Security Review Report
NM-0158 TruFin

NETHERMIND
SECURITY

(Dec 21, 2023)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind for TruFin's TruStake vault. TruStake provides access to MATIC staking on Ethereum for institutions featuring auto-compounding of rewards and the ability to allocate rewards to other addresses. The protocol is permissioned, ensuring that each user has passed know-your-customer (KYC) checks and that only whitelisted users can stake. A user's stake is represented by ERC20 tokens, which track the increase in value from the accrual of rewards. The protocol supports multiple validators. Users can choose to stake into any supported validator or use a default validator specified by the TruFin team.

The reviewed code represents an upgrade of the existing TruStake vault, encompassing several changes. Notably, the protocol introduces support for staking across multiple validators, where users can select a specific validator for staking and unstaking or opt for TruStake's default validator. Moreover, the vault exclusively operates in loose mode, where the distributor isn't obliged to maintain funds to cover their allocations. This allows the removal of allocations at any time without distributing the accrued rewards.

During the re-audit phase, the TruFin team implemented the concept of private validators as a preventive measure against potential validators' manipulation within TruStake. If a user is designated as a private user, he's restricted from interacting exclusively with a specified private validator. Meanwhile, non-private users are prohibited from staking or unstaking within private validators.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contracts, and (d) creation of test cases. Along this document, we report 13 points of attention, where 1 is classified as `Medium`, 5 are classified as `Low`, and 7 are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation and test output. Section 9 concludes the document.

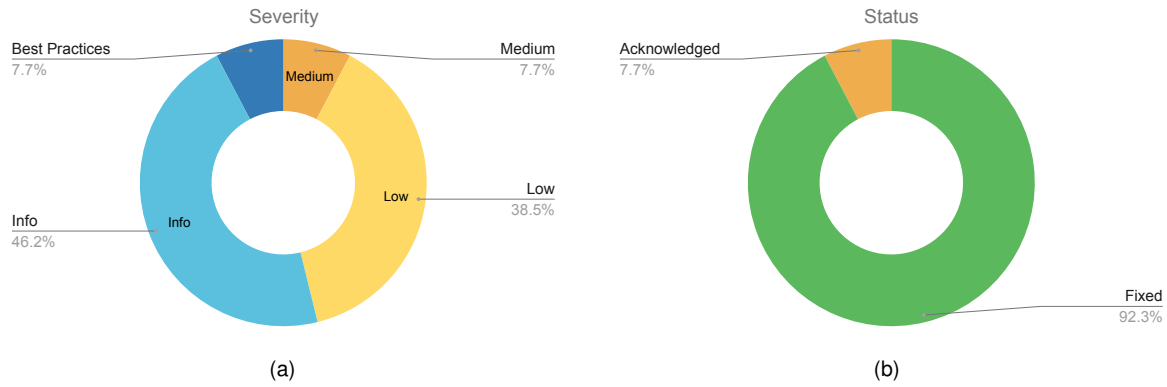

Fig 1: Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (5), **Undetermined** (0), **Informational** (6), **Best Practices** (1). **Distribution of status: Fixed** (12), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Dec 12, 2023
Response from Client	Dec 13, 2023
Final Report	Dec 21, 2023
Methods	Manual Review, Automated Analysis
Repository	https://github.com/TruFin-io/staking-contracts/
Commit Hash (Initial Audit)	4fa26ced4d2d045260720800aba78e94056f991b
Branch (Review Audit)	6935d66d049f6a8177e0d3256827a48d153d0ae1
Documentation	Provided protocol documentation (PDFs)
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	contracts/matic-staker/contracts/test/MaliciousValidator_Withdraw.sol	34	5	14.7%	21	60
2	contracts/matic-staker/contracts/test/MaliciousValidator_Deposit.sol	35	5	14.3%	22	62
3	contracts/matic-staker/contracts/main/ERC4626Storage.sol	7	4	57.1%	3	14
4	contracts/matic-staker/contracts/main/TruStakeMATICv2.sol	543	239	44.0%	162	944
5	contracts/matic-staker/contracts/main/Types.sol	20	17	85.0%	5	42
6	contracts/matic-staker/contracts/main/TruStakeMATICv2Storage.sol	24	27	112.5%	22	73
7	contracts/matic-staker/contracts/interfaces/IValidatorShare.sol	16	6	37.5%	14	36
8	contracts/matic-staker/contracts/interfaces/IStakeManager.sol	47	2	4.3%	21	70
9	contracts/matic-staker/contracts/interfaces/ITruStakeMATICv2.sol	90	85	94.4%	47	222
10	contracts/matic-staker/contracts/interfaces/IMasterWhitelist.sol	22	28	127.3%	15	65
11	contracts/whitelist/contracts/main/MasterWhitelist.sol	58	32	55.2%	25	115
12	contracts/whitelist/contracts/interfaces/IMasterWhitelist.sol	22	28	127.3%	15	65
	Total	**918**	**478**	**52.1%**	**372**	**1768**

3 Summary of Issues

	Finding	Severity	Update
1	Incorrect computation of treasury shares in `_deposit(...)` and `_withdrawRequest(...)` functions	Medium	Fixed
2	Drop in share price after the upgrade can be exploited if the default validator is not added	Low	Acknowledged
3	Fees from unstaked rewards are taken before adjusting share price	Low	Fixed
4	Possible inconsistency between accounted and actual staked amount in `_stake(...)` function	Low	Fixed
5	The function `compoundRewards(...)` optimistically mints shares	Low	Fixed
6	Withdrawn amount can be different than what is expected	Low	Fixed
7	Different rounding direction between function logic and emitted event	Info	Fixed
8	Incorrect Natspec documentation	Info	Fixed
9	Redundant check in `compoundRewards(...)` function	Info	Fixed
10	Removing owner address as agent indicates success when removal has failed	Info	Fixed
11	Unnecessary event emission for users with no allocations in `distributeAll(...)` function	Info	Fixed
12	Users can distribute their rewards after being removed from the whitelist	Info	Fixed
13	Hardcoded values in the codebase	Best Practices	Fixed

4 Protocol Overview

TruFin protocol has the following main components:

4.1 Master Whitelist

The `MasterWhitelist` contract is used by the TruStake vault, ensuring that only whitelisted users can interact with the vault. It defines two allowance levels:

- Agents: Responsible for adding and removing users from the whitelist and updating their statuses. The contract owner is considered an agent.
- Users: A user has one of the following statuses in the whitelist: None, Whitelisted or Blacklisted. The contract provides methods for agents to whitelist, blacklist or clear a user's status.

4.2 TruStake vault

Staking

The staking process is initiated by depositing MATIC into the vault, resulting in minting an equivalent amount of TruMatic as shares back to the user. The deposited MATIC is directly staked within the validator. Deposits are possible through the `deposit(uint256 _assets)` and `depositToSpecificValidator(uint256 _assets, address _validator)` functions, where the former uses the default validator set in the contract and the latter allows the user to designate one of the supported validators.

Notably, the deposit function allows zero-amount and zero-address calls, enabling the staking of available MATIC in the vault into the validator.

Unstaking

The unstaking process involves two phases:

- The user initiates a withdrawal request for a specific amount of MATIC, resulting in the vault burning the equivalent TruMatic shares and triggering an unbonding process on the validator. Similar to deposits, users can withdraw from a designated validator or use TruStake's default validator, using `withdraw(uint256 _assets)` and `withdrawFromSpecificValidator(uint256 _assets, address _validator)` functions respectively.
- Upon completion of the withdrawal delay, users can claim their funds via the `withdrawClaim(uint256 _unbondNonce, address _validator)` function by providing the unbonding nonce that is generated in the request phase.

Auto-compounding

The vault provides the `compoundRewards(...)` function that is periodically called to restake rewards across different validators and stake any claimed rewards and available MATIC in the vault. The treasury takes a fee for any restaked or staked rewards.

Rewards allocation and distribution

Users can allocate rewards from their staked amounts to specific recipients with the rewards allocation feature. The allocation process involves keeping a record of the recipient's address, the amount of MATIC whose rewards are allocated to that recipient, and the current TruMatic share price. The user is not obligated to distribute the rewards or maintain enough funds in the vault to cover his allocations. Users can distribute rewards to their list of recipients in MATIC or TruMatic, transferring the amount directly from their wallet to the recipient's wallet. Moreover, they can deallocate or cancel previous allocations without distributing rewards. Rewards are computed based on the share price increase between allocation and distribution times.

Fees

The TruStake vault implements two distinct fees in the form of percentages:

- The rewards fee, denoted by `phi`, is deducted from claimed rewards acquired from validators. This fee is applicable during deposit, withdrawal, and compounding processes as these flows involve claiming rewards from validators.
- The distribution fee, denoted by `distPhi`, is taken with each reward distribution as a percentage of the distributed amount.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Medium] Incorrect computation of treasury shares in `_deposit(...)` and `_withdrawRequest(...)` functions

File(s): TruStakeMATICv2.sol

Description: The `_deposit(...)` function handles a user's staking of a specified amount of MATIC into a given validator. Accrued rewards on the validator are implicitly claimed when staking and transferred to the contract. The function mints the corresponding shares for the user while also collecting a fee for the treasury by minting a `phi` percentage of the claimed rewards.

However, it presents an issue in the computation of the shares intended for the treasury, where `shareIncreaseTsy` is computed based on `totalRewards()`, which represents the total rewards obtained across all validators, not just the provided `_validator`. Consequently, the function mints an excessive number of shares to the treasury, resulting in an overestimation of shares compared to the claimed rewards of the specified validator.

```
function _deposit(address _user, uint256 _amount, address _validator) private returns (uint256) {
    // ...
    // @audit `shareIncreaseTsy` computed based on the total rewards across all validators
    uint256 shareIncreaseTsy = (totalRewards() * phi * 1e18 * globalPriceDenom) / (globalPriceNum * PHI_PRECISION);
    // ...
    _mint(treasuryAddress, shareIncreaseTsy);
    // ...
    // @audit only rewards of `_validator` are claimed
    _stake(stakeAmount, _validator);
    // claimed rewards increase here as liquid rewards on validator share contract
    // are set to zero rewards and transferred to this vault

    // ...
}
```

To illustrate, let's assume the system has four validators with 25 MATIC in liquid rewards each and a fee percentage of 5%. When a user stakes on `validator_1`, the function computes the shares as a percentage of the total rewards (100 MATIC) across all validators, while the claimed rewards are 25 MATIC of `validator_1`.

```
_deposit( ... , validator_1):
     - shares minted = convertToShares( 100 MATIC * 5% )
     - claimed rewards = 25 MATIC
```

The same function can be repeatedly called for the same validator, minting additional shares even when the provided validator has no additional claimable rewards.

```
_deposit( ... , validator_1):
     - shares minted = convertToShares( 75 MATIC * 5% )
     - claimed rewards = 0 MATIC
```

The same issue is present in `_withdrawRequest(...)` function, where fees are taken from the rewards of all validators instead of the specified one.

Recommendation(s): Consider computing the treasury fee shares based on the claimed rewards from the provided validator instead of all validators.

Status: Fixed

Update from the client: Fixed with commit 5c28bfabf0270c6e0499f67728a6fbcf099d0516

6.2 [Low] Drop in share price after the upgrade can be exploited if the default validator is not added

File(s): `TruStakeMATICv2.sol`, `TruStakeMATICv2Storage.sol`

Description: The default validator inherits its value from the previous contract's validator within the upgrade process since they share the same storage slot. However, before adding that validator to the list of validators, called `validatorAddresses`, the contract doesn't consider the amount of MATIC staked and rewards earned on that validator. This happens because the `totalRewards(...)` and `totalStaked(...)` functions go through the `validatorAddresses` list to calculate the total amounts.

While these amounts are not considered in the vault, the corresponding shares are in circulation and part of the total supply of TruMatic. Consequently, after the upgrade, the share price drops incorrectly. Users can exploit this by making a deposit or an allocation using the erroneous share price. For instance, the user can make a deposit and mint a large number of shares. Later, when the default validator is added, the share price will return to the correct value, allowing the user to burn those shares for a larger amount of MATIC.

```
function sharePrice() public view returns (uint256, uint256) {
    if (totalSupply() == 0) return (1e18, 1);
    // @audit after upgrade, `totalStaked()` and `totalRewards()` do not include amounts from the default validator
    uint256 totalCapitalTimesPhiPrecision = (totalStaked() + totalAssets()) *
        PHI_PRECISION +
        (PHI_PRECISION - phi) *
        totalRewards();
    // @audit `totalSupply()` includes shares minted for the default validator staked amounts and accrued rewards
    return (totalCapitalTimesPhiPrecision * 1e18, totalSupply() * PHI_PRECISION);
}
```

Recommendation(s): Ensure the default validator is set in the contract within the upgrade process to not allow users to utilize the incorrect share price.

Status: Acknowledged

Update from the client: We've added setting up the default validator on initialise for completeness, but for the upgrade, we'll batch the transaction upgrade with a transaction that sets the validator properly (using Safe batched transactions).
Fixed with commit dec35f051c7ff86345648b3efac6353c73caddc6

6.3 [Low] Fees from unstaked rewards are taken before adjusting share price

File(s): TruStakeMATICv2.sol

Description: The `compoundRewards(...)` function handles the restaking of unclaimed rewards across all validators. It calculates the number of shares to be minted for the treasury as fees for restaked rewards. These shares are calculated before the actual restaking process and minted at the end of the function.

Once the restaking process occurs via the `_restake()` function, the code checks if the total asset balance exceeds zero. If so, these assets are staked using the `_deposit(...)` function. During this phase, if the specified `_validator` possesses remaining unstaked rewards, they are claimed, and a fee is minted to the treasury.

At this stage in the code, the rewards have been restaked, but the restaking treasury shares have not been minted yet. Consequently, the share price used within the `_deposit(...)` function does not account for these unminted shares, leading to a slightly higher share price used to compute the treasury fees.

```
function compoundRewards(address _validator) external nonReentrant {
   // ...
   // @audit share price after restaking doesn't account for the treasury shares yet
    _restake();

    if (totalAssetBalance > 0) {
       // ...
       // @audit if `_validator ` has unstaked rewards, they will be claimed, and treasury fees are computed using the
        ↪  non-adjusted share price
        _deposit(address(0), 0, _validator);
    }
    //@audit treasury shares are minted to adjuste the share price
    _mint(treasuryAddress, shareIncrease);
   // ...

}
```

Recommendation(s): Consider moving the `_mint(...)` call before the deposit happens to ensure the share price is updated before any share computation occurs.

Status: Fixed

Update from the client: Fixed with commit 5276d208fb9c459809434e1056946f858171d7a8

6.4 [Low] Possible inconsistency between accounted and actual staked amount in `_stake(...)` function

File(s): TruStakeMATICv2.sol

Description: The `_stake(...)` function is responsible for staking the provided amount into a particular validator. It increases the validator's staked amount by the provided `_amount`. Staking into the validator occurs within the `buyVoucher(...)` function. This function takes the amount to be staked and a minimum share amount to be minted, returning the actual staked amount.

```
function _stake(uint256 _amount, address _validator) private {
    validators[_validator].stakedAmount += _amount;
    //@audit `buyVoucher(...)` returns the actual staked amount
    IValidatorShare(_validator).buyVoucher(_amount, _amount);
}
```

However, the validator's staked amount is updated using the input parameter `_amount`, ignoring the returned value from the `buyVoucher(...)` call. The latter function implementation states the possibility of the actual staked amount being less than expected. This creates a potential inconsistency between the accounted amount and the actual staked amount. Any difference between the two amounts will be accounted for twice within the vault, as a staked amount, and within the vault's total assets.

Recommendation(s): We recommend updating `stakedAmount` using the returned value from the `buyVoucher(...)` call to avoid potential inconsistencies.

Status: Fixed

Update from the client: Fixed with commit 90ee9734ddbabe929ce3f3219c48b5ec70f6bf7c

6.5 [Low] The function `compoundRewards(...)` optimistically mints shares

File(s): `TruStakeMATICv2.sol`

Description: The function `compoundRewards(...)` calculates the amount of shares to mint by calling `totalRewards(...)`. Assuming `totalRewards(...)` is fixed according to the previously mentioned issues in this report, and it only considers enabled validators, it is still possible for an incorrect amount of shares to be minted.

The amount of shares to be minted is calculated before the call to `_restake(...)` where the restaking occurs. In `_restake(...)`, if an attempt to restake on a validator fails, the revert is caught, and then an error is emitted. This means that if a validator happens to revert, the reward amount is not restaked, even though in the calling function `compoundRewards(...)`, the mint amount has already been calculated assuming that it will succeed.

This could be exploited by the treasury owner in a hard-to-detect way by taking advantage of revert conditions inside the validators, such as when a validator is behaving incorrectly or when the minimum amount of rewards in order to successfully claim is not reached. The `compoundRewards(...)` function could be called on loop, where no rewards are actually claimed, but shares are still minted for the treasury, increasing treasury value at the cost of the value of the user shares.

Recommendation(s): Consider updating the computation of shares to account for the actual staked amount in `_restake(...)`, rather than relying on the optimistic reward amount in `totalRewards(...)`.

Status: Fixed

Update from the client: Fixed with commit c83fc467029c1856752853c1116864c43e22e757 and commit d0db3998d0af7cb0b0d6bd3eca0b2fd0d7c70cc6

6.6 [Low] Withdrawn amount can be different than what is expected

File(s): `TruStakeMATICv2.sol`

Description: The withdrawal process within the staker contract happens in two phases. First, users request a withdrawal of a specified amount of MATIC, and an unbonding process is started on the validator. Once the withdrawal delay of 80 epochs elapses, users can claim their MATIC by invoking `withdrawClaim(...)` function and providing the unbound nonce received during the withdrawal request.

On the validator side, the unbonding involves converting the specified amount of MATIC into shares and storing it with a unique nonce. During the claim using the nonce, the amount of shares is retrieved from storage and then converted to MATIC using the current rate. This amount is transferred to the staker vault.

While the staker vault transfers the total requested amount to the user during the claim process, changes in the share price within the validator can lead to inconsistencies. If the actual withdrawn amount is lower than expected, the vault will implicitly cover the difference from its balance if sufficient, or the transfer will revert if it cannot.

```
function _withdrawClaim(uint256 _unbondNonce, address _validator) private {
    Withdrawal memory withdrawal = withdrawals[_validator][_unbondNonce];
    // ...

    // @audit claiming from the validator can return an amount different than expected.
    _claimStake(_unbondNonce, _validator);

    // transfer claimed MATIC to claimer
    // @audit the exact amount is transferred to the claimer regardless of the claimed amount
    IERC20Upgradeable(stakingTokenAddress).safeTransfer(msg.sender, withdrawal.amount);

    emit WithdrawalClaimed(msg.sender, _validator, _unbondNonce, withdrawal.amount);
}
```

Recommendation(s): Revisit the withdrawal process to handle potential differences in the withdrawn amount caused by changes in the validator's share price.

Status: Fixed

Update from the client: Fixed with commit c0390941fdf1ad3662538b6d1405c577e59985fa

6.7 [Info] Different rounding direction between function logic and emitted event

File(s): TruStakeMATICv2.sol

Description: The `_distributeRewards(...)` function distributes rewards to a specified recipient. The rewards are transferred in TruMatic or MATIC, based on the boolean input `_inMatic`. When rewards are in MATIC, the function converts the shares to be transferred `sharesToMove` to the corresponding amount of MATIC by using the `previewRedeem(...)` function.

However, the event emitted by this function converts the shares to MATIC differently, using the `convertToAssets(...)` function. The difference between the two is that the former rounds the result up, while the latter rounds down. This can result in a slight difference between the amount reported by the event and the actual transferred amount.

```
function _distributeRewards(...) private {
    // ...
    if (_inMatic) {
        uint256 maticAmount = previewRedeem(sharesToMove);
         // @audit transferred amount of MATIC computed using `previewRedeem(...)`, which rounds up
        IERC20Upgradeable(stakingTokenAddress).safeTransferFrom(_distributor, _recipient, maticAmount);
    } else {
        _transfer(_distributor, _recipient, sharesToMove);
    }

   // ...

    emit DistributedRewards(
        _distributor,
        _recipient,
       // @audit amount of MATIC computed using `convertToAssets(...)`, which rounds down
        convertToAssets(sharesToMove),
        sharesToMove,
        globalPriceNum,
        globalPriceDenom
    );
}
```

Recommendation(s): To ensure accuracy in the event data, consider updating the event to use the same function to compute the transferred MATIC amount.

Status: Fixed

Update from the client: Fixed with commit ee63fdae17d1575ed30d123d43caaa109abbc372

6.8 [Info] Incorrect Natspec documentation

File(s): TruStakeMATICv2.sol

Description: The Natspec comment for the `_unbond(...)` function states that it is used to unstake from the default validator. However, the function takes the `_validator` address as a parameter, allowing unstaking from that specific validator.

```
/// @notice Requests to unstake a certain amount of MATIC from the default validator.
// ...
function _unbond(uint256 _amount, address _validator) private returns (uint256) {...}
```

Recommendation(s): Update the function's NatSpec comment to align with the current implementation.

Status: Fixed

Update from the client: Fixed with commit 3c35f4091d08908146e30c4a71268085d487d7bf

6.9 [Info] Redundant check in the function `compoundRewards(...)`

File(s): `TruStakeMATICv2.sol`

Description: The `compoundRewards(...)` function performs a validation check for the validator state, ensuring that the validator is enabled before depositing. However, a similar state check is also present within the subsequently called `_deposit(...)` function, resulting in an unnecessary check duplication.

```
function compoundRewards(...) external nonReentrant {
    // ...
    if (totalAssetBalance > 0) {
        // @audit redundant check from `_deposit(...)`
        if (validators[_validator].state != ValidatorState.ENABLED) {
            revert ValidatorNotEnabled();
        }
        _deposit(address(0), 0, _validator);
    }
    // ...
}
```

Recommendation(s): Remove the duplicated check within `compoundRewards(...)` function.

Status: Fixed

Update from the client: Fixed with commit 850ba7baa76a22a6e2cd34511d2115638ce5c6fb

6.10 [Info] Removing owner address as agent indicates success when removal has failed

File(s): `MasterWhitelist.sol`

Description: The function `removeAgent(...)` is callable by any agent and sets the boolean in the `agents` mapping for a given address to false, removing agent privileges for the address. If the function is called with the argument `_agent` set to the owner address, the associated boolean in the `agents` mapping will be set to false, and an event will be emitted indicating they have been removed successfully. However, this address will still be considered an agent due to the check in `isAgent`, which allows the owner to be considered an agent.

Recommendation(s): Consider adding a check if the `_agent` argument in `removeAgent(...)` is equal to the owner address. If they are equal, consider reverting and instead use the built-in features of the OpenZeppelin `OwnableUpgradeable` implementation to change ownership to a different address.

Status: Fixed

Update from the client: Fixed with commit 4ecd4c7ef9e7cbebb3d92a26c99435b4102a69aa

6.11 [Info] Unnecessary event emission for users with no allocations in `distributeAll(...)` function

File(s): `TruStakeMATICv2.sol`

Description: The `distributeAll(...)` function distributes allocated rewards from the caller to their list of recipients. However, the function lacks validation for the presence of allocations for that caller. When the recipients' list is empty, the function will skip the loop and unnecessarily emit the `DistributedAll` event. This can be misleading when listening to the contract's events.

```
function distributeAll(bool _inMatic) external nonReentrant {
    address[] storage rec = recipients[msg.sender][false];
    uint256 recipientsCount = rec.length;
    (uint256 globalPriceNum, uint256 globalPriceDenom) = sharePrice();
     // @audit loop is skipped when user has no recipients
    for (uint256 i; i < recipientsCount; ) {
       // ...
    }
    // @audit an event is emitted for a user with no allocations
    emit DistributedAll(msg.sender);
}
```

Recommendation(s): Consider implementing a check within the function to verify that the caller's recipients list is not empty before proceeding with rewards distribution.

Status: Fixed

Update from the client: Fixed with commit cb86e651d0d29f9ce51df31f6eeb812c9053d6ed

6.12 [Info] Users can distribute their rewards after being removed from the whitelist

File(s): `TruStakeMATICv2.sol`

Description: When a user is removed from the whitelist, he can no longer withdraw from the vault. However, he can distribute his accrued rewards if he made a previous allocation. We can have a scenario where a user allocates 100% of rewards to a second address he controls. If that user is removed from the whitelist due to malicious behavior, he can still claim rewards by distributing them to his second address. That is possible because both `distributeAll(...)` and `distributeRewards(...)` functions don't require the caller to be whitelisted.

Recommendation(s): We recommend revisiting if this aligns with the intended behavior of the protocol. If not, consider restricting access to these functions only to whitelisted users.

Status: Fixed

Update from the client: Fixed with commit cee1a4557c2fee205ebaa7dd1981d1be1dcc0876

6.13 [Best Practices] Hardcoded values in the codebase

File(s): `TruStakeMATICv2.sol`

Description: Within the `TruStakeMATICv2` contract, hardcoded values such as `1e18` and `1e22` are utilized. These values lack explicit descriptions, making their purpose and significance unclear. This increases code complexity and the likelihood of errors, especially when updating them.

Recommendation(s): To enhance code clarity and reduce the risk of inconsistencies, consider replacing hardcoded values with constants with descriptive names.

Status: Fixed

Update from the client: We didn't replace all instances where 1e18 and 1e22 figured but only where doing so would make the code easier to read. Fixed with commit 901685f57b1554d416907000ef7832c522cb5e83

7 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about TruFin documentation

The `TruFin` team has provided extensive `documentation` on their GitHub repo, including technical insights into deposit and withdraw flows, precision and rounding choices, and rewards allocation and distribution. Each function is complemented by NatSpec comments describing its behavior, inputs, and outputs. Moreover, the team conducted a comprehensive code walkthrough and maintained open communication to address any inquiries or concerns raised by the Nethermind auditors.

8 Test Suite Evaluation

8.1 Compilation Output

```
> npx hardhat compile

Downloading compiler 0.8.14
Generating typings for: 21 artifacts in dir: typechain-types for target: ethers-v5
Successfully generated 74 typings!
Compiled 22 Solidity files successfully (evm target: paris).
```

8.2 Tests Output

```
dev@dev:~/dev/nm/synnax/gauss/NM-0158/staking-contracts$ npm test

> smart-contracts@1.0.0 test
> lerna run test --stream --concurrency 1 --no-bail

lerna notice cli v7.4.2

 >  Lerna (powered by Nx)   Running target test for 2 projects:

    - matic-staker
    - whitelist

 ———————————————————————————————————————————————————————————————————————————————————————————————

> matic-staker:test

matic-staker: > matic-staker@1.0.0 test
matic-staker: > npx hardhat test
matic-staker:
matic-staker:   DEALLOCATE
matic-staker:     Individual Allocation State
matic-staker:       Individual allocation price is not changed during deallocation
matic-staker:       Reduces individual allocation by deallocated amount
matic-staker:       Deletes individual allocation from storage if full individual deallocation
matic-staker:     Total Allocation State
matic-staker:       Returns zero for total deallocation
matic-staker:       Decreases total allocation amount if partial deallocation
matic-staker:     Functionality
matic-staker:       Deallocate reduces rewards proportionally (4420ms)
matic-staker:       Deallocation leads to rewards if the reduced amount was allocated initially (before any
 ↪  distribution) (4220ms)
matic-staker:       Flow of allocating, deallocating and distributing as rewards accrue (16134ms)
matic-staker:   Staker
matic-staker:     Inflation attack check
matic-staker:       Basic inflation attack (223ms)
matic-staker:   Inflation Attack
matic-staker:     Checks
matic-staker:       Check Inflation not possible with user leaving tiny remaining balance (3166ms)
matic-staker:   Reentrancy Attack
matic-staker:     Attempt to reenter nonReentrant functions
matic-staker:       Deposit reentrancy attack reverts (1775ms)
matic-staker:       Withdraw request reentrancy attack reverts (622ms)
matic-staker:   Staker
matic-staker:     Scenario 1: transfer of allocated shares
matic-staker:       Scenario (2036ms)
matic-staker:   Scenario -- Check storage after allocate/deallocate
matic-staker:     Flow
matic-staker:       Deposit as user1, deployer (1864ms)
matic-staker:       Allocate user1 -> user2 (85ms)
matic-staker:       Allocate user1 -> deployer (83ms)
matic-staker:       Deallocate half user1 -> user2 (54ms)
matic-staker:       Deallocate last half user1 -> user2
matic-staker:       Deallocate deployer
matic-staker:       Allocate user1 -> deployer again (77ms)
```

```
matic-staker:   Staker
matic-staker:     Setters
matic-staker:        Set whitelist
matic-staker:        Set treasury
matic-staker:        Set phi
matic-staker:        Set dist phi
matic-staker:     Main
matic-staker:        Deposit (119ms)
matic-staker:        Withdraw (1059ms)
matic-staker:     Allocations
matic-staker:        Allocate to user (317ms)
matic-staker:        Deallocate from user (174ms)
matic-staker:        Distribute rewards (370ms)
matic-staker:     Revert
matic-staker:        When try to initialize contract again
matic-staker:        When not owner tries to add a validator
matic-staker:        When not owner tries to set whitelist
matic-staker:        When not owner tries to set treasury
matic-staker:        When not owner tries to set phi
matic-staker:        When not owner tries to set dist phi
matic-staker:        When try to set phi if it is too large
matic-staker:        When try to set dist phi if it is too large
matic-staker:        When not whitelisted user tries to allocate
matic-staker:        When not whitelisted user tries to deallocate
matic-staker:        When try to allocate 0 amount
matic-staker:        When try to deallocate
matic-staker:        When not whitelisted user tries to deposit
matic-staker:        When not whitelisted user tries to withdraw
matic-staker:   INIT
matic-staker:     INITIALISATION
matic-staker:        global variables initialised with correct values
matic-staker:        reverts if phi is too large
matic-staker:        reverts if distPhi is too large
matic-staker:     MODIFIERS
matic-staker:        onlyWhitelist
matic-staker:     SETTERS - events and ownable
matic-staker:        setWhitelist
matic-staker:        setTreasury
matic-staker:        setEpsilon
matic-staker:        setPhi
matic-staker:        setDistPhi
matic-staker:        owner sets epsilon
matic-staker:        non-owner setting epsilon fails
matic-staker:     ATTACKS
matic-staker:        inflation frontrunning attack investigation (152ms)
matic-staker:        fail: depositing under 1 MATIC
matic-staker:        pass: deposit 1 MATIC or more (240ms)
matic-staker:   GETTERS
matic-staker:     Max functions
matic-staker:        maxWithdraw (169ms)
matic-staker:        pass: output of maxWithdraw is greater than to just deposited amount without accrual (114ms)
matic-staker:        pass: withdraw output of maxWithdraw after depositing (341ms)
matic-staker:        fail: cannot withdraw 1 + output of maxWithdraw after depositing (140ms)
matic-staker:        pass: withdraw output of maxWithdraw after depositing and accruing rewards (4382ms)
matic-staker:        fail: cannot withdraw 1 + output of maxWithdraw after depositing and accruing rewards (4231ms)
matic-staker:        preview functions circular check (20678ms)
matic-staker:     TruMATIC token: getters + metadata
matic-staker:        name
matic-staker:        symbol
matic-staker:     Validators
matic-staker:        getValidators
matic-staker:        get all validators, whether they are active, and the amount staked (3207ms)
matic-staker:   SETTERS
matic-staker:     setWhitelist
matic-staker:        Reverts with zero address
matic-staker:        Works with a new address
matic-staker:        Works with the same address
matic-staker:     setTreasury
matic-staker:        Reverts with zero address
matic-staker:        Works with a new address
matic-staker:        Works with the same address
```

```
matic-staker:     setPhi
matic-staker:        General input validation; reverts when too high
matic-staker:        Works with a new value
matic-staker:        Works with the same value
matic-staker:     setDistPhi
matic-staker:        General input validation; reverts when too high
matic-staker:        Works with a new value
matic-staker:        Works with the same value
matic-staker:     setEpsilon
matic-staker:        Reverts with too high value
matic-staker:        Works with a new value
matic-staker:        Works with the same value
matic-staker:     setMinDeposit
matic-staker:        Reverts with too low value
matic-staker:        Works with a new value
matic-staker:        Works with the same value
matic-staker:        Fails if set by non-owner
matic-staker:   Validators
matic-staker:     addValidator
matic-staker:        Adds a new validator
matic-staker:        Sets the amount staked on the validator
matic-staker:        Emits the expected event
matic-staker:        Reverts with zero address
matic-staker:        Reverts with an existing address
matic-staker:        Reverts when the caller is not the owner
matic-staker:     disableValidator
matic-staker:        Disable an enabled validator
matic-staker:        Emits the expected event
matic-staker:        Reverts with zero address
matic-staker:        Reverts with an unknown validator address
matic-staker:        Reverts with a disabled validator address
matic-staker:        Reverts when the caller is not the owner
matic-staker:     enableValidator
matic-staker:        Enable a disabled validator
matic-staker:        Emits the expected event
matic-staker:        Reverts with zero address
matic-staker:        Reverts with an unknown validator address
matic-staker:        Reverts with an enabled validator address
matic-staker:        Reverts when the caller is not the owner
matic-staker:     setDefaultValidator
matic-staker:        Sets a default validator (325ms)
matic-staker:        Emits the expected event
matic-staker:        Reverts when called by non-owner
matic-staker:        Reverts with zero address
matic-staker:        Reverts with a non-enabled validated (331ms)
matic-staker:   Other
matic-staker:     allocate
matic-staker:        Reverts with zero address (131ms)
matic-staker:   Deployment
matic-staker:     INITIALISATION
matic-staker:        Reverts on zero address (143ms)
matic-staker:   Checkpoint Submissions
matic-staker:      rewards are increasing with each checkpoint (24986ms)
matic-staker:   DEPOSIT
matic-staker:      single deposit (187ms)
matic-staker:      deposit returns the minted shares (117ms)
matic-staker:      single deposit to a specific validator (187ms)
matic-staker:      deposit to a specific validator returns the minted shares (120ms)
matic-staker:      single deposit with too little MATIC fails (52ms)
matic-staker:      single deposit to a non-existent validator fails
matic-staker:      single deposit to a deactivated validator fails
matic-staker:      repeated deposits (329ms)
matic-staker:      repeated deposits to specific validator (327ms)
matic-staker:      multiple account deposits (543ms)
matic-staker:      multiple account deposit to a specific validator (480ms)
matic-staker:      multiple account deposit to a specific and default validator (486ms)
matic-staker:      deposit zero MATIC (121ms)
matic-staker:      deposit zero MATIC to specific validator (119ms)
matic-staker:      Can withdraw maxWithdraw amount (31136ms)
matic-staker:      can immediately withdraw deposited amount (413ms)
matic-staker:      unknown non-whitelist user deposit fails
matic-staker:      unknown non-whitelist user cannot deposit to a whitelisted user's address
```

```
matic-staker:       user cannot drain vault by depositing zero (129ms)
matic-staker:       user cannot deposit less than the minDeposit
matic-staker:       user can deposit the minDeposit exactly (135ms)
matic-staker:       updates validator struct correctly post deposit (129ms)
matic-staker:       user can deposit to specific validator (387ms)
matic-staker:       user can deposit the minDeposit exactly to a specific validator (383ms)
matic-staker:       unknown non-whitelist user deposit to  specific validator fails (319ms)
matic-staker:       unknown non-whitelist user cannot deposit to specific validator to a whitelisted user's address
 ↪  (328ms)
matic-staker:       user cannot drain vault by depositing zero to a specific validator (393ms)
matic-staker:       user cannot deposit less than the minDeposit to  specific validator (339ms)
matic-staker:       deposit zero MATIC to a specific validator (371ms)
matic-staker:     WITHDRAW REQUEST
matic-staker:       initiate a partial withdrawal (365ms)
matic-staker:       withdraw returns the burned shares and unbond nonce (328ms)
matic-staker:       initiate a partial withdrawal from a specific validator (372ms)
matic-staker:       withdraw from a specific validator returns the burned shares and unbond nonce (319ms)
matic-staker:       initiate a complete withdrawal (353ms)
matic-staker:       initiate a complete withdrawal from a specific validator (351ms)
matic-staker:       initiate multiple partial withdrawals (517ms)
matic-staker:       initiate withdrawal with rewards wip (5860ms)
matic-staker:       try initiating a withdrawal of size zero
matic-staker:       try initiating withdrawal of more than deposited (194ms)
matic-staker:       try initiating withdrawal from a non existent validator (176ms)
matic-staker:       try initiating a withdrawal from a specific validator with a non-whitelisted user
matic-staker:     WITHDRAW CLAIM
matic-staker:       User: withdrawClaim
matic-staker:         try claiming withdrawal requested by different user (49ms)
matic-staker:         try claiming pre-upgrade withdrawal requested by different user (2987ms)
matic-staker:         try claiming withdrawal requested 79 epochs ago (77ms)
matic-staker:         try claiming withdrawal with unbond nonce that doesn't exist
matic-staker:         try claiming already claimed withdrawal (87ms)
matic-staker:         try claiming withdrawal as non-whitelisted user
matic-staker:         claim withdrawal requested 80 epochs ago with expected changes in state and balances (159ms)
matic-staker:         claim withdrawal requested from a specific validator 80 epochs ago (501ms)
matic-staker:         claim withdrawal from a different validator than was deposited into (2303ms)
matic-staker:         updates validator struct post withdrawal claim (80ms)
matic-staker:       User: claimList
matic-staker:         try to claim test unbonds when one has not matured (134ms)
matic-staker:         try to claim test unbonds as a non-whitelisted user
matic-staker:         try to claim test unbonds when one has already been claimed (89ms)
matic-staker:         try to claim test unbonds when one has a different user (125ms)
matic-staker:         claim three test unbonds consecutively (195ms)
matic-staker:         claim two of three test unbonds inconsecutively (145ms)
matic-staker:         claim just one withdrawal (98ms)
matic-staker:     RESTAKE
matic-staker:       Vault: Simulate rewards accrual
matic-staker:         Simulating `SubmitCheckpoint` transaction on RootChainProxy (28252ms)
matic-staker:       Vault: compound rewards
matic-staker:         rewards compounded correctly (compoundRewards: using unclaimed rewards) (17326ms)
matic-staker:         rewards compounded correctly (compoundRewards: using claimed rewards) (6247ms)
matic-staker:         compoundRewards correctly updates staked amount (6559ms)
matic-staker:         does not revert when compounding zero rewards (80ms)
matic-staker:         emits an event when restaking on a validator reverts (83ms)
matic-staker:         stakes MATIC present in the vault with the selected validator (293ms)
matic-staker:         reverts when the selected validator is disabled (119ms)
matic-staker:         does not revert when a non-specified validator is disabled (46ms)
matic-staker:         does not revert when MATIC in the vault is below the validator min amount (217ms)
matic-staker:         leaves no MATIC in the vault when there are liquid rewards to restake (11884ms)
matic-staker:         restakes liquid rewards on multiple validators (6417ms)
matic-staker:     RESTAKE
matic-staker:       Vault: Simulate rewards accrual
matic-staker:         Simulating `SubmitCheckpoint` transaction on RootChainProxy (28100ms)
matic-staker:       Vault: compound rewards
matic-staker:         rewards compounded correctly (compoundRewards: using unclaimed rewards) (17253ms)
matic-staker:         rewards compounded correctly (compoundRewards: using claimed rewards) (6170ms)
matic-staker:         compoundRewards correctly updates staked amount (6465ms)
matic-staker:         does not revert when compounding zero rewards (76ms)
matic-staker:         emits an event when restaking on a validator reverts (75ms)
matic-staker:         stakes MATIC present in the vault with the selected validator (285ms)
matic-staker:         reverts when the selected validator is disabled (105ms)
```

```
matic-staker:        does not revert when a non-specified validator is disabled (45ms)
matic-staker:        does not revert when MATIC in the vault is below the validator min amount (204ms)
matic-staker:        leaves no MATIC in the vault when there are liquid rewards to restake (11746ms)
matic-staker:        restakes liquid rewards on multiple validators (6483ms)
matic-staker:        exploit scenario1 (17187ms)
matic-staker:   ALLOCATION
matic-staker:      pass: making two allocations adding up to MATIC amount larger than user deposited MATIC (118ms)
matic-staker:      pass: allocating twice adds up to allocated amount correctly (119ms)
matic-staker:      pass: first allocation updates state accordingly (121ms)
matic-staker:      pass: multiple allocations update share prices in allocation mappings correctly (28467ms)
matic-staker:      pass: multiple allocations to different people work correctly (6066ms)
matic-staker:      multiple allocations to different people at different shareprices work correctly (22743ms)
matic-staker:      pass: should be able to transfer allocated balance (96ms)
matic-staker:      pass: allocate full amount deposited to one other user (230ms)
matic-staker:      pass: overallocation: allocate 3x more than deposited, recipient rewards math is unchanged,
 ↪  distributors deposit amount is reduced to cover all rewards (29912ms)
matic-staker:      pass: distributing rewards does not affect allocated amount (28409ms)
matic-staker:      pass: repeated allocation to the same recipient (with accrual of rewards in between), computes new
 ↪  combined allocation share price correctly (28701ms)
matic-staker:      dump, deposit, allocate, accrue, distribute, withdraw deposited returns correct getUserInfo (6258ms)
matic-staker:      deposit, allocate, accrue, distribute, withdraw deposited returns correct getUserInfo (23039ms)
matic-staker:      fail: distributing rewards fails if distributor has transferred deposited amount beforehand (5799ms)
matic-staker:      fail: allocating more than deposited at once (38ms)
matic-staker:      fail: allocating zero
matic-staker:   DEALLOCATE
matic-staker:      Emits 'Deallocated' event with expected parameters
matic-staker:      Reverts if caller has not made an allocation to the input recipient
matic-staker:      Reverts via underflow if deallocated amount larger than allocated amount
matic-staker:      Reverts if remaining allocation is under 1 MATIC
matic-staker:      Removes recipient from distributor's recipients if full individual deallocation (280ms)
matic-staker:      Removes distributor from recipient's distributors if full individual deallocation (283ms)
matic-staker:     Individual Allocation State
matic-staker:        Individual allocation price is not changed during deallocation
matic-staker:        Reduces individual allocation by deallocated amount
matic-staker:        Deletes individual allocation from storage if full individual deallocation
matic-staker:     Total Allocation State
matic-staker:        Returns zero for total deallocation
matic-staker:        Decreases total allocation amount if partial deallocation
matic-staker:     Functionality
matic-staker:        Deallocate reduces rewards proportionally (5972ms)
matic-staker:        Deallocation leads to rewards if the reduced amount was allocated initially (before any
 ↪  distribution) (5994ms)
matic-staker:        Flow of allocating, deallocating and distributing as rewards accrue (22815ms)
matic-staker:   DISTRIBUTION
matic-staker:      Rewards earned via allocation equal rewards earned via deposit (5845ms)
matic-staker:      Can withdraw allocated amount after distributeRewards call (6050ms)
matic-staker:      Can withdraw combined allocated amounts after distributeAll call (11406ms)
matic-staker:      Multiple distributeRewards calls are equivalent to single distributeAll call (12045ms)
matic-staker:     External Methods
matic-staker:       distributeAll
matic-staker:         No rewards to distribute
matic-staker:            No rewards distributed to recipients (59ms)
matic-staker:         Distribute rewards
matic-staker:            Rewards get distributed to recipients (109ms)
matic-staker:            getTotalAllocated returns current global price after distribution (121ms)
matic-staker:     Internal Methods
matic-staker:       _distributeRewards
matic-staker:          Emits 'DistributedRewards' with correct parameters inside distributeAll call (58ms)
matic-staker:          Transfers rewards as TruMATIC to recipient (79ms)
matic-staker:          Transfers TruMATIC recipientOneTruMATICFee to treasury (70ms)
matic-staker:          Updates individual price allocation (72ms)
matic-staker:       _distributeRewardsUpdateTotal
matic-staker:          Reverts if no allocation made by distributor to input recipient
matic-staker:          Skips reward distribution if global share price same as individual share price (48ms)
matic-staker:          Updates price of distributor's total allocation (5628ms)
matic-staker:          Emits 'DistributedRewards' event with correct parameters (5540ms)
matic-staker:     distribute MATIC rewards
matic-staker:       distributeRewards
matic-staker:          Reverts if distributor does not have enough MATIC (5498ms)
matic-staker:          No TruMATIC is transferred to recipients when distributing MATIC (5652ms)
matic-staker:          The equivalent amount of TruMATIC is transferred to the user when distributing MATIC (14199ms)
```

```
matic-staker:       distributeAll
matic-staker:         Reverts if distributor does not have enough MATIC (5526ms)
matic-staker:         No TruMATIC is transferred to recipients when distributing MATIC (5518ms)
matic-staker:         The equivalent amount of TruMATIC is transferred to the user when distributing MATIC (11524ms)
matic-staker:   MULTI CHECKPOINTS
matic-staker:     Lifecycle testing: Receiver correctly aggregates shares+rewards across multiple checkpoints,
 ↪  Depositor, receiver and treasury can withdraw max withdraw amount. (28666ms)
matic-staker: ACCRUED
matic-staker:     Invariant testing: allocating across two sets of users. Same workflow accross two separate user
 ↪  groups accrues the same amount of rewards (11848ms)
matic-staker:   TruMATIC ERC20 Functionality
matic-staker:     has a name
matic-staker:     has a symbol
matic-staker:     totalSupply
matic-staker:       totalSupply equals totalStaked for first deposits
matic-staker:       totalSupply is not altered by rewards accrual without deposit (5512ms)
matic-staker:       new deposit after reward accrual increases totalSupply (5671ms)
matic-staker:       withdraw requests pre accrual decrease totalSupply correctly (355ms)
matic-staker:       withdraw requests post accrual decrease totalSupply correctly (5954ms)
matic-staker:     balanceOf
matic-staker:       correctly updates balances post deposit (177ms)
matic-staker:       correctly updates sharePrice post reward accrual (5637ms)
matic-staker:     transfer
matic-staker:       correctly transfers post deposit (201ms)
matic-staker:       Reverts with custom error if more than users balance is transferred
matic-staker:       Transfer post allocation works (263ms)
matic-staker:     transferFrom
matic-staker:       Reverts without allowance/with insufficient balance (39ms)
matic-staker:       transferFrom after deposit works (207ms)
matic-staker:       transferFrom after allocation works (271ms)
matic-staker:     Allocation
matic-staker:       totalSupply
matic-staker:         distributing rewards does not affect totalSupply (5777ms)
matic-staker:   UPGRADE
matic-staker:     Staker contract
matic-staker:       can upgrade staker contract (3155ms)
matic-staker:   267 passing (14m)

> whitelist:test
whitelist:   HAPPY PATH
whitelist:     should return true when checking whether the owner is an agent
whitelist:     allows the owner to whitelist a user without being explicitly listed as an agent
whitelist:     addAgent adds an agent but doesn't whitelist the agent's address (39ms)
whitelist:     removeAgent should not remove the agent from the whitelist (53ms)
whitelist:     should whitelist a new user (48ms)
whitelist:     should blacklist a new user (44ms)
whitelist:     should whitelist a blacklisted user (54ms)
whitelist:     should blacklist a whitelisted user (52ms)
whitelist:     should clear a blacklisted user's whitelisting status (55ms)
whitelist:     should clear a whitelisted user's whitelisting status (56ms)
whitelist:     should whitelist a cleared user's whitelisting status (85ms)
whitelist:     should blacklist a cleared user's whitelisting status (75ms)
whitelist:     adding an agent should emit an event
whitelist:     removing an agent should emit an event
whitelist:     should raise an event when a user is whitelisted
whitelist:     should raise an event when a user is blacklisted
whitelist:     should raise an event when a blacklisted user's whitelisting status is cleared
whitelist:     should raise an event when a whitelisted user's whitelisting status is cleared
whitelist:     should raise an event when a whitelisted user is blacklisted
whitelist:     should raise an event when a blacklisted user is whitelisted
whitelist:   UNHAPPY PATH
whitelist:     should fail when being initialized again
whitelist:     addAgent should revert if not called by the agent
whitelist:     removeAgent should revert if not called by the agent
whitelist:     should revert if a non-agent is trying to whitelist a user
whitelist:     should revert if a non-agent is trying to blacklist a user
whitelist:     should revert if a non-agent is trying to clear a user's whitelisting status
whitelist:     should revert if trying to whitelist an already whitelisted user
whitelist:     should revert if trying to blacklist an already blacklisted user
whitelist:     should revert if trying to clear an already cleared whitelisting status
whitelist:   29 passing (4s)
```

8.3 Code Coverage

```
> npm run coverage-sol
```

The relevant output is presented below.

```
matic-staker: ----------------------------|----------|----------|----------|----------|----------------|
matic-staker: File                        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
matic-staker: ----------------------------|----------|----------|----------|----------|----------------|
matic-staker:  interfaces/                |      100 |      100 |      100 |      100 |                |
matic-staker:   IMasterWhitelist.sol      |      100 |      100 |      100 |      100 |                |
matic-staker:   IStakeManager.sol         |      100 |      100 |      100 |      100 |                |
matic-staker:   ITruStakeMATICv2.sol      |      100 |      100 |      100 |      100 |                |
matic-staker:   IValidatorShare.sol       |      100 |      100 |      100 |      100 |                |
matic-staker:  main/                      |      100 |    92.65 |      100 |      100 |                |
matic-staker:   ERC4626Storage.sol        |      100 |      100 |      100 |      100 |                |
matic-staker:   TruStakeMATICv2.sol       |      100 |    92.65 |      100 |      100 |                |
matic-staker:   TruStakeMATICv2Storage.sol |      100 |      100 |      100 |      100 |                |
matic-staker:   Types.sol                 |      100 |      100 |      100 |      100 |                |
matic-staker: ----------------------------|----------|----------|----------|----------|----------------|
matic-staker: All files                   |      100 |    92.65 |      100 |      100 |                |
matic-staker: ----------------------------|----------|----------|----------|----------|----------------|

whitelist: ----------------------|----------|----------|----------|----------|----------------|
whitelist: File                  |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
whitelist: ----------------------|----------|----------|----------|----------|----------------|
whitelist:  interfaces/          |      100 |      100 |      100 |      100 |                |
whitelist:   IMasterWhitelist.sol |      100 |      100 |      100 |      100 |                |
whitelist:  main/                |      100 |      100 |      100 |      100 |                |
whitelist:   MasterWhitelist.sol  |      100 |      100 |      100 |      100 |                |
whitelist: ----------------------|----------|----------|----------|----------|----------------|
whitelist: All files             |      100 |      100 |      100 |      100 |                |
whitelist: ----------------------|----------|----------|----------|----------|----------------|
```

8.4 Slither

All the relevant issues raised by Slither have been incorporated into the issues described in this report.

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the Blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer