hashlock.

2024

# Security Audit

## Balanced (Token / dApp)

25/06/2024 info@hashlock.com.au Hashlock Pty Ltd

# Table of Contents

## CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE THAT COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR THE USE OF THE CLIENT.

# Executive Summary

The Balanced team partnered with Hashlock to conduct a security audit of their Balanced protocol smart contracts. Hashlock manually and proactively reviewed the code to ensure the project's team and community that the deployed contracts were secure.

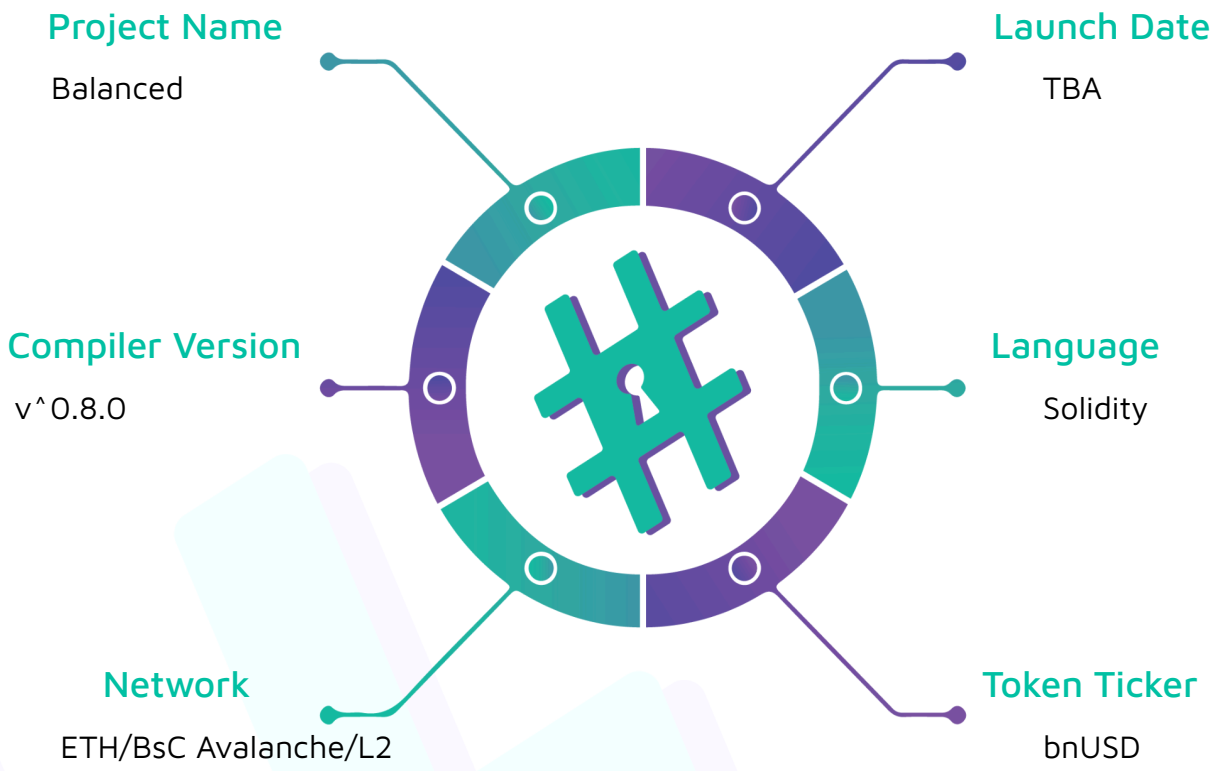
# Project Context

The Balanced protocol is a decentralized platform that allows users to mint a stablecoin called Balanced Dollar (bnUSD) by depositing cryptocurrency as collateral. bnUSD is pegged to the US Dollar and its value is maintained through over-collateralization, liquidation mechanisms, and the Stability Fund, which allows 1:1 swaps with other stablecoins.

Users can also earn rewards by depositing bnUSD into the Savings Rate, providing liquidity, and taking advantage of arbitrage opportunities. The protocol supports various collateral types and operates on multiple blockchains.

**Project Name**: Balanced
**Compiler Version**: ^0.8.0
**Website**: https://balanced.network/
**Logo**:

**Visualised Context:**



**Project Visuals:**


# Audit scope

We at Hashlock audited the solidity code within the Balanced project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

| Description | Balanced Protocol Smart Contracts |
|---|---|
| Platform | Ethereum / Solidity |
| Audit Date | June, 2024 |
| Contract 1 | asset-manager/AssetManager.sol |
| Contract 1 MD5 Hash | 8aa8b927e4b91309581e1ba8392f91e7 |
| Contract 2 | asset-manager/Messages.sol |
| Contract 2 MD5 Hash | 03edce15fbd8a73089aeb673f4e86fa4 |
| Contract 3 | asset-manager/RLPDecodeStruct.sol |
| Contract 3 MD5 Hash | 80c343cc971a0e652f18c4159394e6d6 |
| Contract 4 | asset-manager/RLPEncodeStruct.sol |
| Contract 4 MD5 Hash | d157dbc9f3bfb5b0dafed131b90e3d41 |
| Contract 5 | bnusd/BalancedDollar.sol |
| Contract 5 MD5 Hash | 1873bc5d5567d18c1ffc3933f129b3fe |
| Contract 6 | bnusd/Messages.sol |
| Contract 6 MD5 Hash | ad973b847b007316f28b29cbbc734322 |
| Contract 7 | bnusd/RLPDecodeStruct.sol |
| Contract 7 MD5 Hash | 914a7424fd6968e07c08f4c76e58a267 |
| Contract 8 | bnusd/RLPEncodeStruct.sol |
| Contract 8 MD5 Hash | 073272a5b3e8d1b291be725ddbdab29b |

| | |
|---|---|
| **Contract 9** | oracle-proxy/OracleProxy.sol |
| **Contract 9 MD5 Hash** | eed10bda95c99c41aa2bdb86222fa5bc |
| **Contract 10** | oracle-proxy/Messages.sol |
| **Contract 10 MD5 Hash** | 078a9528a4721368e36c4ff4159b9089 |
| **Contract 11** | oracle-proxy/RLPEncodeStruct.sol |
| **Contract 11 MD5 Hash** | 26c1d17ffa522a03767b54d2afa2b89c |
| **Contract 12** | xcall-manager/XCallManager.sol |
| **Contract 12 MD5 Hash** | c4c342ba0813cd091cde23fb7bc92e0e |
| **Contract 13** | xcall-manager/RLPEncodeStruct.sol |
| **Contract 13 MD5 Hash** | accb35454dfa7deedbeddab7977a5ff7 |
| **Contract 14** | xcall-manager/RLPDecodeStruct.sol |
| **Contract 14 MD5 Hash** | 2a230f1839a461e4ce66d990dd9d3915 |
| **Contract 15** | xcall-manager/Messages.sol |
| **Contract 15 MD5 Hash** | 03be2f09ac5510156a1adaef23d51f03 |
| **Contract 16** | library/btp2/utils/Integers.sol |
| **Contract 16 MD5 Hash** | e3d539bdfbc1bc771779239375c3885b |
| **Contract 17** | library/btp2/utils/NetworkAddress.sol |
| **Contract 17 MD5 Hash** | b3d02bcf18199307b399901bbf4f6387 |
| **Contract 18** | library/btp2/utils/ParseAddress.sol |
| **Contract 18 MD5 Hash** | 717833bbbc0f961eda0f1b00dd1dec8c |
| **Contract 19** | library/btp2/utils/RLPDecode.sol |
| **Contract 19 MD5 Hash** | 8a237d675ba6b1b4a7ff47dfcb229869 |
| **Contract 20** | library/btp2/utils/RLPEncode.sol |
| **Contract 20 MD5 Hash** | 3d288700d76882ab990800af30932805 |
| **Contract 21** | library/btp2/utils/Strings.sol |
| **Contract 21 MD5 Hash** | 15688932876357610555acf8dafa69d9 |

| | |
|---|---|
| **Contract 22** | lib/interfaces/IXCallManager.sol |
| **Contract 22 MD5 Hash** | 28334801583d1670e90b6ad145e1d7b9 |
| **Contract 23** | evm/contracts/xcall/CallService.sol |
| **Contract 23 MD5 Hash** | 9dad6a1ef15f815643ec96ccd0a0316a |

# Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts and ICON's XCall solidity libraries. We have identified some significant vulnerabilities that were addressed prior to launch.


*The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on-chain monitoring technology.*

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section.

All vulnerabilities we have identified have yet to be resolved or acknowledged.

**Hashlock found:**

2 High-severity vulnerabilities

1 Medium-severity vulnerabilities

15 Low-severity vulnerabilities

0 Gas Optimisations

**Caution:** *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

# Intended Smart Contract Behaviours

| Claimed Behaviour | Actual Behaviour |
|---|---|
| **AssetManager.sol**<br>- Allows users to:<br>  - Deposit ERC20 and native ETH tokens<br>  - Handles the cross-bridge token deposit/withdraw logic between the Ethereum <-> ICON chains<br>- Allows admins to:<br>  - Configure and reset the withdraw rate limit.<br>  - Upgrade the contract. | **Contract achieves this functionality.** |
| **BalancedDollar.sol**<br>- ERC20 contract used to:<br>  - Handles the transfer of bnUSD between other chains.<br>  - Burns bnUSD on transfers to another chain.<br>  - Mints new bnUSD on transfers to current chain.<br>- Allow admins to:<br>  - Upgrade the contract. | **Contract achieves this functionality.** |
| **XCallManager.sol**<br>- A periphery contract used to:<br>  - Verify protocols used in the cross-chain bridging.<br>- Allows admins to:<br>  - Update the protocols. | **Contract achieves this functionality.** |

| | |
|---|---|
| - Upgrade the contract.<br>- Configure available sources and destinations. | |
| **OracleProxy.sol**<br>- A price oracle contract used to:<br>  - Calculate the price of the assets stored in the ERC4626 vaults.<br>  - Send cross-chain price updates.<br>- Allows admins to:<br>  - Add credit vaults.<br>  - Remove credit vaults.<br>  - Upgrade the contract. | **Contract achieves this functionality.** |
| **CallService.sol**<br>- Contract to make calls between different blockchain networks<br>- Handles calls to and from adapters | **Contract achieves this functionality.** |

# Code Quality

This Audit scope involves the smart contracts of the Balanced project, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring was required.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

# Audit Resources

We were given the Balanced project's smart contract code in the form of GitHub access.

As mentioned above, code parts are well-commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments help us understand the overall architecture of the protocol.

# Dependencies

Per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry-standard open-source projects.

Apart from libraries, its functions are used in external smart contract calls.

# Severity Definitions

| Significance | Description |
|---|---|
| **High** | High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community. |
| **Medium** | Medium-level difficulties should be solved before deployment, but won't result in loss of funds. |
| **Low** | Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future. |
| **Gas** | Gas Optimisations, issues, and inefficiencies |

# Audit Findings

## High

### [H-01] XCallManager.sol#verifyProtocolsUnordered - Array comparison does not work as implemented

**Description**

`verifyProtocolsUnordered` function deals with an unordered comparison of arrays. However, the implemented function will consider these two arrays as equivalent:

['alice','alice','alice']

['alice', 'bob', 'charlie']

**Vulnerability Details**

```
… REDACTED FOR BREVITY …

for (uint i = 0; i < array1.length; i++) {
      for (uint j = 0; j < array2.length; j++) {
      if (array1[i].compareTo(array2[j])) {
            break;
      } else {
            if (j == array2.length - 1) return false;
            continue;
      }
      }
}
```

This function is satisfied by each item occuring in the other array.

**Impact**

This code is also relied on by external calls into this contract, i.e. through `handleCallMessage()`. As demonstrated, this code will not correctly work and will thus break assumptions made by external callers relying on this functionality.

**Recommendation**

Either make sure that arrays do not contain duplicates, or count the number of occurrences per item and then check that the occurrences counts are equal. Meaning if Alice exists twice in array A it should also exist twice in array B.

**Status**

Resolved

### [H-02] Balanced Contracts - Token transfers may not actually execute

**Description**

Tokens are transferred here but the returned boolean indicating success is ignored. Therefore this function may proceed even though the transfer has actually failed.

**Vulnerability Details**

```
IERC20(token).transferFrom(msg.sender, address(this), amount);
IERC20(token).transfer(to, amount);
```

These functions can return bool to indicate the success of the transfer. This must not be ignored.

**Impact**

The contract assumes that tokens have been transferred when they may not have been. This will lead to a loss of funds for ERC20 implementations returning false to indicate transfer failure.

**Recommendation**

Use OpenZeppelins SafeERC20 functionality i.e. safeTransferFrom() and safeTransfer(), these will correctly deal with returned values by throwing an error.

**Status**

Resolved.

# Medium

## [M-01] AssetManager.sol#_deposit - Deposit function is vulnerable to fee-on-transfer accounting-related issues

### Description

`_deposit` function transfers funds from the caller to the receiver via `safeTransferFrom()`, but do not ensure that the actual number of tokens received is the same as the input amount to the transfer.

### Vulnerability Details

Here the _deposit function do not ensure that the actual number of tokens received is the same as the input amount to the transfer.

```
function _deposit(
        address token,
        uint amount,
        string memory to,
        bytes memory data
    ) internal {
        require(amount >= 0, "Amount less than minimum amount");
        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
        sendDepositMessage(token, amount, to, data, msg.value);
    }
```

### Impact

If the token is a fee-on-transfer token, the balance after the transfer will be smaller than expected, leading to accounting issues.

### Recommendation

One way to address this problem is to measure the balance before and after the transfer, and use the difference as the amount, rather than the stated amount. For example:

```
function _deposit(
        address token,
        uint amount,
        string memory to,
        bytes memory data
    ) internal {
        require(amount >= 0, "Amount less than minimum amount");
        uint256 balanceBefore = IERC20(token).balanceOf(address(this))
        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
        uint256 balanceAfter = IERC20(token).balanceOf(address(this));
        uint256 actualReceived = balanceAfter - balanceBefore;
        sendDepositMessage(token, actualReceived, to, data, msg.value);
    }
```

### Status

Resolved.

# Low

## [L-01] Multiple Balanced Contracts - Missing zero address checks

### Description

There are a number of missing zero checks in this codebase:

- /src/asset-manager/AssetManager.sol
- /src/oracle-proxy/OracleProxy.sol
- /src/xcall-manager/XCallManager.sol
- /src/bnusd/BalancedDollar.sol

### Vulnerability Details

```
function initialize(
      address _xCall,
      string memory _iconOracle,
      address _xCallManager
) public initializer {
      xCall = _xCall;
      iconOracle = _iconOracle;
      xCallManager = _xCallManager;
      __Ownable_init(msg.sender);
}
```

None of the initializers have zero address checks on any address parameter.

```
function configure(
      address _xCall,
      string memory _iconOracle,
      address _xCallManager
) external onlyOwner {
      xCall = _xCall;
      iconOracle = _iconOracle;
      xCallManager = _xCallManager;
```

```
}
function setAdmin(address _admin) external onlyAdmin() {
      admin = _admin;
}
```

The admin can be set to the zero address.

**Impact**

Zero checks are important in order to avoid mistakes. Zero values could indicate default values and calls might happen due to mistakes. Setting the admin to zero will lead to a loss of control over the contract.

**Recommendation**

Add tests against the zero address when accepting addresses from user input.

**Status**

Resolved.

### [L-02] Multiple Balanced Contracts - Lack of events for important state updates

**Description**

There are no events being emitted for important state updates.

**Vulnerability Details**

Relevant functions to consider:

OracleProxy.configure(), AssetManager.configureRateLimit(), AssetManager.resetLimit(), XCallManager.proposeRemoval(),XCallManager.whitelistAction(), XCallManager.removeAction(), XCallManager.setAdmin(), XCallManager.setProtocols().

**Impact**

This makes it harder for users to detect changes in the contract state. Events are also relevant for 3rd party developers who might want to build external tools.

**Recommendation**

Add some events to support 3rd party developers.

**Status**

Resolved.

### [L-03] Multiple Balanced Contracts - Return value not checked

**Description**

There are 3 instances of calls being done with ICallService(xCall).sendCallMessage{}(). This function returns a value which is ignored:

- /src/asset-manager/AssetManager.sol:sendCallMessage(..)
- /src/oracle-proxy/OracleProxy.sol:sendCallMessage(..)
- /src/bnusd/BalancedDollar.sol:sendCallMessage(..)

**Vulnerability Details**

```
ICallService(xCall).sendCallMessage{value: msg.value}(
      iconBnUSD,
      xcallMessage.encodeCrossTransfer(),
      rollback.encodeCrossTransferRevert(),
      protocols.sources,
      protocols.destinations
);
```

The function called returns uint256:

```
function sendCallMessage(
```

```
        string memory _to,
        bytes memory _data,
        bytes memory _rollback,
        string[] memory sources,
        string[] memory destinations
    ) external payable returns (
        uint256
    );
```

**Impact**

Since this is an external function, it is not clear whether this function can fail and attempt to indicate that using the returned value i.e. by returning 0.

**Recommendation**

Verify this is implemented correctly. If the return value is supposed to be ignored, state this in a comment and explain why the return value should be ignored.

**Status**

Acknowledged.

### [L-04] OracleProxy.sol#fetchCreditVaultPrice - Decimal conversion function does not work for assets with more than 18 decimals

**Description**

The function `fetchCreditVaultPrice` implements a conversion logic for assets with different numbers of decimals. This function however can not work for assets with more than 18 decimals as the calculation will underflow. At the same time nothing prevents assets with more decimals from being used.

**Vulnerability Details**

```
rate = rate * 10**(18-assetDecimals);
```

The calculation will underflow if assetDecimals is > 18.

**Impact**

This contract won't work for assets with more than 18 decimals.

**Recommendation**

Adjust the logic if the asset has more than 18 decimals.

**Status**

Acknowledged.

## [L-05] AssetManager - Invalid amount checks

**Description**

The amount in the `_deposit` function is checked to be >= 0 when it should be checked to be > 0. Also, the `verifyWithdraw` function does not establish that the amount is less or equal to balance, which will cause an underflow and revert.

**Vulnerability Details**

```
function _deposit(
      address token,
      uint amount,
      string memory to,
      bytes memory data
) internal {
      require(amount >= 0, "Amount less than minimum amount");
      ...
}



function verifyWithdraw(address token, uint amount) internal {
      uint balance = balanceOf(token);
      uint limit = calculateLimit(balance, token);
      require(balance - amount >= limit, "exceeds withdraw limit");
      ...
```

```
}
```

**Impact**

Transferring 0 tokens in `_deposit` will just burn gas. The verifyWithdraw function will revert with an underflow if the amount exceeds balance.

**Recommendation**

Do not allow 0 amounts. Check if the amount exceeds the available balance.

**Status**

Resolved.

### [L-06] XCallManager.sol#initialize - Duplicates accepted in various arrays

**Description**

There are a number of arrays used in this contract. All of which do accept duplicates which leads to issues.

**Vulnerability Details**

```
function initialize(
      address _xCall,
      string memory _iconGovernance,
      address _admin,
      string[] memory _sources,
      string[] memory _destinations
) public initializer {
      xCall = _xCall;
      xCallNetworkAddress = ICallService(xCall).getNetworkAddress();
      iconGovernance = _iconGovernance;
      admin = _admin;
      sources = _sources;
      destinations = _destinations;
      __Ownable_init(msg.sender);
```

```
}
```

These are not checked for duplicates. Also make sure to verify they are equal length if required. The same also applies to:

```
function setProtocols(string[] memory _sources, string[] memory _destinations)
external onlyOwner() {

     sources = _sources;

     destinations = _destinations;

}

…. REDACTED FOR BREVITY ..

} else if (method.compareTo(Messages.CONFIGURE_PROTOCOLS_NAME)) {

     Messages.ConfigureProtocols memory message = data

     .decodeConfigureProtocols();

     sources = message.sources;

     destinations = message.destinations;

}
```

**Impact**

Duplicates will cause issues with array comparison and when removing items.

**Recommendation**

Make sure the inputs do not contain duplicates.

**Status**

Resolved.

### [L-07] XCallManager.sol, CallService.sol#setAdmin - Transfer admin should be a two step process

**Description**

The admin can be modified, there is some risk that the wrong address is set which can not be corrected easily.

**Vulnerability Details**

```
function setAdmin(address _admin) external onlyAdmin() {
     admin = _admin;
}
```

Inputting the wrong address for admin can lead to losing access to admin only functions in the contract. It is good practice to have a two step process when setting a new admin.

**Impact**

By mistake, access to the contract may be lost.

**Recommendation**

It would be ideal to use a two-step transfer solution that the new admin needs to accept. Also, consider if this function should be onlyOwner rather than onlyAdmin.

**Status**

Resolved.

## [L-08] AssetManager.sol#configureRateLimit - Unbounded values

**Description**

Some inputs should be checked for bounds.

**Vulnerability Details**

```
function configureRateLimit(
     address token,
     uint _period,
     uint _percentage
) external onlyOwner {
     require(_percentage <= POINTS,"Percentage should be less than or equal to POINTS");
```

```
    period[token] = _period;
    ...
```

The period for some token can be set to some huge value. There is a check elsewhere which does prevent a division by 0 though.

```
function _depositNative(
    uint amount,
    string memory to,
    bytes memory data
) internal {
    require(msg.value >= amount, "Amount less than minimum amount");
    uint fee = msg.value - amount;
    ...
```

The fee used here can be any value. Make sure this is correctly implemented.

**Impact**

The code might not work as intended if these values are extremes.

**Recommendation**

Make sure this functionality is implemented as intended.

**Status**

Resolved.

### [L-09] AssetManager.sol#sendDepositMessage - Destination address not verified

**Description**

The `sendDepositMessage` takes a string as an address. The callee should verify this is the correct address.

### Vulnerability Details

```
function sendDepositMessage(
      address token,
      uint amount,
      string memory to,
      bytes memory data,
      uint fee
) internal {
      ...
}
```

### Impact

User funds might be lost if mistakes are easily made.

### Recommendation

Make sure to check that this string is a valid address.

### Status

Acknowledged, it's up to the integrations to track this for now.

## [L-10] Multiple Balanced Contracts - Code clarity improvements

### Description

There are some places where the code can be improved for clarity and maintainability.

### Vulnerability Details

```
uint private constant POINTS = 10000;
block.timestamp*1000000
```

Use _ to improve the readability of large numbers i.e.: 1_000_000.

```
for (uint j = 0; j < array2.length; j++) {
      if (array1[i].compareTo(array2[j])) {
      break;
```

```
        } else {
        if (j == array2.length - 1) return false;
        continue;
        }
}
```

The continue statement is superfluous.

```
function decodeDeposit(
        bytes memory _rlp
) internal pure returns (Messages.Deposit memory) {
        RLPDecode.RLPItem[] memory ls = _rlp.toRlpItem().toList();
        return
        Messages.Deposit(
                string(ls[1].toBytes()),
                string(ls[2].toBytes()),
                string(ls[3].toBytes()),
                ls[4].toUint(),
                ls[5].toBytes()
        );
}
```

This function is never used.

Overall the codebase also lacks comments, which are essential for helping with the maintenance of this project.

**Impact**

These problems decrease the readability and maintainability of the codebase.

**Recommendation**

Keep the code as easy to read as possible. Improve the documentation in the codebase to aid maintainability.

**Status**

Resolved.

### [L-11] Multiple Balanced Contracts - Use of old Solidity version

**Description**

The contracts allow the use of old Solidity versions starting from 0.8.0. This version was released in December 2020.

**Vulnerability Details**

```
pragma solidity >=0.8.0
```

**Impact**

Optimizations introduced by newer versions could improve performance and reduce gas costs.

**Recommendation**

Require a more recent minimum version.

**Status**

Acknowledged

### [L-12] OracleProxy#addCreditVault - Interface compliance is not checked

**Description**

The function `addCreditVault` does not check if the given address is a contract or complies with IERC4626 which is later required.

**Vulnerability Details**

```
function addCreditVault(address _vault) external onlyOwner {
     creditVaults[_vault] = true;
}
```

```
Function fetchCreditVaultPrice(address _vault) internal view
returns(Messages.UpdatePriceData memory) {

    string memory symbol = IERC4626(_vault).symbol();

    uint sharesDecimals = IERC4626(_vault).decimals();

    ...
```

**Impact**

Setting the wrong address will partially break the contract if it attempts to use this address in other calls.

**Recommendation**

Check for interface compliance or whether the address is a contract.

**Status**

Acknowledged

### [L-13] Multiple Balanced Contracts - Non Disabled Implementation Contract

**Description**

The upgradeable contracts do not disable initializers in the constructor, as recommended by the imported Initializable contract. To reduce the potential attack surface, `_disableInitializers` in the constructor need to be called.

**Vulnerability Details**

Following contracts implement `UUPSUpgradeable`:

- /src/asset-manager/AssetManager.sol
- /src/oracle-proxy/OracleProxy.sol
- /src/xcall-manager/XCallManager.sol
- /src/bnusd/BalancedDollar.sol

**Impact**

This means that anyone can call the initializer on the implementation contract to set the contract variables and assign the roles.

**Recommendation**

Build a constructor function in the upgradeable contracts that calls the `disableInitializers()` function.

**Status**

Resolved.

### [L-14] CallService.sol#setProtocolFee - Centralization risk for privileged function

**Description**

The function allows the admin/owner to set the protocol fee. This fee is not allowed to be 0 with the implemented check. However, the function lacks any upper boundaries for the amount that the fee can be set in.

**Vulnerability Details**

```
function setProtocolFee(uint256 _value) external override onlyAdmin {
        require(_value >= 0, "ValueShouldBePositive");
        protocolFee = _value;
    }
```

**Recommendation**

Implement a check to set an upper boundary for the protocol fee so that the admin can not set it to unrealistic amounts.

**Status**

Acknowledged.

### [L-15] XCallManager.sol#getModifiedProtocols - Array writes out of bounds

**Description**

The getModifiedProtocols() function assumes that the item to be removed is present in the array, which is not guaranteed.

### Vulnerability Details

```
function proposeRemoval(string memory protocol) external onlyAdmin {
     proposedProtocolToRemove = protocol;
}
// Here anything can be set as something to be removed.

function getModifiedProtocols() internal view returns (string[] memory) {
….

string[] memory newArray = new string[](sources.length - 1);
uint newIndex = 0;
for (uint i = 0; i < sources.length; i++) {
     if (!sources[i].compareTo(proposedProtocolToRemove)) {
     newArray[newIndex] = sources[i];
     newIndex++;
     }
}
```

However, if `proposedProtocolToRemove` is not found, this will write the last item out of bounds. This function does not consider duplicates an option.

### Impact

This code could revert by doing out-of-bounds writes.

### Recommendation

Make sure not to write outside of array bounds. Establish that the item to be removed is actually present in the array. Avoid duplicates.

### Status

Resolved.

# Centralisation

The Balanced project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

# Conclusion

After Hashlocks analysis, the Balanced project seems to have a sound and well-tested code base, now that our findings have been resolved or acknowledged to achieve security. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

# Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits are to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

**Manual Code Review:**

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

**Vulnerability Analysis:**

Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

**Documenting Results:**

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we still need to verify the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown not to represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

**Suggested Solutions:**

We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contracts details are made public.

# Disclaimers

## Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

## Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

# About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

**Website:** hashlock.com.au
**Contact:** info@hashlock.com.au


#Hashlock.