QuillAudits

Audit Report
October, 2023

For

ALVARA

Table of Content

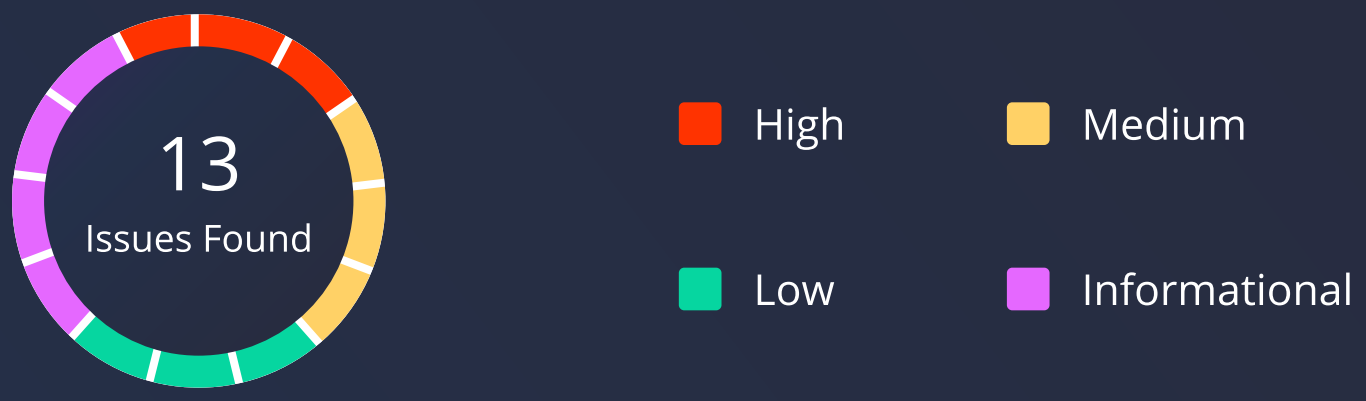
Table of Content

Table of Content

Executive Summary

Project Name	Alvara
Overview	The Alvara platform enables the creation and self-management of tokenized basket funds. Its factory provides a platform for users to design and mint their unique BTS tokens. In essence, anyone can step into the shoes of a fund manager by incorporating tokens from any supported blockchain into their BTS.
Timeline	11th September 2023 - 21st September 2023
Method	Manual Review, Automated Testing, Functional Testing, etc.
Language	Solidity
Blockchain	Ethereum
Audit Scope	The scope of this audit was to analyze the Alvara codebase for quality, security, and correctness:
	https://github.com/Alvara-Protocol/alvara-contracts/tree/main
Branch	Main
Commit	a90cbb635cc2ce1fb3fd0bdede22e02e055b357e
Contracts in Scope	Factory, Alvara, BasketTokenStandard, & BasketTokenStandardPair.
Fixed In	44413465a8229ed840bde761e66c1cd7dc224e7b
Review 2	12th October 2023 - 18th October 2023

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	3	3	3	3

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Common Issues

High Severity Issues

No issues were found.

Medium Severity Issues

A.1 The Use of `msg.value` attached in loop

Line

70 - 84

Function - Factory.sol:createBTS(), BTS.sol:contribute

```
70          for (uint i = 0; i < _tokens.length; ) {
71              IUniswapV2Router(Constants.ROUTER)
72                  .swapExactETHForTokensSupportingFeeOnTransferTokens{
73                  value: (msg.value * _weights[i]) / Constants.PERCENT_PRECISION
74              }(
75                  0,
76                  Helper.getPath(Constants.WETH, _tokens[i]),
77                  _btsPair,
78                  block.timestamp
79              );
80
81              unchecked {
82                  ++i;
83              }
84          }
```

A.1 The Use of `msg.value` attached in loop

Line

133-148

Function - Factory.sol:createBTS(), BTS.sol:contribute

```
133         for (uint i = 0; i < _tokenDetails.tokens.length; ) {
134             IUniswapV2Router(Constants.ROUTER)
135                 .swapExactETHForTokensSupportingFeeOnTransferTokens{
136                 value: (msg.value * _tokenDetails.weights[i]) /
137                     Constants.PERCENT_PRECISION
138             }(
139                 0,
140                 Helper.getPath(Constants.WETH, _tokenDetails.tokens[i]),
141                 btsPair,
142                 block.timestamp
143             );
144 
145             unchecked {
146                 ++i;
147             }
148         }
```

Impact

1. if totalETHswapped > msg.value user forces contract to pay all remaining eth used in the swap
2. if totalETHswapped < msg.value, the unused user ETH is locked in the contract.

Description

msg.value is attached multiple times to external swap calls and not compared with the actual total after swap iteration.

Remediation

Check that the sum of all ETH amounts swapped in the end is less than or matches msg.value. Consider returning the amount of unused ETH to the beneficiary if the sum < msg.value.

Status

Resolved

Low Severity Issues

A.2 Missing Event Emission for Significant Actions and UnIndexed Events

Description

Whenever certain significant actions are performed within the contract, it is recommended to emit an event about it. And also there are events in the contract but are unindexed which will fail to track on-chain changes. With the use of indexed events, forming a topic, it is possible to track for on-chain changes.

The functions to emit for are as follows:

- **Factory.sol:**
  - updateAlva
  - updateMinPercentALVA
  - updateBTSImplementation
  - updateBTSPairImplementation

- **BTS.sol:**
  - updateUpperLimit
  - updateLowerLimit

Remediation

Consider emitting an event whenever certain significant changes are made in the contracts.

Status

Resolved

Reference

https://www.rareskills.io/post/ethereum-events

A.3 Missing Check for Zero Address

Line

94 - 104

Function - updateBTSImplementation & updateBTSPairImplementation

```
94    function updateBTSImplementation(
95        address _btsImplementation
96    ) external onlyOwner {
97        btsImplementation = _btsImplementation;
98    }
99
      ftrace | funcSig
100   function updateBTSPairImplementation(
101       address _btsPairImplementation
102   ) external onlyOwner {
103       btsPairImplementation = _btsPairImplementation;
104   }
```

Description

There are some functions in the contract that allows for the passage of address input values but fails to check that these addresses are not the null address. Although these functions are privileged functions allowed for the contract owner, it is important to safeguard setting some addresses to zero addresses.

Remediation

Add a null address check to these functions to prevent setting a null address.

Status

Resolved

Informational Issues

A.4 Absence of Proper Code Comment in All Contracts

Description

Proper code comments explain the purpose of functions in the contracts and also about the kind of parameters a function expects as an input value. This is why it is recommended that contracts should have a NATSPEC code format to give a comprehensive meaning to the functions. While there are few comments that explain the data type and few lines of the code, it is not substantial.

Remediation

Add a proper natspec comment format across the contracts.

A.4 Absence of Proper Code Comment in All Contracts

Status

Acknowledged

Reference

https://docs.soliditylang.org/en/v0.8.20/style-guide.html#natspec

A.5 Functions Certain to Revert When Called by Normal Users can be Marked Payable

Description

With the use of modifiers on some functions restrict the privilege of who can call them. This gives us the certainty of some functions reverting when called by users not allowed to call this function, it is recommended to mark these functions as payable as this will save gas when they are called by privileged addresses.

Remediation

Functions with the onlyOwner functions should be marked payable to save gas.

Status

Resolved

A.6 Unlocked Solidity Version

```
2   pragma solidity ^0.8.9;
```

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here some the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Resolved

B. Factory

High Severity Issues

No issues were found.

Medium Severity Issues

B.1 Failure to Verify that Users Pay a Minimum of 1 ETHER when createBTS is Invoked

Line

53 -84

Function - createBTS

```
53      function createBTS(
54          string calldata _name,
55          string calldata _symbol,
56          address[] calldata _tokens,
57          uint[] calldata _weights,
58          string memory _tokenURI,
59          bool _enableAutoRebalance
60      ) external payable {
61          (address _bts, address _btsPair) = _initializeBTSWithPair(
62              _name,
63              _symbol,
64              _tokens,
65              _weights,
66              _tokenURI,
67              _enableAutoRebalance
68          );
69
70          for (uint i = 0; i < _tokens.length; ) {
71              IUniswapV2Router(Constants.ROUTER)
72                  .swapExactETHForTokensSupportingFeeOnTransferTokens{
73                  value: (msg.value * _weights[i]) / Constants.PERCENT_PRECISION
74              }(
75                  0,
76                  Helper.getPath(Constants.WETH, _tokens[i]),
77                  _btsPair,
78                  block.timestamp
79              );
80
81              unchecked {
82                  ++i;
83              }
```

B.1 Failure to Verify that Users Pay a Minimum of 1 ETHER when createBTS is Invoked

Description

According to the documentation, “To successfully mint the BTS, the use must make a minimum contribution of 1 ETH (or 1 ETH equivalent in any currency). The minimum contribution is designed to prevent the frivolous, excessive creation of numerous SmartFunds’. In the contract implementation, this is not added to the createBTS function. Due to the failure to verify, users are at liberty to create numerous SmartFunds with amounts less than 1 ether.

Remediation

Add a check that verifies the amount the users sent when invoking the function.

Status

Resolved

Low Severity Issues

B.2 Inaccurate Input Validation

```
ftrace | funcSig
136     function updateALVA(address _alva⬆) external onlyOwner {
137         if (alva == address(0)) revert InvalidToken();
138         alva = _alva⬆;
139     }
```

Description

The updateALVA function is primarily created to allow the owner of the Factory to update the Alva token address. The issue arises when in line 137, the if statement checks that the alva variable (a state variable already set at the initialize function) is not the null address. This check was intended to check that the input parameter passed by the contract owner is not a null address before it updates the **alva** state variable.

Remediation

Check that the input parameter - _alva - is checked instead of the state variable.

Status

Resolved

Informational Issues

No issues were found.

C. BasketTokenStandard

High Severity Issues

C.1 Immediate profit stealing of funding yield from the BasketTokenStandardPair contract due to shares not calculated after the reserve have been updated

Description

Impact:

1. An attacker is able to atomically steal large amounts of the funding yield from the BasketTokenStandardPair contract. This is due to the fact that the mint function of the BasketTokenStandardPair contract will first issue the attacker shares based on the current reserve in the contract which is not the actual reserve balance in the contract, and then increase the reserve, which updates the reserve to the actual reserve balance in the contract. The Attacker is then able to withdraw their initial funds from the protocol, while also getting an equivalent amount of the funding yield which was just deposited into the contract (immediate profit from outstanding shares they gain atomically).

2. This is also susceptible to a flash loan attack. An attacker can exploit the vulnerability by executing flash loan transactions using both the contribute() and withdraw() functions. This allows the attacker to acquire large extra lp tokens and increase their share of assets within seconds, all without any meaningful contribution. The attacker only needs to obtain a flash loan of eth as the fund asset, and then they can use the contribute function to mint shares and subsequently call the withdraw function to receive the flash loan amount along with additional stolen lp tokens.

C.1 Immediate profit stealing of funding yield from the BasketTokenStandardPair contract due to shares not calculated after the reserve have been updated

Proof Of Concept:

The **mint** function of the BasketTokenStandardPair contract is defined as follows:

```
62      function mint(address _to↑) external returns (uint liquidity↑) {
63          uint[] memory amounts = new uint[](tokens.length);
64          uint totalETH;
65
66          for (uint i = 0; i < amounts.length; ) {
67              amounts[i] =
68                  IERC20Upgradeable(tokens[i]).balanceOf(address(this)) -
69                  reserves[i];
70              if (amounts[i] == 0) revert InsufficientLiquidity();
71
72              totalETH += Helper.getAmountsOut(
73                  amounts[i],
74                  Helper.getPath(tokens[i], Constants.WETH)
75              );
76
77              unchecked {
78                  ++i;
79              }
80          }
81
82          if (totalSupply() == 0) {
83              liquidity↑ = 1000 ether;
84          } else {
85              liquidity↑ = calculateShareLP(totalETH);
86          }
87          _mint(_to↑, liquidity↑);
88
89          for (uint i = 0; i < amounts.length; ) {
90              reserves[i] += amounts[i];
91
```

As mentioned earlier, the shares are calculated using the current total amount of reserves in the protocol (**_totalReservedETH()**).

```
function calculateShareLP(uint _amountETH)
        public
        view
        returns (uint amountLP)
    {
        amountLP = ((_amountETH * totalSupply()) / _totalReservedETH());
    }
```

C.1 Immediate profit stealing of funding yield from the BasketTokenStandardPair contract due to shares not calculated after the reserve have been updated

```
function _totalReservedETH() private view returns (uint totalReservedETH) {
        for (uint i = 0; i < reserves.length; ) {
            totalReservedETH += Helper.getAmountsOut(
                reserves[i], // @note
                Helper.getPath(tokens[i], Constants.WETH)
            );

            unchecked {
                ++i;
            }
        }
    }
```

Following this calculation of shares, reserve is then increased:

```
for (uint i = 0; i < amounts.length; ) {
            reserves[i] += amounts[i];

            unchecked {
                ++i;
            }
        }
```

Importantly, at the end of this function, the reserves are then increased, this is when it actually updates the reserve to the actual reserve balance in the contract.

At this point, after the execution of the **contribute** function, the attacker can then call **withdraw**, which will send them back their initial deposit & at the same time send them a portion of the funding yield equivalent to the ratio of their balance of shares vs the total supply of shares minted using the actual updated reserve balance in the contract this time around.

C.1 Immediate profit stealing of funding yield from the BasketTokenStandardPair contract due to shares not calculated after the reserve have been updated

```
function calculateShareTokens(uint _amountLP)
        public
        view
        returns (uint[] memory amountTokens)
    {
        amountTokens = new uint[](tokens.length);
        for (uint i = 0; i < reserves.length; ) {
            amountTokens[i] =
                (_amountLP *
                    IERC20Upgradeable(tokens[i]).balanceOf(address(this))) / //@note
actual reserve balance in the protocol
                totalSupply();

            unchecked {
                ++i;
            }
        }
    }
```

Remediation

In the mint function, shares should be calculated after the reserve have been updated.

Status

Resolved

C.2 Rebalance Function can be Called by Any User to Change the Tokens and Truncate Protocol Flow

Line

194 - 235

Function - rebalance()

```
194     function rebalance(address[] memory _newTokens⬆, uint[] memory _newWeights⬆)
195         public
196         checkLength(_newTokens⬆.length, _newWeights⬆.length)
197     {
198         for (uint i = 0; i < _tokenDetails.tokens.length; i++) {
199             _isTokenPresent[_tokenDetails.tokens[i]] = false;
200         }
201         _checkValidTokensAndWeights(_newTokens⬆, _newWeights⬆);
202         IBTSPair(btsPair).rebalance();
203         uint _wethBought;
204         for (uint i = 0; i < _tokenDetails.tokens.length; ) {
205             _wethBought += _swapTokensForTokens(
206                 _tokenDetails.tokens[i],
207                 Constants.WETH,
208                 IERC20Upgradeable(_tokenDetails.tokens[i]).balanceOf(
209                     address(this)
210                 ),
211                 address(this)
212             );
213 
214             unchecked {
215                 ++i;
216             }
217         }
218         for (uint i = 0; i < _newWeights⬆.length; ) {
219             _swapTokensForTokens(
220                 Constants.WETH,
221                 _newTokens⬆[i],
222                 (_wethBought * _newWeights⬆[i]) / Constants.PERCENT_PRECISION,
223                 btsPair
224             );
225 
226             unchecked {
227                 ++i;
228             }
229         }
230         emit RebalanceBTS(address(this), _tokenDetails.weights, _newWeights⬆);
231 
232         IBTSPair(btsPair).updateTokens(_newTokens⬆);
233         _tokenDetails.tokens = _newTokens⬆;
234         _tokenDetails.weights = _newWeights⬆;
235     }
```

Description

Impact:

Anyone can choose to change the smartfunds of any targetted BTS fund manager to bad or poor tokens and destroy the targetted BTS, affecting all of the contributions made on the BTS at zero cost to the attacker.

C.2 Rebalance Function can be Called by Any User to Change the Tokens and Truncate Protocol Flow

Proof of Concept:
As specified in the doc:
BTS:Rebalance():
"Allows the owner to rebalance the protocol by specifying new tokens and weights."
as you can see, this function was intended to be called only by the owner.

However, the current implementation of the rebalance() function has it function visibility set to **public** without any access control:

```
194     function rebalance(address[] memory _newTokens⬆, uint[] memory _newWeights⬆)
195         public
196         checkLength(_newTokens⬆.length, _newWeights⬆.length)
197     {
198         for (uint i = 0; i < _tokenDetails.tokens.length; i++) {
199             _isTokenPresent[_tokenDetails.tokens[i]] = false;
200         }
201         _checkValidTokensAndWeights(_newTokens⬆, _newWeights⬆);
202         IBTSPair(btsPair).rebalance();
203         uint _wethBought;
204         for (uint i = 0; i < _tokenDetails.tokens.length; ) {
205             _wethBought += _swapTokensForTokens(
206                 _tokenDetails.tokens[i],
207                 Constants.WETH,
208                 IERC20Upgradeable(_tokenDetails.tokens[i]).balanceOf(
209                     address(this)
210                 ),
211                 address(this)
212             );
213 
214             unchecked {
215                 ++i;
216             }
217         }
218         for (uint i = 0; i < _newWeights⬆.length; ) {
219             _swapTokensForTokens(
220                 Constants.WETH,
221                 _newTokens⬆[i],
222                 (_wethBought * _newWeights⬆[i]) / Constants.PERCENT_PRECISION,
223                 btsPair
224             );
225 
226             unchecked {
227                 ++i;
228             }
229         }
230         emit RebalanceBTS(address(this), _tokenDetails.weights, _newWeights⬆);
231 
232         IBTSPair(btsPair).updateTokens(_newTokens⬆);
233         _tokenDetails.tokens = _newTokens⬆;
234         _tokenDetails.weights = _newWeights⬆;
235     }
```

C.2 Rebalance Function can be Called by Any User to Change the Tokens and Truncate Protocol Flow

This means anyone can choose to change the smartfunds of any targetted BTS fund manager to bad or poor tokens and destroy the targetted BTS, affecting all of the contributions made on the BTS at zero cost to the attacker.

Remediation

Add the onlyCreator modifier to this function.

Fixed

Separated the rebalance function to have private and public functions. The private function was called in the _autoRebalance and the public was designed for the NFT owner to invoke.

Status

Resolved

Medium Severity Issues

C.3 Index Out of Bound

Line

19

Variable - _alvaIndex / Function - _checkValidTokensaAndWeights

```
19    uint private _alvaIndex;
```

Line

90 - 118

Variable - _alvaIndex / Function - _checkValidTokensaAndWeights

```
      ftrace | funcSig
90    function _checkValidTokensAndWeights(
91        address[] memory _tokens⬆,
92        uint[] memory _weights⬆
93    ) private {
94        _alvaIndex = type(uint).max;
95
96        uint _totalWeight;
97
98        for (uint i = 0; i < _tokens⬆.length; ) {
99            if (!_isTokenPresent[_tokens⬆[i]] && _weights⬆[i] != 0) {
100               if (_tokens⬆[i] == IFactory(factory).alva()) _alvaIndex = i;
101
102               _isTokenPresent[_tokens⬆[i]] = true;
103               _totalWeight += _weights⬆[i];
104           } else {
105               revert InvalidToken();
106           }
107
108           unchecked {
109               ++i;
110           }
111       }
112
113       if (
114           _weights⬆[_alvaIndex] < IFactory(factory).minPercentALVA() ||
115           _totalWeight != Constants.PERCENT_PRECISION
116       ) revert InvalidWeight();
117   }
118
```

C.3 Index Out of Bound

Description

The issue of index out of bounds arises when users invoke the primary createBTS function from the Factory contract in order to create new instance of the BTS and BTSPair. This issue disrupts the successful creation of the BTS because of the update being done to the **_alvaIndex** state variable. When the createFunction is triggered, it creates new instances of the BTS and BTSPair and afterwards invoke the initialize function from the BTS contract.

Within the initialize function in BTS contract, the _alvaIndex variable is updated to the maximum of data type uint256 (Line 70) and also updated again (Line 94) in the **_checkValidTokensAndWeights** function (Informational - to save gas, invoke once) but transaction would encounter index out of bounds within the _checkValidTokensAndWeights function when it reaches the if block (Line 113).

Supposing one of the token addresses passed is the Alva token address (Line 100), the _alvaIndex would change from the `type(uint256).max` to the index of the address in the 'token[i]' array, this will escape the index out of bounds error when it reaches Line 114 where _weight[_alvaIndex] < IFactory(factory)minPercentLVA() is meant to verify that the index should not be less than the minPercentALVA() read from the factory contract. However, if it is a case where none of the addresses is Alva token, _alvaIndex will be the maximum of type uint256 and on trying to know the location of _alvaIndex in _weight will cause the whole transaction to revert.

Remediation

Separate the if conditions in line 113. The first condition can be added to the condition in line 100 like the example below:

```
if (!_isTokenPresent[_tokens[i]] && _weights[i] != 0) {
    if (_tokens[i] == IFactory(factory).alva()) {
    _alvaIndex = i;
    if( _weights[_alvaIndex] < IFactory(factory).minPercentALVA()) revert InvalidAlvaWeight();
    }
```

This way, after the first if condition confirms that one of the provided input token parameters is Alvara token address, it sets the _alvaIndex to the value of the index of tokens[i] then the other condition confirms that it has the expected weight or revert. By doing so it will be needless to set _alvaIndex to the max of type uint256.

Status

Resolved

Low Severity Issues

No issues were found.

Informational Issues

C.4 OwnableUpgradable is Initialized but Not Used

Line

13

Library - OwnableUpgradable

```
13  contract BasketTokenStandard is
14      ERC721URIStorageUpgradeable,
15      OwnableUpgradeable
```

Description

In the BTS contract, the OwnableUpgradable was initialized at the initialize function but none of its modifiers or functions were invoked. Instead, the current implementation has a different modifier called onlyCreator.

Remediation

Remove the OwnableUpgradable library. Since the Factory contract would always become the owner when new instances of a BTS contract is created, it does not have any privilege functions to call; this is to help save gas.

Status

Resolved

D. BasketTokenStandardPair

High Severity Issues

D1. Users can steal any funds stored in the BasketTokenStandardPair contract due to missing access control or validation

Description

Impact:

1. Any funds minted or deposited into btsPair contract and not used in that transaction can be stolen immediately
2. Unintended behaviour as it expected to be interacted with only through bts for auto balancing logic but is bypassed

Proof Of Concept:

User interact with the BTSPair contract through the BTS contract which safely handles the interaction and ensure to autorebalance.

```
        IBTSPair(btsPair).mint(msg.sender);
        _autoRebalance(); // @note auto rebalance
```

```
        emit WithdrawnFromBTS(address(this), _withdraw(_liquidity, msg.sender));

        _autoRebalance();// @note auto rebalance
```

However, users are still able to interact with the BTSPair directly due to missing access control in both the mint and burn functions:

D1. Users can steal any funds stored in the BasketTokenStandardPair contract due to missing access control or validation

```
ftrace | funcSig
62  function mint(address _to↑) external returns (uint liquidity↑) {
63      uint[] memory amounts = new uint[](tokens.length);
64      uint totalETH;
65
66      for (uint i = 0; i < amounts.length; ) {
67          amounts[i] =
68              IERC20Upgradeable(tokens[i]).balanceOf(address(this)) -
69              reserves[i];
70          if (amounts[i] == 0) revert InsufficientLiquidity();
71
72          totalETH += Helper.getAmountsOut(
73              amounts[i],
74              Helper.getPath(tokens[i], Constants.WETH)
75          );
76
77          unchecked {
78              ++i;
79          }
80      }
81
82      if (totalSupply() == 0) {
83          liquidity↑ = 1000 ether;
84      } else {
85          liquidity↑ = calculateShareLP(totalETH);
86      }
87      _mint(_to↑, liquidity↑);
88
89      for (uint i = 0; i < amounts.length; ) {
90          reserves[i] += amounts[i];
91
92          unchecked {
93              ++i;
94          }
95      }
96  }
```

D1. Users can steal any funds stored in the BasketTokenStandardPair contract due to missing access control or validation

```
ftrace | funcSig
156  function burn(address _to⬆) external returns (uint[] memory amounts⬆) {
157      uint _liquidity = balanceOf(address(this));
158
159      _burn(address(this), _liquidity);
160
161      amounts⬆ = calculateShareTokens(_liquidity);
162      for (uint i = 0; i < tokens.length; ) {
163          if (amounts⬆[i] == 0) revert InsufficientLiquidity();
164          IERC20Upgradeable(tokens[i]).safeTransfer(_to⬆, amounts⬆[i]);
165
166          reserves[i] -= amounts⬆[i];
167
168          unchecked {
169              ++i;
170          }
171      }
172  }
```

Notice that it transfers any funds gotten from the contract to the address the caller specifies, same with the mint, at no cost to the caller.

Remediation

The ownership of this contract is owned by the factory and later transfer to the BTS and therefore mint and burn should only be called or interacted with by the owner which is the BTS contract.

Add onlyOwner modifier as done correctly in the rebalance() function.

Status

Resolved

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

E. Alvara

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

Functional Tests

Some of the tests performed are mentioned below:

- Should be able to mint and burn tokens
- Should be able to contribute and gain LP token
- Should be able to withdraw
- Should owner be able to rebalance
- Should regulate the rebalancing with the update of Upper and Lower limits
- Should know the state of effect when Fund managers rebalance to new tokens
- Should verify that the Alva ERC20 token can be minted beyond 200 million tokens
- Should affirm that only the contract owner can mint the Alvara token
- Should be able to create BTS as a fund manager
- Should successfully create BTS with amounts less than 1 ether
- Should confirm the new instances of the BTS created after the update of BTS and BTS pair address in Factory
- Should revert when unequal array of tokens and weights are passed
- Should revert when the totalWeights is not equal to the percent precision
- Should confirm the issues of index out of bounds in the _checkvalidTokensAndWeights

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Alvara Protocol. We performed our audit according to the procedure described above.

Some High, Medium, Low and Informational Issues were Found during the audit, Almost All Issues has been Resolved by Alvara Team.

Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Alvara smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Alvara smart contract. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Alvara to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2023

For

ALVARA

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com