#HashEx
BLOCKCHAIN SECURITY

Synbo Vault

smart contracts
final audit report

February 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 Synbo team to perform an audit of their smart contract. The audit was conducted between 11/02/2025 and 14/02/2025.

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 SHA-1 of the audited files are:`33331fb33a15c1c4d09381d2cbab2ed467571e05` TreasuryVault.sol

`ad44142b4d5fe4fbcea7fb2bd0bf33dba1ec3ca8` VaultRequestManager.sol

`08e4c8992861b7f592bd57e57eb53c2a4eb6b842` ITreasuryVault.sol

`0fd4aad8060f098ed51d1df71859e5fe425a5779` IERC20MintableBurnable.sol

`bde269cd203651e8915803833e0c7494abeba034` SafeMath.sol

`76251654e451de41ff650c620daf82451877f27f` AlphaToken.sol

2.1 Summary

Project name	Synbo Vault
URL	https://synbo.org
Platform	Binance Smart Chain
Language	Solidity
Centralization level	🟥 High
Centralization risk	🟥 High

2.2 Contracts

Name	Address
TreasuryVault	
VaultRequestManager	
AlphaToken	
SafeMath	
Interfaces and imports	

3. Project centralization risks

C8eCR39 Owner privileges

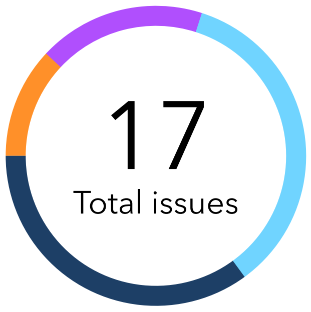
The owner can drain all available funds and make all withdrawals unusable.

The owner can set the `foundationAddress` to a one that doesn't accept native transfers blocking all or selectively filtering withdrawals.

The owner can change addresses of ERC20 tokens to be minted and burned during deposits and withdrawals.

The owner menages the `COO_ROLE` role that allows to update token rates and fees without any safety limits.

4. Found issues



●	High	2 (12%)
●	Medium	3 (18%)
●	Low	6 (35%)
●	Info	6 (35%)

C8e. TreasuryVault

ID	Severity	Title	Status
C8eId4	● High	Token addresses can be changed	Open
C8eId8	● High	Borrow requests authorization	Open
C8eId5	● Medium	Lack of input validation	Open
C8eIeb	● Medium	Underflow error	Open
C8eId2	● Low	Lack of events	Open
C8eId1	● Low	Gas optimizations	Open
C8eId7	● Info	Function totalSupply not including borrows	Open
C8eId9	● Info	Unnecessary declaration of experimental abiencoder	Open
C8eId3	● Info	Lack of documentation (NatSpec)	Open
C8eId6	● Info	Typographical error	Open

C8eld0	● Info	Inconsistent comment	Open

C8f. VaultRequestManager

ID	Severity	Title	Status
C8flce	● Medium	Authorization with tx.origin	Open
C8flcc	● Low	Possible gas limit problem	Open
C8flcb	● Low	Gas optimizations	Open
C8flcd	● Low	User history not cleared	Open
C8flcf	● Info	Typographical error	Open

C91. SafeMath

ID	Severity	Title	Status
C91lca	● Low	Incorrect compiler version	Open

5. Contracts

C8e. TreasuryVault

Overview

A funding pool contract that allows users to deposit native EVM currency to ERC20 tokens of 2 types with fixed rate. Those tokens can be burned to withdraw EVM currency. Additionally, addresses with a special role, managed by the contract owner, can request and withdraw stored funds with fixed delay.

Issues

C8eId4 Token addresses can be changed ● High ⓘ Open

The incoming deposits of native currency mint 1 or 2 ERC20 tokens with fixed rate. Those minted tokens later can be burned to redeem stored funds. The owner can update addresses of the ERC20 tokens to block deposits, withdrawals or to drain all stored funds.

```
function setPistionToken(address _pistionToken) public onlyRole(DEFAULT_ADMIN_ROLE) {
    pistionToken = _pistionToken;
}

function setYieldToken(address _yieldToken) public onlyRole(DEFAULT_ADMIN_ROLE) {
    yieldToken = _yieldToken;
}
```

Recommendation

Remove the owner's ability to change token addresses.

C8eld8 Borrow requests authorization ● High ⓘ Open

The `requestWithdrawal()` function allows addresses with the `VAULTREQUEST_ROLE` to register withdrawal request with the pending period, during which the special wards with the `REQUEST_REJ_ROLE` can cancel the request.

The problem is that the request is authorized via transaction signer `tx.origin`, but the actual receiver of the funds is the `msg.sender` of the `borrow()` function call.

The `REQUEST_REJ_ROLE` wards may be unable to reject a request basing only on the `tx.origin`, but not the actual receiver of the funds.

```
    /**
     * @notice Project investment logic
     * - Step 1: Create a withdrawal request.
     * - Step 2: Execute withdrawal.
     */
    function requestWithdrawal(uint256 _chequeId, uint256 _round, uint256 _amount)
external onlyRole(VAULTREQUEST_ROLE) {
        _requestWithdrawal(_chequeId, _round, _amount);
    }

    function rejectWithdrawal(uint256 _requestId) external onlyRole(REQUEST_REJ_ROLE) {
        _rejectWithdrawal(_requestId);
    }

    function borrow(uint256 _requestId) public onlyRole(VAULTREQUEST_ROLE) whenNotPaused {
        require(requestByUser[tx.origin] != 0, "Existing loan active");
        require(_requestId == getActiveRequestID(tx.origin),"Invaild permissions");

        VaultRequest memory request = VaultRequesting[_requestId];
        require(request.amount <= balance(), "Insufficient funds in vault");

        _executeWithdrawalbyId(_requestId);

        totalDebt = totalDebt.add(request.amount);
        emit Borrow(tx.origin, request.amount);
    }
```

Recommendation

Consider using address as a parameter of the request.

C8eld5 Lack of input validation ● Medium Open

The `_type` parameter of the `deposit()` function is interpreted as 2 regardless its specified value in any case `except _type == 1`.

```
/**
 * @notice User-side operations: deposit and withdraw.
 */
function deposit(uint8 _type) public payable whenNotPaused {
    ...

    if (_type == 1) {
        ...
    } else {
        ...
    }
}
```

Recommendation

Use enum for type or add input validation.

C8eleb Underflow error ● Medium Open

The `pendingInvestment()` function returns difference between total deposited funds and the sum of total withdrawn amount and current balance. Since the contract can receive native currency directly (`receive()` function), the `pendingInvestment()` may result in underflow error.

```
function pendingInvestment() public view returns (uint256 pending) {
    pending = totalDeposit - totalWithdraw - balance();
    return pending;
}
```

Recommendation

Return 0 in case of underflow and document function's logic with a NatSpec description.

C8eId2 Lack of events ● Low ⓘ Open

The functions `setFoundationAddress()`, `setPistionToken()`, `setYieldToken()`, `setSellFee()`, `setBuyFee()`, `setBuyFee2()`, `setBaseRate()` change important variables in the contract storage, but no events are emitted.

We recommend adding events to these functions to make it easier to track their changes offline.

C8eId1 Gas optimizations ● Low ⓘ Open

1. Multiple reads from storage in the `deposit()` function: `pistionToken`, `yieldToken`, `foundationAddress`, `buyFee`, `baseRate`, `buyFee2` variables.

2. Unnecessary calculations in the `deposit()` function: `feeAmount` and `inAmount` can be calculated with `baseRate` multiplication, `totalDeposit` increment and `Deposit` event parameter are always equal to `msg.value`.

3. Multiple reads from storage in the `withdraw()` function: `baseRate`, `sellFee`, `foundationAddress` variables.

4. Repetitive checks in the `borrow()` function: `requestByUser[tx.origin] != 0` is checked twice.

5. Multiple reads from storage in the `borrow()` function: `requestByUser[tx.origin]`, `VaultRequesting[_requestId]` variables.

C8eId7 Function totalSupply not including borrows ● Info ⓘ Open

The `totalSupply()` function doesn't account total debt and total repayment. We recommend include a NatSpec description to document the function's returning values.

C8eId9 Unnecessary declaration of experimental abiencoder ● Info ⓘ Open

Since Solidity 0.8.0, the ABIEncoderV2 has been fully integrated and no longer considered experimental. The line

```
pragma experimental ABIEncoderV2;
```

should be removed from the contract.

C8eId3 Lack of documentation (NatSpec) ● Info ⓘ Open

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

C8eId6 Typographical error ● Info ⓘ Open

The smart contract code contains a typographical errors in `Invaild`, `pistion,` which should be correctly spelled as `Invalid`, `piston` to align with standard English grammar conventions.

C8eId0 Inconsistent comment ● Info ⓘ Open

Roles' description is incomplete and misleading. `ADAPTER_ROLE` is missing, `VAULTREPAYMENT_ROLE` is commented out, `REQUEST_REJ_ROLE` and are not documented.

```
    /**
     * @notice Role management and definitions
     * - DEFAULT_ADMIN_ROLE: Default super administrator. Can be multiple accounts and
transferable to multisig.
     * - COO_ROLE: Responsible for parameter settings.
     * - ADAPTER_ROLE: Withdrawal permission management for business contracts (divided
into VAULTBORROW_ROLE and VAULTREPAYMENT_ROLE).
     */
    bytes32 public constant COO_ROLE = keccak256("COO_ROLE");
    bytes32 public constant VAULTREQUEST_ROLE = keccak256("VAULTREQUEST_ROLE");
    // bytes32 public constant VAULTREPAYMENT_ROLE = keccak256("VAULTREPAYMENT_ROLE");
    bytes32 public constant REQUEST_REJ_ROLE = keccak256("REQUEST_REJ_ROLE");
```

C8f. VaultRequestManager

Overview

An abstract contract to register, execute and cancel withdrawal requests. Successful withdrawals are paid in native EVM currency. All requests share the same time constants for the pending period and the total lifetime period, after which they can be marked as expired.

Issues

C8fIce Authorization with tx.origin ● Medium Open

In the `_executeWithdrawalbyId()` function both `tx.origin` and `msg.sender` are mixed as withdraw requester and withdraw receiver. Using transaction signer address can significantly complicate support of smart account such as ERC4337 or Gnosis Safe.

```
    /**
     * @dev Executes a withdrawal request.
     * @param _requestId The ID of the request to execute.
     */
    function _executeWithdrawalbyId(uint256 _requestId) internal nonReentrant {
        VaultRequest storage request = VaultRequesting[_requestId];
        require(request.timestamp != 0, "Request not found");
        require(request.requester == tx.origin, "Not your request");
        require(!request.rejected, "Request was rejected");
        require(block.timestamp >= request.timestamp + LOCK_PERIOD, "Lock period not
over");
        require(block.timestamp <= request.expirationTime, "Request expired");
        require(!request.executeWithdrawaled, "Already executed");

        uint256 amount = request.amount;
        request.executeWithdrawaled = true;
        requestByUser[tx.origin] = 0;

        (bool success, ) = payable(msg.sender).call{value: amount}("");
        require(success, "Transfer failed");

        emit WithdrawalCompleted(_requestId, tx.origin, amount);
    }
```

Recommendation

Consider not using the `tx.origin` and receive user address in parameters.

C8flcc Possible gas limit problem ● Low ⓘ Open

The `userRequestHistory()` and `getUserRequestHistory()` getter functions return full array of user's request IDs, which can be an arbitrary length long.

```
    mapping(address => uint256[]) public userRequestHistory;

    function _requestWithdrawal(
        uint256 _chequeId,
        uint256 _round,
        uint256 _amount
    ) internal {
        ...
        userRequestHistory[tx.origin].push(newRequestId);
    }
```

Recommendation

Modify the getUserRequestHistory() function to return slice of array.

C8flcb Gas optimizations

● Low ⓘ Open

1. The `VaultRequest` struct can be repacked to 7 storage slots by combining address with booleans.

2. Multiple reads from storage in the `_requestWithdrawal()` function: `requestIdCounter` variable.

3. Multiple reads from storage in the `_expireRequest()` function: `request.requester` variable.

4. Multiple reads from storage in the `_executeWithdrawalbyId()` function: `request.timestamp` variable.

C8flcd User history not cleared

● Low ⓘ Open

The `_expireRequest()` function removes the VaultRequesting entry but remains the `_requestId` in the user's request history.

```
/**
 * @dev Marks a withdrawal request as expired if it has exceeded the expiration time.
 * @param _requestId The ID of the request to expire.
 */
function _expireRequest(uint256 _requestId) internal {
    VaultRequest storage request = VaultRequesting[_requestId];
    require(request.timestamp != 0, "Request not found");
    require(block.timestamp > request.expirationTime, "Request not expired");

    requestByUser[request.requester] = 0;
    delete VaultRequesting[_requestId];

    emit WithdrawalExpired(_requestId, request.requester);
}
```

Recommendation

Use an OpenZeppelin's EnumerableSet for user history to clear expired entries with O(1) or add expiration field into `VaultRequest` struct to exclude deleting.

C8flcf **Typographical error** ● Info ⓘ Open

The smart contract code contains a typographical error in `Withdrawaled`, which should be correctly spelled as `Withdrawn` to align with standard English grammar conventions.

C90. **AlphaToken**

Overview

An ERC20 standard token contract. The initial supply is fixed, i.e. there's no active minting functionality after deployment. The token contract is inherited from OpenZeppelin's ERC20 implementation, which is considered the best practice.

No issues were found.

C91. **SafeMath**

Overview

OpenZeppelin's library for arithmetic operations with added overflow checks.

Issues

C91lca **Incorrect compiler version** ● Low ⓘ Open

The SafeMath library has incorrect compiler version. The post-0.8 versions of Solidity don't need explicit checks for mathematical operations. All functions of the SafeMath should be performed in unchecked blocks.

```
OpenZeppelin release-v3.4 version:
  pragma solidity >=0.6.0 <0.8.0;

Audited version:
  pragma solidity >=0.8.20;
```

Recommendation

Remove the SafeMath completely.

C92. Interfaces and imports

Overview

OpenZeppelin contracts are imported from a local copy anr correspond to the v5.2 release.

No issues were found.

6. Conclusion

2 high, 3 medium, 6 low severity issues were found during the audit. No 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