OpenZeppelin | security

Trufin Audit

July 11, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-06-05 To 2023-06-23
Languages	Solidity

Total Issues	20 (18 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	7 (5 resolved, 1 partially resolved)
Notes & Additional Information	12 (12 resolved)

Scope

We audited the `https://github.com/TruFin-io/staker-audit-april` repository at the `9f199451b5220f73cfc1eb95dc13381acf804b15` commit.

In scope were the following contracts:

```
contracts/
   ├─ helpers/
   │  └─ MasterWhitelist.sol
   ├─ interfaces/
   │  ├─ IMasterWhitelist.sol
   │  ├─ IStakeManager.sol
   │  ├─ ITruStakeMATICv2.sol
   │  └─ IValidatorShare.sol
   └─ main/
      ├─ TruStakeMATICv2.sol
      ├─ TruStakeMATICv2Storage.sol
      └─ Types.sol
```

System Overview

The `TruStakeMATICv2` and its supporting contracts allow users to stake MATIC via delegation to a trusted validator and distribute the rewards to arbitrary recipients in the form of vault shares. For this purpose, a modified version of the `ERC-4626 Tokenized Vault Standard` was implemented. The system can be better explained in the following parts:

`TruStakeMATICv2` Contract

This contract provides most of the functionality. It inherits from OpenZeppelin's `ERC4626Upgradeable`. Users who want to stake MATIC deposit it into the vault and receive `TruMATIC` shares in return. The vault immediately transfers this MATIC to a trusted validator. As stated in their documentation, the TruFin team is currently using Twinstake as their staking partner. Staked MATIC produces rewards that are periodically restaked when a user makes a deposit or when dedicated restaking functions are called. The protocol regularly monitors unstaked rewards off-chain, restaking them when they cross a predefined threshold. More information about the restaking mechanism can be found in the protocol's documentation.

Users can choose to allocate any percentage of their future rewards to arbitrary addresses, namely, recipients. Allocations can be loose or strict. Loose allocations do not enforce rewards distribution and users who performed them can cancel them at any time. On the other hand, strict allocations guarantee reward distribution. Recipients will receive `TruMATIC` shares according to the allocated amount. Currently, it is only possible to perform loose allocations, but this configuration might change in the future.

It is important to clarify that the deposited MATIC is always the property of the user who deposited it. The only asset that is transferred to the allocation's recipient is a proportional part of the earned rewards.

Protocol fees are deducted from the rewards in the form of `TruMATIC` shares. These shares are minted to the `treasuryAddress` address. Additionally, a fee is taken upon reward distribution.

`MasterWhitelist` Contract

This contract implements the protocol's required whitelisting. Users who would like to interact with it must undergo a KYC process. This process relies on third-party vendors who perform the initial verification but also regularly monitor users' wallets to detect deviations from the accepted guidelines. In specific cases, manual validation can be performed by privileged users called `Lawyers` by the protocol.

If a previously whitelisted user fails a monitoring check, they are temporarily blacklisted. The `investigation_period` parameter defines the amount of time that the investigation process can last.

It is important to mention that users can be removed from the whitelist at any time. If this happens, their funds will be locked in the protocol until they are whitelisted again. They can nevertheless still freely transfer their `TruMATIC` shares.

To receive allocations it is not necessary to be whitelisted. In case a non-whitelisted address receives `TruMATIC` shares it will be possible to interact with them as with any other ERC-20 token. However, only whitelisted addresses can redeem `TruMATIC` shares for MATIC tokens.

Privileged Roles

The protocol implements ownership for both the `TruStakeMATICv2` and `MasterWhitelist` contracts.

In the `TruStakeMATICv2` contract, the owner can:

- Change the address of the `validator_share` contract. This contract acts as the interface with the designated Validator.
- Change the address of the `whitelist` contract. This contract manages the whitelist used to allow or deny users into the protocol.
- Change the `treasuryAddress` address. This address receives the fees charged by the protocol.
- Change the maximum amount that can be deposited in the vault.
- Change the amount taken as fee from rewards by the protocol.
- Change the amount taken as fee upon reward distribution by the protocol.
- Change the amount used to offset rounding.
- Enable or disable strict allocations.

- Modify the ownership of the contract.

In the `MasterWhitelist` contract, the owner can:

- Modify the ownership of the contract.
- Have the role of a lawyer.

Users designated as lawyers can:

- Add or remove `Swap Managers`.
- Change the investigation period.
- Add or remove lawyers.
- Add or remove users from the whitelist.
- Add users to the whitelist specifying the provider.
- Add or remove users from the blacklist.
- Add users to the blacklist indefinitely.
- Add or remove market makers from the whitelist.
- Add market makers to the whitelist with a specific ID.
- Add or remove vaults from the whitelist.
- Add or remove assets from the whitelist.
- Add or remove countries from the blacklist.
- Change the `KYCPassport`, `KYCReader` and `KYCRegistry` addresses used to perform user validation by the third-party providers.

It is expected that these privileged addresses will act in the protocol's best interest. Currently, the EOA `0xDbE6ACf2D394DBC830Ed55241d7b94aaFd2b504D` is set as the owner of both contracts.

Security Assumptions

- Third-party applications used to perform users' whitelisting were out of the scope of this audit. It is expected that they will work correctly.
- It is expected that off-chain processes used by the protocol will promptly notice any change in the users' status (verification revocation) and will trigger the required on-chain actions.

Client-Reported Issues

Wrong State Update in `distributeRewards`

There are two types of allocations in the TruStake protocol: strict and loose. In loose allocations, there is no guarantee that the rewards will be distributed to the recipients and the `distributeRewards` function can only be called by the allocator(distributor). If the allocation is strict, the rewards from the staked MATIC are guaranteed and anyone should be able to call the `distributeRewards` function to distribute the rewards to the recipient.

When the `distributeRewards` function is called, it calls the `_distributeRewardsUpdateTotal` function which distributes the reward and updates the state. The `_distributeRewardsUpdateTotal` updates the `totalAllocated` of the `msg.sender` instead of the distributor. Since anyone can call the `distributeRewards` function, this leads to wrong state updation.

Update: *Resolved in pull request #1 at commit 091b908.*

High Severity

H-01 Incorrect Calculation of Total Amount Staked

The `TruStakeMATICv2` contract implements the `totalStaked` function to calculate the total amount of MATIC staked by the vault on the validator. The function incorrectly calculates this, as it divides the amount of shares held by the vault by the `exchangeRate`, where it should multiply by it.

As the current `exchangeRate` is 1, the result of the function is still correct. However, if the `exchangeRate` value changes, the result will be incorrect.

In contrast, the `getTotalStake(address user)` function from the `ValidatorShare` contract performs the same operation but *multiplies* the amount of shares held by the `user` address passed as a parameter. Additionally, this function takes into account the exchange rate decimals of the validator, making it unnecessary to hardcode it.

Consider using `getTotalStake` function of the `ValidatorShare` contract to get the total amount of MATIC staked by the vault. Additionally, consider including additional tests to validate the calculated amount.

Update: *Resolved in pull request #1 at commit 89c54da.*

Low Severity

L-01 Lack of Event Emission After Sensitive Actions

The following functions do not emit relevant events after executing sensitive actions.

`MasterWhiteList` contract:

- When changing the KYC Passport address
- When changing the KYC Reader address

- When changing the KYC Registry address
- When adding or removing assets from the whitelist
- When adding or removing countries from the blacklist
- When countries are added to the `blacklistedCountries` mapping
- When a user is removed from the blacklist by being added to the whitelist via `addUserToWhitelist` function
- When a user is removed from the blacklist by being added to the whitelist via `addUserToWhitelistWithProvider` function
- When setting the id for a market maker

Consider emitting events after sensitive changes occur to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Resolved in pull request #1 at commits 5a95710, 9ea10bf and 71cc3a4.*

L-02 Hardcoded Values

Throughout the codebase, there are several occurrences of literal values with unexplained meanings. For example:

`TruStakeMATICv2` contract:

- In the `totalStaked` function, the amount of shares returned by the validator is multiplied by `1e29`.
- In the `isClaimable` function, the epochs needed to allow withdrawing are calculated adding `80`.
- In the `setEpsilon` function, the parameter is checked against `1e12`.

`MasterWhitelist` contract:

- Consider documenting the origin of `CODE_RISK` and `CODE_COUNTRY`.
- Digits `0` to `4` are often used to represent user types.
- In `addUserToWhitelistUsingPassport`, the passport's AML risk is compared against `5`.

To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name.

Update: *Resolved in pull request #1 at commits 4b9e4d4 and d0453ae.*

L-03 Use of Deprecated Method

`The MasterWhitelist` contract uses the `balanceOf` function from Quadrata's `QuadReader` contract. According to Quadrata's documentation, this function has been deprecated.

Consider instead using `getAttributes` function to query for a specific attribute and validate the returned result.

Update: *Acknowledged, not resolved. The TruFin team stated:*

> *Not fixed. We have decided not to use this method for a few reasons. This method as it is works well for us and the team at Quadrata has assured us that they do not plan to remove it. Also, this method is free to use, as opposed to the alternative method* `getAttributes` *which we would have to pay for.*

L-04 Missing Docstrings

Some of the following contracts have functions with incomplete or lacking docstrings:

- `MasterWhitelist.sol`
- `TrueStakeMaticv2.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well.

Also, it is unclear from the documentation how the `idMM`, `whitelistedVaults`, `whitelistedAssets` and `whitelistedSwapManagers` mappings will integrate with the rest of the system. Consider documenting their use cases.

Update: *Resolved in pull request #1 at commits 52525b0 and f7555da.*

L-05 A Future Action Can Lead to a Vulnerability

In upgradeable contracts, every new version of the implementation contract needs to have all the state variables of the previous implementation in the same layout or else it can lead to a storage collision. Attackers can use this vulnerability to overwrite sensitive information.

`TruStakeMATICv2` is an upgradeable contract. The `TruStakeMATICv2Storage` contract stores the state variables for the `TruStakeMATICv2` contract. On lines 55 and 58 of the `TruStakeMATICv2Storage` contract, the comments indicate that the developer may move the two storage variables up in the layout. Doing this will cause storage collision with implementation versions and may lead to a vulnerability.

Consider keeping the same storage layout throughout all versions of the `TruStakeMATICv2` contract and removing the misleading comments.

Update: *Resolved in pull request #1 at commit fcd6b61.*

L-06 Removing Market Makers From Whitelist Can Leave Inconsistent State

The `idMM` mapping is used to connect a market maker's address to the market maker id they belong to. Key-value pairs are set using either the `addMMToWhitelistWithId` or the `setIdMM` functions.

When removing a market maker from the whitelist, only the `whitelistedMMs` mapping is altered, while `idMM` remains set. Depending on future integrations with `idMM`, this could lead to a vulnerability.

When removing a market maker from the whitelist, consider also deleting the corresponding entry from the `idMM` mapping.

Update: *Resolved in pull request #1 at commit 3498b70.*

L-07 Some Functions Are Not ERC-4626 Compliant

Since the `TruStakeMATICv2` contract is an ERC-4626 vault, it is important that it complies with all the specifications of the standard. Some functionality of the vault diverges from the standard:

- The `maxDeposit` and `maxMint` functions must not revert under any circumstances.
  - The `maxDeposit` function in `TruStakeMATICv2` will revert if `cap` < `totalStaked()`.

    - The `maxMint` function would also revert under the same circumstance as it makes a call to the `maxDeposit` function.

- The ERC-4626 standard stipulates that an approved EIP-20 spender is able to call the `deposit`, `mint`, `withdraw` and `redeem` functions on behalf of the asset/share owner and deposit/withdraw the assets.

    - In the `TruStakeMATICv2` contract, only the owner of the tokens/shares can call these functions.

- The standard also stipulates that the `withdraw` and `redeem` functions are the functions in which assets are transferred to the recipient. If an implementation requires pre-requesting to the vault before a withdrawal can be performed then those methods should be performed separately.

    - In `TruStakeMATICv2` contract, the `withdraw` and `redeem` functions are used to unstake MATIC from the validator. The actual transfer happens by calling the `withdrawClaim` function after 80 checkpoints.

Contracts that integrate with the `TruStakeMATICv2` vault may wrongly assume that the functions are EIP-4626 compliant, which can cause integration problems in the future, potentially leading to a wide range of issues for both parties, including loss of funds.

Consider making all functions ERC-4626 compliant to prevent any integration issues.

Update: *Partially resolved in pull request #1 at commit 4514cfb. Functions that are not compliant with the ERC-4626 standard are documented.*

Notes & Additional Information

N-01 Commented-out Code

Throughout the codebase, there are lines of code that have been commented out with `//`. This can lead to confusion and is detrimental to overall code readability. We have provided a non-exhaustive list of examples below:

- Lines 8, 17 and 42 of `MasterWhitelist.sol`.

Consider removing any unneeded commented-out lines of code.

Update: *Resolved in pull request #1 at commit 75d4221.*

N-02 Gas Inefficiencies

There are several places across the codebase where changes can be made to improve gas consumption. For example:

- Several `for` loops assign the starting index to a default value, which is unnecessary. See `initialize` and `claimList`.
- The `getDistributors` and `getRecipients` functions can be removed to reduce deployment gas cost, as well as code size. Consider using the default `distributors()` and `recipients()` getters instead.
- In `addUserToWhitelistUsingPassport`, consider swapping the calls to `hasPassport` and `isUserBlacklisted`. On average, prioritizing `require` statements that do not involve external calls would return more gas to the user in case the method reverts.
- `_countryBlacklist` is read-only, and can be declared as `calldata` to save gas.
- In `deallocate` function, consider moving the call to `sharePrice` after the `revert` statements, as the share price is only needed if the initial checks pass.
- It is recommended that if a storage variable will be read multiple times in the same function, a copy to memory should first be created since reading from storage is expensive. Here are a few of the places where this change could decrease gas cost: `allocate`, `deallocate`, `distributeAll`.

When performing these changes, aim to reach an optimal tradeoff between gas optimization and readability. Having a codebase that is easy to understand reduces the chance of errors in the future and improves transparency for the community.

Update: *Resolved in pull request #1 at commits fbfcc80 and 62fbe7b.*

N-03 Unused Imports

In the codebase, there is an import that is unused and could be removed:

- Import `ERC20Upgradeable` of `TruStakeMATICv2.sol`.

Consider removing the unused import to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #1 at commit 7a93a9c.*

N-04 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as the function's output. They are an alternative to explicit in-line `return` statements.

In `TruStakeMATICv2.sol`, there are multiple instances of unused named return variables. For instance:

- The `unbondNonce` return variable in the `_unbond` function.
- The `shares` return variable in the `_convertToShares` function.

Consider either using or removing any unused named return variables.

Update: *Resolved in pull request #1 at commit f861037.*

N-05 Non-explicit Imports are Used

The use of non-explicit imports can decrease the code's clarity, and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple definitions exist within the same Solidity file.

Several global imports are being used, such as:

- Line 22 of `TruStakeMATICv2.sol`.
- Line 5 of `TruStakeMATICv2Storage.sol`.

Following the principle that clearer code is better code, consider using named import syntax (import {A, B, C} from "X") everywhere.

Update: *Resolved in pull request #1 at commit 84c7f3c.*

N-06 State Variable Visibility Not Explicitly Declared

Within `MasterWhitelist.sol` there are multiple state variables that lack an explicitly declared visibility. For instance:

- The state variables initialized from line 53 to line 135.

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #1 at commit 9bed3dc.*

N-07 Lack of indexed event parameters

Throughout the codebase, several events do not have their parameters indexed. For instance:

- Line 180 of `MasterWhitelist.sol`
- Line 10, line 77, line 172, line 174, line 176, line 178, line 180, line 182, line 184, and line 186 of `ITruStakeMATICv2.sol`

Consider indexing event parameters to improve the ability of off-chain services to search and filter for specific events.

Update: *Resolved in pull request #1 at commits 5c9bba9 and a82a051.*

N-08 Typographical Errors

Consider addressing the following typographical errors.

In `TruStakeMATICv2Storage.sol`:

- On line 33 "*phi*" should be "*distPhi*".
- On line 43 "*fudns*" should be "*funds*".

In `MasterWhitelist.sol`:

- On line 166 "*the address added to blacklisted*" should be "*the address added to the blacklist*".
- On line 564 "users *adds* themselves" should be "users *add* themselves".

Update: *Resolved in pull request #1 at commit 2406960.*

N-09 Inconsistent Coding Style

Throughout the codebase, there are several places that have inconsistent code style:

- `TruStakeMATICv2.sol` and `TruStakeMATICv2Storage.sol` write NatSpec using single-line comments, while other files use block comments.
- `removeSwapManagerFromWhitelist` and `removeUserFromBlacklist` assign default values instead of using `delete`.
- Consider moving the `INF_TIME` constant above `investigation_period`, for better visibility (same for `PROV_CODE_MANUAL` and `CODE_COUNTRY`).
- The imports in `MasterWhitelist.sol` are in the order: vendor contracts, Trufin contracts. However, in `TruStakeMATICv2.sol` they are in the opposite order. Consider having the same order for imports to improve the codebase's readability. The imports should also be grouped with a space in between.
- IMasterWhitelist is under the Polygon imports section, not Trufin.
- `++i` is used consistently across the codebase, except for inside `claimList`.
- `investigation_period`, `investigation_time`, `KYCPassport`, `KYCReader` and `KYCRegistry` should use *camelCase*.
- Several comments have no spacing between `//` and the comments' text. Here are some examples: #1, #2, #3.

Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with the help of linter tools such as Solhint is recommended.

Update: *Resolved in pull request #1 at commits 2da2114, 1989b80 and 0b75e11.*

N-10 Naming Suggestions

Some events in `MasterWhitelist.sol` can benefit from better naming. Specifically:

- `UserAddedToWhitelist` should be `addedToWhitelist`.
- `userRemovedFromWhitelist` should be `removedFromWhitelist`.

As these events are used for logging data for all types of roles and not just the `user` role, considering renaming these events to improve the clarity of the codebase.

Update: *Resolved in pull request #1 at commit 50a80ff.*

N-11 Unused Variable

`MasterWhitelist` uses `Quadrata QuadReader` to query a user's KYC verification attributes. However, it also stores a pointer to a `QuadPassport` contract, which is unused.

Moreover, Quadrata's documentation states that `QuadPassport` **contains the logic for onboarding/minting Quadrata Passport**, which is not the responsibility of the `MasterWhitelist`.

Consider removing the `KYCPassport` storage variable to improve the codebase's quality and reduce gas consumption.

Update: *Resolved in pull request #1 at commit 8de5818.*

N-12 Incorrect or Misleading Documentation

- On line 204 of the `MasterWhitelist` contract, the Natspec comment could be better rephrased as "_countryBlacklist is an array of the keccak256 hashes of 2 letter country codes".
- In the `Types` contract, the Natspec comments at the end of the file are misplaced and incomplete.
- On lines 7, 50 and 144 of the `MasterWhitelist` contract, the Natspec comments omit mentioning the `swapManager` user type.
- On line 97 of the `MasterWhitelist` contract, the Natspec comment could be better rephrased as "Mapping of users to the kycProvider used for verification", as it is not the `kycProvider` that whitelists, but a lawyer or the users themselves.
- On line 36 of the `TruStakeMATICv2Storage` contract, the Natspec comments read "cap on deposits into the vault" but it is actually the cap on total amount staked with the validator, which is different because the latter also counts MATIC that was rewarded and restaked.

- There were instances in this codebase where function parameters appear in a function signature to comply with the ERC-4626 but are never used within their respective function.
    - The `address` parameter of the `maxDeposit` function
    - The `address` parameter of the `maxMint` function
    - To improve clarity, consider documenting the reason for having these unused function parameters.

Consider correcting the missing or misleading documentation.

Update: *Resolved in pull request #1 at commit d08ab44.*

Conclusions

One high-severity issue was identified. Several changes were proposed to follow best practices and reduce the potential attack surface. The codebase was well-written and thoroughly documented. The Trufin team was very responsive and provided the auditors with extensive documentation about the project.

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Trufin team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

Privileged Entities and Roles

Critical: There are numerous privileged actions with serious security implications as described in detail in the Privileged Roles and Trust Assumptions sections of this report. Consider monitoring the trigger of admin functions to ensure all changes are expected. This should help the team remain vigilant against malicious actors.

Low: It would be useful to know when users whitelist themselves using third-party KYC verification, as it would increase awareness of new parties having access without facilitation by lawyers. It would also be indicative of protocol adoption.

Technical

Critical: The protocol relies on multiple third-party smart contracts that are behind upgradeable proxies. Consider monitoring such upgrades, to be notified early if the dependencies are no longer backwards-compatible and introduce bugs.

Low: It would be useful to know when the total amount of staked `MATIC` is close or has reached the maximum cap. This could indicate the need to increase the cap preemptively if needed, making sure user deposits do not revert.