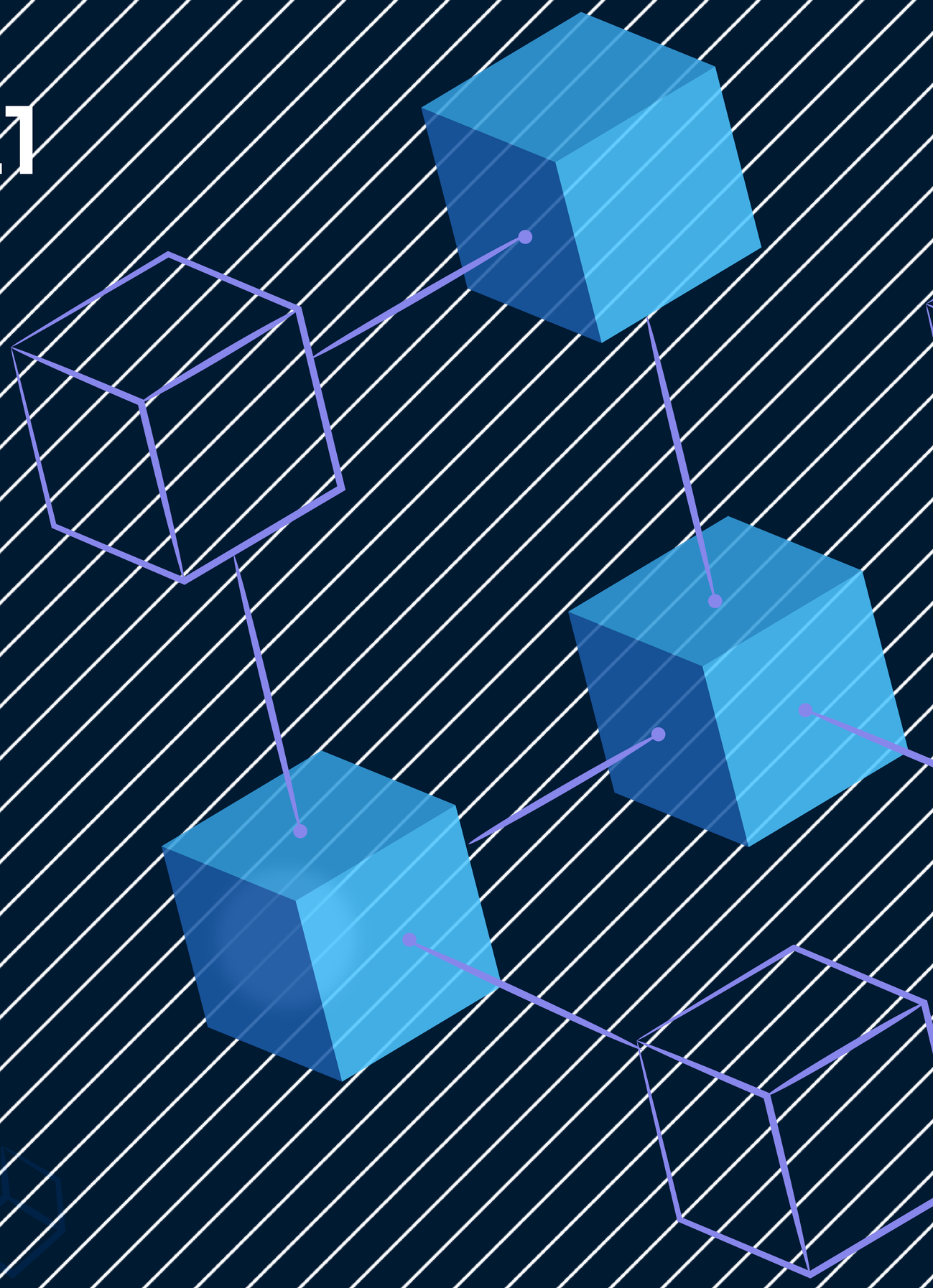
QuillAudits

Audit Report
November, 2021



For

MatrixETF

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the MatrixETF Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 contract, 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 analysed 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
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 analysed, 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 behaviour 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, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis

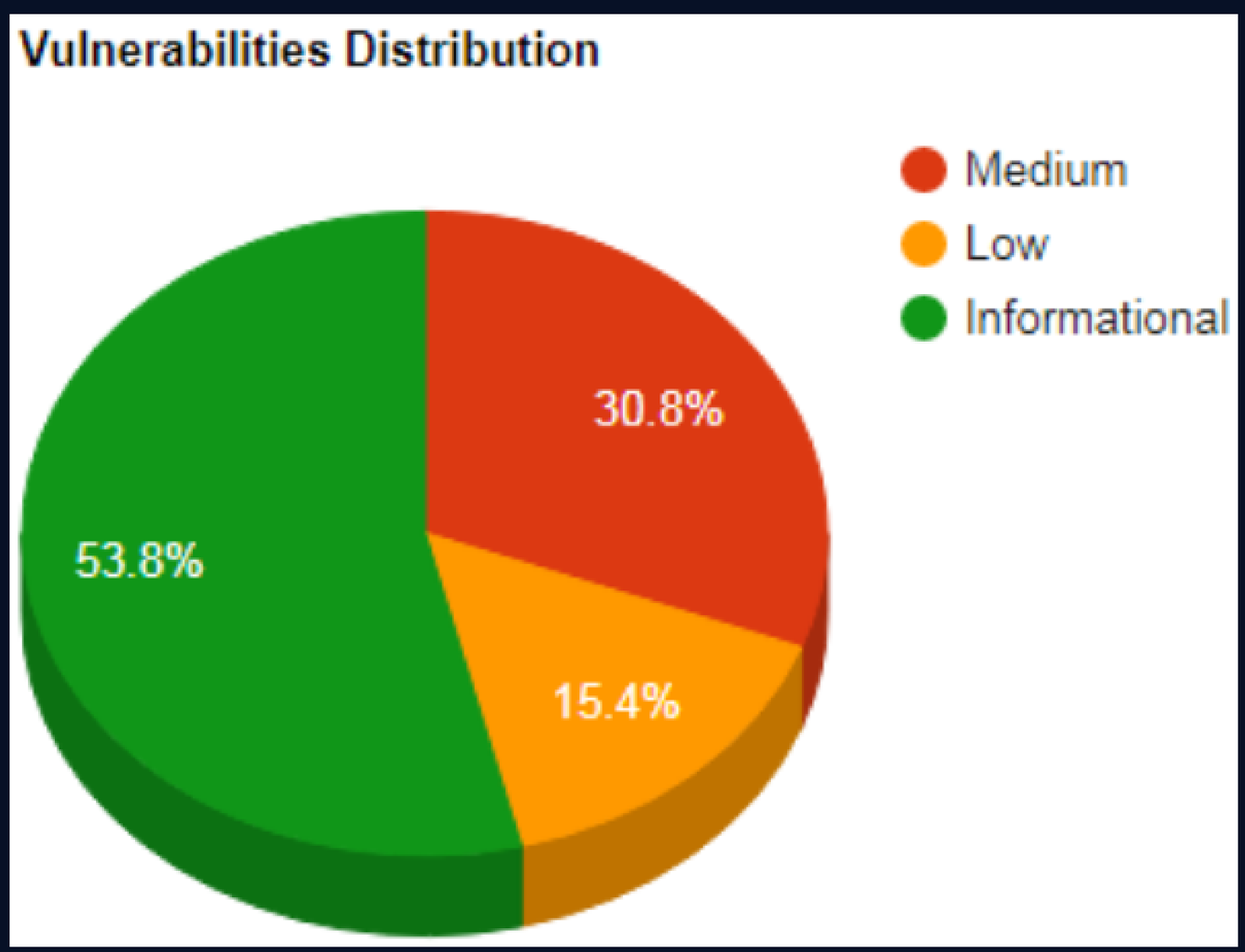
Issue Categories

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.

Risk-level	Description
High	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	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	Low-level severity issues can cause minor impact and or 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 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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	4	2	7
Closed	0	0	0	0



Introduction

During the period of **October 20, 2021 to November 2, 2021** - QuillAudits Team performed a security audit for **MatrixETF** smart contracts.

The code for the audit was taken from following the official link:
- Codebase: https://github.com/MatrixETF/MatrixETF-Pool-V2

Ver	Date	Commit hash	Branch
1	October	2d5c5252e17f91680645fc6fb0d2108ff681037b	Master

Issues Found

A. Contract – MatrixETFPool

High severity issues

No issues were found.

Medium severity issues

M.1 Unused SafeERC20 wrapper functions for IERC20 operations

BPool implements **SafeERC20** as a wrapper for IERC20 operations, which provide Wrappers around ERC20 operations that throw on failure (when the token contract returns false). Tokens that return no value (and instead revert or
throw on failure) are also supported, non-reverting calls are assumed to be
successful.
But it doesn't use **safeTransfer** and **safeTransferFrom** wrapper functions for token operations.

```
1185        function _pullUnderlying(address erc20, address from, uint amount)
1186            internal
1187        {
1188            bool xfer = IERC20(erc20).transferFrom(from, address(this), amount);
1189            require(xfer, "ERC20_FALSE");
1190        }
1191
1192        function _pushUnderlying(address erc20, address to, uint amount)
1193            internal
1194        {
1195            bool xfer = IERC20(erc20).transfer(to, amount);
1196            require(xfer, "ERC20_FALSE");
1197        }
```

Status: Acknowledged
Developer Comments: The team will take care of it and will make sure that there should not be any invalid token in the pool.

M.2 Possibly Incorrect Token Amount Check

Function **joinswapExternAmountIn** takes **tokenAmountIn** worth of tokens from a user in order to mint **poolAmountOut** worth of shares. The **poolAmountOut** is calculated over **tokenAmountInAfterFee** (after deducting community join fee from **tokenAmountIn**).

```
978             poolAmountOut = calcPoolOutGivenSingleIn(
979                                 inRecord.balance,
980                                 _getDenormWeight(tokenIn),
981                                 _totalSupply,
982                                 _getTotalWeight(),
983                                 tokenAmountInAfterFee,
984                                 _swapFee
985                             );
```

tokenAmountInAfterFee worth of tokens later are pulled by user and are added to the token's record balance and **poolAmountOut** is minted and allocated to the user as share.

But the function checks and restricts **tokenAmountIn** to not exceed the maximum number of input tokens allowed instead of **tokenAmountInAfterFee** which is the one used to calculate **poolAmountOut** and the one actually added to the contract's balance. As a consequence, the function may disallow legitimate input to mint shares. For eg: Let's say tokenAmountIn is exceeding the maximum number of input tokens allowed, but after deducting the fees, **tokenAmountInAfterFee** may pass this required check.

Status: Acknowledged
Developer Comments: The team will reevaluate the checks!

M.3 Binding Tokens may exceed MAX_TOTAL_WEIGHT

Contract defines a limit for tokens to be binded that the total weight of all tokens should not exceed **MAX_TOTAL_WEIGHT** but doesn't restrict tokens for the same, while binding/rebinding tokens.

Status: Acknowledged
Developer Comments: The team will make sure that the total weight of tokens should not exceed **MAX_TOTAL_WEIGHT** while binding and rebinding.

M.4 Allowed Token Management Actions irrespective of finalized status:

Contract supports some **Token Management** actions as and when required, but doesn't consider the **finalized** status, as a result allows it to update values even after the pool gets finalized.
Some of the functions include:

- **bind**
- **rebind**
- **unbind**

Practical Scenario:
Controller may **bind/unbind/rebind** a token after the finalization of a pool, taking away/modifying the token liquidity of the pool and making an imbalance with pool shares and token liquidity. It may also affect the future minting/burning of pool shares as totalSupply and total denormalized weight have not been updated, reflecting the changes by **binding/unbinding/rebinding** of tokens.

Status: Acknowledged
Developer Comments: The team will strictly manage the administration.

Low severity issues

L.1 Unimplemented Functions & Code with No Use

Contract does unnecessary operations over some unimplemented functions. The unnecessary functions can be removed to optimize the code and logic.
Unimplemented functions are:
- _addTotalWeight
- _subTotalWeight
- gulp

Code with No Use: function **rebind** takes the desired token weight and performs some operations with **_addTotalWeight** and _subTotalWeight based on whether the **oldWeight** is less/more than the desired weight.

```
558         uint oldWeight = _records[token].denorm;
559         if (denorm > oldWeight) {
560             _addTotalWeight(bsub(denorm, oldWeight));
561         } else if (denorm < oldWeight) {
562             _subTotalWeight(bsub(oldWeight, denorm));
563         }
```

As these functions are not implemented, the code is meaningless and hence can be optimized/removed. Also, Contract **MatrixPool** overrides these functions but again doesn't add any code logic to them.

Recommendation: Consider optimizing the code

Status: Acknowledged

Developer Comments: The functions are reserved to be used in a later version.

L.2 Missing Zero Address Validation

Function **initialize()**: Missing Zero Address Check for **controller** address
Function **setCommunityFeeAndReceiver()**: Missing Zero Address Check for c**ommunityFeeReceiver** address
Function **setController()**: Missing Zero Address Check for **manager** address
Function **setWrapper()**: Missing Zero Address Check for **wrapper** address

Status: Acknowledged

Informational issues

INF1 Missing Zero Address Validation

_totalWeight defines Pool's total denormalized weight, but the contract contains no logic to update it, hence unused. Instead, MatrixPool overrides **_getTotalWeight** to calculate it, whenever called.

Recommendation: Consider optimizing the code

Status: Acknowledged

INF2 Allowed Configuration Actions irrespective of finalized status:

Contract supports some configuration actions as and when required, but doesn't consider the **finalized** status, as a result allows it to update values even after the pool gets finalized.
Some of the functions include:

- **setSwapFee**
- **setCommunityFeeAndReceiver**
- **setSwapsDisabled**
- **setRestrictions**

Status: Acknowledged

INF3 MatrixPool's constructor passes empty name and symbol string values to BPool constructor

```
35
36        constructor() public BPool("", "") {}
37
```

Status: Acknowledged

INF4 **function joinswapExternAmountIn** deducts **_communityJoinFee** from input token amount while **function joinswapPoolAmountOut** deducts it from output shares. Incorrect values may result in inappropriate deductions.

Status: Acknowledged

INF5 **Token Mismatch and Incorrect Share Burning while exiting pool**

Joining a pool mints pool shares for an appropriate amount of input token. Also exiting a pool burns the appropriate pool shares for a desired output token. Exiting a pool doesn't check for which token has been opted as an output token, and thus a token may be chosen, for which the liquidity had never been added into the pool while joining, and hence may create an unexpected imbalance.

Status: Acknowledged

INF6 **block.timestamp** has been used for comparisons

Recommendation: Avoid using **block.timestamp** as it may be manipulated by miners

Status: Acknowledged

INF7 **Inefficient Strict Comparisons**

[#L700]: require check in **joinPool** strictly restricts **tokenAmountIn** to be less than **maxAmountsIn** which would be inefficient for cases where **tokenAmountIn** equals to **maxAmountsIn**

[#L757]: require check in **exitPool** strictly checks for **tokenAmountOut** to be more than **minAmountsOut** which would be inefficient for cases where **tokenAmountOut** equals to **minAmountsOut**

Recommendation: Replacing checks from **strictly less/more than** to **less than or equal to/more than or equal to** would be more efficient.

Status: Acknowledged

Functional Test Cases

- Only **Controller** should be able to take **Configuration**, **Voting** and **Token Management** Actions

  PASS
- Should not perform **Token Management** Actions once the pool is finalized

  FAIL
- **Swap fee** & **Community swap/join/exit** fee should only be set within bounds of **MIN_FEE** and **MAX_FEE** i.e, **BONE/10******6** and **BONE/10**

  PASS
- Should not finalize if tokens are less than a set **MIN_BOUND_TOKENS** i.e **2**

  PASS
- Should not bind a token with weight not within bounds of **MIN_WEIGHT** and **MAX_WEIGHT** i.e, **1000000000** and **BONE * 50**

  PASS
- Should not bind a token with balance less than a **MIN_BALANCE** of **BONE / 10******12**

  PASS
- Total weight of tokens should not exceed **MAX_TOTAL_WEIGHT** while binding tokens

  FAIL
- Should not bind same token twice

  PASS
- Should mint **INIT_POOL_SUPPLY** shares to controller once **Finalized**

  PASS

- **Liquidity Provider and Token Swaps** should only be done either by **Wrappe**r or in **Non-Wrapper Mode** once the pool is **finalized**

  PASS
- **Liquidity Provider and Token Swaps** should only happen **once per block**

  PASS
- **joinPool** should mint appropriate shares based on **totalSupply** and desired **poolAmountOut** and should fetch appropriate bounded tokens' amount based on the corresponding tokens' balance available in the pool

  PASS
- **exitPool** should burn appropriate shares based on **totalSupply** and desired **poolAmountIn** and should send appropriate bounded tokens' amount based on the corresponding tokens' balance available in the pool. A marginal share should be minted to **_communityFeeReceiver** by deducting **_communityExitFee**.

  PASS
- **Token Swaps**, **Join Swaps** and **Exit Swaps** should not work if the **public swap** is **disabled**

  PASS
- **swapExactAmountIn** and **swapExactAmountOut** should swap **tokenIn** for appropriate amount of **tokenOut** after deducting **_communitySwapFee** from input/output tokens amount. Swap Price also should not exceed the desired **maxPrice**. Input/Output token amount should be within the desired **maximum/minimum value**. and input/output tokens for swap should not exceed a set maximum input/output swap value.

  PASS

- **joinswapExternAmountIn** and **joinswapPoolAmountOut** should mint appropriate shares for input tokens **tokenAmountIn** after deducting a **_communityJoinFee** from input tokens/pool share which will be forwarded to **_communityFeeReceiver**. Also the output shares should be more than a desired **minimum value** and input tokens should not exceed a desired **maximum value** respectively.

  PASS
- **joinswapExternAmountIn** should restrict **tokenAmountInAfterFee** to not exceed maximum input value to calculate output shares

  Reason: It restricts **tokenAmountIn** instead

  FAIL
- **joinswapPoolAmountOut** should restrict **tokenAmountIn** to not exceed maximum input value to calculate output shares

  PASS
- **exitswapPoolAmountIn** and **exitswapExternAmountOut** should burn appropriate pool shares and send output tokens after deducting a **_communityExitFee** from the output tokens which will be forwarded to **_communityFeeReceiver**. **Output tokens/Input shares** should be **more/less** than a desired **minimum/maximum value** respectively. Also the output tokens should not exceed the maximum output value

  PASS
- Controller should set dynamic weight of a **bound** token within the bounds of set **_minWeightPerSecond** and **_maxWeightPerSecond** with a target denormalized weight within the MIN_WEIGHT and **MAX_WEIGHT** bounds

  PASS
- Total target denormalized weight should not exceed **MAX_TOTAL_WEIGHT** while setting dynamic weights

  PASS

Goerli Testnet Contract Addresses [Used while functional testing]

MatrixPool: 0xa5bf4d413cDD32230E75222bf34F33bad9a152cA
TokenX: 0xd96CaEE63f01921FDb5095f035e1d91320C16C5B
TokenY: 0xE1d00aaACE300f41b43eEC62eC7E6C7e5d112603
TokenZ: 0x08B250bb3a75E09Bc4a59444c873fc1DEEEc0aAc

Closing Summary

Overall, smart contracts are very well written, documented and adhere to guidelines. Several issues of Medium and Low issues have been reported. Some suggestions are reported in order to improve the code quality of contracts.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **MatrixETF** platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **MatrixETF** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
November, 2021

For

MatrixETF

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com