparent evolution toward full community stewardship.

CbbCR4b Governance risks

The accounts with DEFAULT_ADMIN_ROLE can withdraw all available funds.

The accounts with MATCHER_ROLE can withdraw funds within configured limits. Total amount is not limited, and funds can be drained over time.

The contract is designed to be upgradeable by the owner of the ProxyAdmin contract.

Recommendations

Recommendation

Secure all DEFAULT_ADMIN_ROLE accounts with a multisig contracts.

Establish off-chain monitoring to prevent siphoning funds by MATCHER_ROLE accounts.

CbcCR4a Governance risks

The owner of the Storage contract can authorize an arbitrary address as distributor for the VaultV2 contract.

The owner can withdraw all contract's balance, including already distributed yet not withdrawn funds.

The contract is designed to be upgradeable by the owner of the ProxyAdmin contract.

Recommendations

Recommendation

Secure the ownership with a multisig contract.

CbdCR46 Owner privileges

The owner can withdraw all available funds.

The owner of ProxyAdmin contract can upgrade implementation logic.

Recommendations

Recommendation

Secure the ownership with a multisig contract.

CbeCR49 Owner privileges

All the contract's methods can be called only by the owner.

The contract is designed to be upgradeable by the owner of the ProxyAdmin contract.

Recommendations

Recommendation

Secure the ownership with a multisig contract.

CbfCR47 Owner privileges

The owner is responsible for maintaining stored parameters.

Recommendations

Recommendation

Secure the ownership with a multisig contract.

Cc0CR48 Governance risks

Any `howManyOwnersDecide` number of owners can execute an arbitrary transaction, including ownership transfer.

Recommendations

Recommendation

Configure multisig appropriately to reduce risk of losing ownership. Increase number of independent owners.

4. Found issues

Low	10 (83%)
Info	2 (17%)

Cbb. EHMarketV2

ID	Severity	Title	Status
CbbId1	Low	Lack of integration testing	Acknowledged
CbbId2	Low	Magic constants usage	Acknowledged
CbbI70	Low	Lack of authorization model	Acknowledged
CbbI72	Low	Matcher role overrides admin role	Resolved
CbbI6f	Low	Gas optimizations	Partially fixed
CbbI71	Info	Initialization problem	Resolved

Cbd. CashbackVaultV1

ID	Severity	Title	Status
CbdI76	Low	Lack of domain separator in signature verification	Acknowledged

CbdI6e	Low	The signer address can't be updated	Resolved
CbdI75	Info	Misleading function name withdrawCrumbs	Acknowledged

Cbf. Storage

ID	Severity	Title	Status
CbfI77	Low	Lack of zero value checks	Acknowledged

Cc2. Multicall3

ID	Severity	Title	Status
Cc2I73	Low	tryBlockAndAggregate always returns a zero block hash	Acknowledged
Cc2I74	Low	Misleading block information	Acknowledged

5. Contracts

Cbb. EHMarketV2

Overview

A token holding contract that allows users to deposit fixed ERC20 tokens. Withdrawing is allowed only for addresses with the MATCHER_ROLE or DEFAULT_ADMIN_ROLE roles. Withdraw limits can be configured for individual users and/or for total withdrawn amount for given periods. Withdraw limits are applied only for MATCHER_ROLE, but not for DEFAULT_ADMIN_ROLE.

Issues

CbbId1 Lack of integration testing

Low Acknowledged

The EHMarket is deployed as an upgradeable proxy and the EHMarketV3 is designed to be a proxy implementation. The contract itself is functionally tested, but there's no tests for upgrading.

We recommend add integration testing to ensure seamless upgrade.

CbbId2 Magic constants usage

Low Acknowledged

Request statuses are coded by using `uint8` and encoding scheme is documented by inline comments.

Solidity supports enums for better readability. Another solution to increase code readability is to use named constants for statuses.

CbbI70 Lack of authorization model

Low Acknowledged

The `migrateToV2()` function is open for public use.

CbbI72 Matcher role overrides admin role

Low Resolved

Withdrawing process is restricted to accounts with the MATCHER_ROLE and DEFAULT_ADMIN_ROLE. However, if the caller possesses both these roles, the matcher's limits are applied despite the default admin should be exempted from checks.

```
function withdrawAsset(address from, uint256 amount, uint256 requestId) external {
  bool isMatcher = hasRole(MATCHER_ROLE, _msgSender());
  require(isMatcher || hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), UnauthorizedSigner());
  if (isMatcher) {
    _checkWithdrawLimits(amount, from);
    ...
  }
  ...
}
```

CbbI6f Gas optimizations

Low Partially fixed

1. Multiple reads from storage in the `getMaxWithdrawAmount()` function: `withdrawLimits.length` variable. **Update**: fixed.

2. The `getMaxWithdrawAmount()` function calls `getTotalWithdraw()` and `getUserTotalWithdraw()` functions in a cycle to calculate withdrawn amounts by hours. The withdraw limits don't overlap and sorted in ascending order, so calculations of `totalWithdrawn` and `userWithdrawn` amounts can be optimized to exclude multiple reading of the same data from storage. **Update** : fixed.

3. Multiple reads from storage in the `_checkWithdrawLimits()` function: `withdrawLimits.length` variable. **Update**: fixed.

4. Multiple reads from storage in the `_setWithdrawLimits()` function: `withdrawLimits`, `withdrawLimits.length` variables.

5. Array sorting in the `_setWithdrawLimits()` function is ineffective, it should either use in-memory sorting or be omitted completely in favor of requiring the input data to be sorted.
Update: fixed.

CbbI71 Initialization problem

Info Resolved

The `initialize()` is unusable if the contract is used as an upgrade. Consider using `Initializable.reinitializer` modifier to replace the `migrateToV2()` function and the `Initializable._getInitializedVersion()` to replace `migratedToV2` getter.

Cbc. VaultV2

Overview

A token holder contract that supports token distribution to arbitrary receivers. The `distribute()` function is guarded by `Storage.getBool(keccak256(abi.encodePacked("EH:PartnerVault:Distributor:", msg.sender)))`.

Cbd. CashbackVaultV1

Overview

A vault contract that allows withdrawing fixed ERC20 tokens with a signed permission by back-end signer address.

Issues

CbdI76 Lack of domain separator in signature verification

Low Acknowledged

The `withdraw()` function in the CashbackVaultV1 contract uses a signature verification mechanism based on `keccak256(abi.encodePacked(...))` combined with `toEthSignedMessageHash`.

```
  function withdraw(
    address recipient,
    uint80 requestId,
    uint256 amount,
    bytes memory signature
  ) external whenNotPaused {
    bytes32 signedMessage = keccak256(abi.encodePacked(requestId, recipient, amount));
    if (signedMessage.toEthSignedMessageHash().recover(signature) != signer) {
      revert CashbackVaultV1InvalidWithdrawSignature();
    }
    ...
}
```

The signed hash is not tied to the specific context of this contract or the blockchain it's deployed on. This lack of a domain separator creates opportunities for replay attacks.

CbdI6e The signer address can't be updated

Low Resolved

The signer address is used to authorize user's withdrawals by providing signature that includes recipient address and withdrawing amount. However, the signer can't be updated without full upgrade of the contract.

```
    /// @dev Address of signer payload for withdraw method.
    address public signer;

    /// @notice Withdraw tokens.
    /// @param recipient Tokens recipient address.
```

```
    /// @param requestId Withdraw request ID.
    /// @param amount Withdrawal token amount.
    function withdraw(
        address recipient,
        uint80 requestId,
        uint256 amount,
        bytes memory signature
    ) external whenNotPaused {
        bytes32 signedMessage = keccak256(
            abi.encodePacked(requestId, recipient, amount)
        );
        if (
            signedMessage.toEthSignedMessageHash().recover(signature) != signer
        ) {
            revert CashbackVaultV1InvalidWithdrawSignature();
        }
        if (request[requestId]) {
            revert CashbackVaultV1WithdrawAlreadyCompleted();
        }

        request[requestId] = true;
        IERC20(token).safeTransfer(recipient, amount);
        emit CashbackWithdraw(recipient, requestId, amount);
    }
```

Cbdl75 Misleading function name withdrawCrumbs

● Info ✓ Acknowledged

The `withdrawCrumbs()` function name implies that this function is for withdrawing small, leftover amounts of tokens ("dust" or "crumbs") that might be stuck in the contract.

However, its implementation is an unrestricted onlyOwner withdrawal function that can transfer any amount of the contract's token balance to any specified address.

Cbe. TreasuryV1

Overview

A token holder contract with a single owner.

Cbf. Storage

Overview

A contract to store project config. Supports `uint`, `int`, `bool`, `address`, `bytes`, and `string` values indexed by `bytes32` keys.

Issues

Cbf|77 Lack of zero value checks

Low Acknowledged

The contract provides separate methods for deleting data in mappings by key. However, the setter functions do not check whether the provided value is zero. This could lead to unintended data deletion if a zero value is mistakenly passed to a setter.We recommend adding checks to prevent zero values from being set, in order to avoid accidental data deletion.

Cc0. GovernorMultisig

Overview

A multisig contract based on Multiownable contract. Supports batched arbitrary calls guarded with threshold decision of owners.

Cc1. Multiownable

Overview

A fork of Multiownable contract by BitClave, available in @bitclave/Multiownable repo.
Supports flexible multisig operations with or without threshold.

Cc2. Multicall3

Overview

A fork of Matt Solomon's Multicall3 available in @mds1/multicall3 repo.

Issues

Cc2I73 tryBlockAndAggregate always returns a zero block hash

Low Acknowledged

The `tryBlockAndAggregate()` function calculates blockHash using blockhash(block.number).
Per the EVM specification, `blockhash()` of the current block number always returns 0x0.

Cc2I74 Misleading block information

Low Acknowledged

The contract extensively uses EVM global variables (`block.number`, `blockhash`, `block.difficulty`, `block.coinbase`, `block.gaslimit`) that have different semantics on Eventum network compared to Ethereum L1.

- `getBlockNumber()` returns the L1 block number, not the L2 block number.
- `getCurrentBlockDifficulty()` returns a constant 1.
- `getCurrentBlockCoinbase()` returns a constant sequencer address or the address of the delayed message's poster.
- `getCurrentBlockGasLimit()` returns an artificially large constant.

6. Conclusion

10 low severity issues were found during the audit. 2 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter. 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. 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

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY