Cyberscope

A TAC Security Company

Audit Report

Mage Labs

July 2025

Repository https://github.com/Bors-magedao/staking

Commit 9420f1f3399cfb2d117e98a764a8ca3a73c473ff

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/Bors-magedao/staking
Commit	9420f1f3399cfb2d117e98a764a8ca3a73c473ff

Audit Updates

Initial Audit	11 Jun 2025 https://github.com/cyberscope-io/audits/blob/main/mage/v1/audit.pdf
Corrected Phase 2	05 Jul 2025 https://github.com/cyberscope-io/audits/blob/main/mage/v2/audit.pdf
Corrected Phase 3	17 Jul 2025

Source Files

Filename	SHA256
./errors.rs	4039ed64e810e23f8390123502785a4cbc87f78a24b0b5724935e15f4afc066b
./state/stake_receipt.rs	41de49698ff0632182a8fd808f8b50f05f7adbac692d1acff92e7396b3a82661
./state/mod.rs	27f6ec65b6423d551ddb8c92ef4b2cbc738bbee285be02949c4149c3f36b5e09
./state/stake_pool.rs	6bbc55265e024031090882523857090 23d415e5036c4f66c905787f0444daa6c

./instructions/claim_reward_tokens.rs	95f0d5ffdf58ea2575870d2aef62ac4a1a44f06621b3059e1b53e5a177f03ecc
./instructions/unstake.rs	6e28ba0677668e3671eaef21c3abb3df4096937439f5a04a2577e021d0f64a1c
./instructions/stake.rs	db6a0594486ed0d6aecc15160f6bdd1718785516546f85e1d86d11c8cc45d99f
./instructions/mod.rs	e569b8b8b92046acfc2d8c7ae6ef3ab5ef038ad330b1c5e574976a3dfcd15d6c
./instructions/stake_nft.rs	b0db2c178042d1526611fa9b06a3eec04db60a4ba786709027417345a2c40725
./instructions/withdraw.rs	962b8c698f9367adde71c87dfbe082008b38132dd9b9c46a30fa35eb2cc13e1b
./instructions/admin/update_authority.rs	ce6355b19027fbb8b278b2b068cee76ec34b68d3bdcb194ea15128c471511c46
./instructions/admin/add_token.rs	56c89c0dcc1df1907d0700fe596bda5feec45948d5439106341d3a83cb8cf61c
./instructions/admin/mod.rs	fe84eff17640e1fb35cb7a753f8818fb2d4648e013e119bff48c8a0beb8b2449
./instructions/admin/create_stake_pool.rs	4f46f7d28925c6dd1ac48f5cb2ff9e7067957f02a4d8987fe9a3900aa3c8ad23
./instructions/admin/add_nft.rs	5619d5da033f8f730b16635e619fef61996a0e3fa3ade4163d964b5e54784a10
./instructions/admin/add_reward_pool.rs	057c5162ffcfb05b9df00771f23a20fc516c24c80b742f9ee4c25b36ee072d35

./instructions/increase_stake.rs	66dc80d3da48a1e98f0270acb40c2980bece91065d1d760123c808aa8b2a0e5d
./instructions/mint_burn_redeem.rs	9064ee08236acbed5323b16699233bee61407b59ed951228f01c400f7f298bf6
./lib.rs	10517da7dcf816be6d482485d4097e8d1cc45867e23952337e0c6b9bf31953fd
./uint.rs	1d842809e43e1ee702390e311492eaef864c8ced7711e81fe31e76c239b70a25
./macros.rs	c7e386afda5354bfa4fa90a15fa4e2841bc216c09294810ded3324fd2015a1ab

Overview

The Mage contracts implement a modular and extensible staking system that supports both fungible tokens and NFTs, allowing users to stake assets in exchange for proportional reward distributions. At its core, the system revolves around the `StakePool` account, which maintains authority, tracks custom asset weights, and manages multiple `RewardPools`. Administrators can initialize stake pools, add supported tokens or NFT collections with specific weights, assign reward mints, and update pool authorities. The reward mechanism ensures that rewards are distributed fairly based on weighted stake contributions, with accounting tokens optionally redeemable for actual reward tokens. The design promotes flexibility, precise reward allocation, and composability with various asset types while enforcing access control and account validation throughout the lifecycle of staking and reward operations.

Admin Functionality

The admin functionality of the protocol enables privileged users to configure and manage the `StakePool` through a set of permissioned instructions. Using `CreateStakePool`, an admin initializes a new pool instance with an assigned authority. The `AddToken` and `AddNft` instructions allow the admin to register new stakeable assets—either fungible tokens with associated vaults or NFT collections verified through Metaplex metadata—each with custom weight parameters influencing stake distribution. Through `AddRewardPool`, the admin defines reward configurations by linking real and synthetic reward mints with vaults and setting mint authorities. Finally, `UpdateAuthority` allows for the transfer of administrative control by updating the `StakePool`'s authority key, ensuring flexible and secure protocol governance. All critical operations are gated by signer-based authority checks and account constraints to ensure only authorized entities can modify pool state.

Stake

Users can stake fungible tokens into the protocol by transferring assets from their wallet into a designated vault managed by the `StakePool`. Upon staking, a `StakeReceipt` is generated, recording the user's effective stake based on asset weighting, the original deposit amount, and a snapshot of current reward accumulators. This receipt enables

future reward claims and governs unstaking eligibility. The protocol also recalculates global rewards upon new deposits to ensure accurate distribution.

Increase Stake

The `IncreaseStake` instruction allows users to add more tokens to an existing stake position. Before increasing their stake, users automatically claim and redeem their accumulated rewards. The additional deposit is converted into an updated effective stake, increasing both the user's and the pool's total weighted stake. The process ensures rewards are settled accurately and state remains consistent before stake growth.

Claim Reward Tokens

This function lets users claim synthetic reward tokens that reflect their share of rewards accumulated over time. The protocol recalculates reward rates based on vault balances and user stake before minting the appropriate amount of synthetic tokens. These synthetic tokens represent a user's reward entitlement and can be tracked or redeemed in a later step.

Mint, Burn, and Redeem

This flow enables users to convert synthetic reward tokens into real reward tokens. The contract mints synthetic rewards, burns them from the user's account, and transfers an equivalent amount of real tokens from the reward vault. This two-step process preserves accounting integrity while ensuring users receive actual value from their earned rewards.

Stake NFT

The `StakeNft` instruction allows users to stake NFTs that belong to verified collections. The contract validates the NFT's metadata and ensures it's part of an approved collection. Upon staking, the NFT is transferred to a vault controlled by the `StakePool`, and a `StakeReceipt` is issued to track the user's contribution. The effective stake is computed based on the NFT asset's weight, and rewards begin accruing accordingly. The user's source token account is closed to reclaim rent once the NFT is secured in the vault.

Unstake

Users initiate the unstaking process using the `Unstake` instruction, which applies to both fungible token and NFT stakes. This operation ensures rewards are up to date by recalculating the pool's reward distribution and minting any outstanding rewards. It then decreases the total weighted stake and updates the user's `withdrawable_at` timestamp, enforcing a cooldown period before the actual withdrawal is allowed. This preserves fair reward distribution and prevents immediate stake-exit abuse.

Withdraw

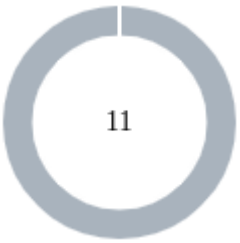
Once the cooldown period ends, users can execute the `Withdraw` instruction to retrieve their staked tokens or NFTs. The contract validates the stake receipt and, if the asset is an NFT, verifies its metadata again. The staked asset is transferred from the protocol vault back to the user's wallet. If the withdrawn asset is an NFT, the associated vault is closed to clean up and reclaim rent. This instruction finalizes the full lifecycle of a stake and ensures secure asset return to the rightful owner.

Here is a clear and concise paragraph describing how rewards are applied in this system:

Reward Distribution Mechanism

The reward system distributes tokens to stakers proportionally based on their *effective stake*, which accounts for the weight of the staked asset. When tokens are deposited into a reward vault, the `recalculate_rewards_per_effective_stake` function updates each `RewardPool` 's `rewards_per_effective_stake` accumulator by computing the difference between the current and previous vault balances. This value is scaled and divided by the `total_weighted_stake` to ensure fair allocation. During withdrawal or unstaking, the user's share of rewards is calculated by multiplying the difference in reward-per-stake with their effective stake, then minting accounting reward tokens. If `burn_and_redeem` is enabled, those tokens are burned and equivalent actual rewards are transferred from the vault. This mechanism ensures precision, fairness, and compatibility with both fungible and NFT-based staking assets.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	11	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Acknowledged
	FSA	First Stake Advantage	Acknowledged
	IRPI	Insecure Reward Pool Input	Acknowledged
	MEE	Missing Events Emission	Acknowledged
	MOTV	Missing Owner Token Validation	Acknowledged
	MPC	Missing Period Check	Acknowledged
	MRAV	Missing Reward Account Validations	Acknowledged
	MRPV	Missing Reward Pool Validation	Acknowledged
	MSTRP	Missing Synthetic Token Redemption Path	Acknowledged
	TSI	Tokens Sufficiency Insurance	Acknowledged
	UNWR	Uniform NFT Weighting Risk	Acknowledged

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	lib.rs#17
Status	Acknowledged

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
  /* Admin related instructions below */

    #[access_control(CreateStakePool::validate(&ctx))]
    pub fn create_stake_pool(
        ctx: Context<CreateStakePool>,
        args: CreateStakePoolArgs,
    ) -> Result<()> {
        instructions::admin::create_stake_pool::handler(ctx, args)
    }

    #[access_control(AddToken::validate(&ctx, &args))]
    pub fn add_token(ctx: Context<AddToken>, args: AddTokenArgs) -> Result<()>
{
        instructions::admin::add_token::handler(ctx, args)
    }

    #[access_control(AddNft::validate(&ctx, &args))]
    pub fn add_nft(ctx: Context<AddNft>, args: AddNftArgs) -> Result<()> {
        instructions::admin::add_nft::handler(ctx, args)
    }

    #[access_control(AddRewardPool::validate(&ctx))]
    pub fn add_reward_pool(ctx: Context<AddRewardPool>) -> Result<()> {
        instructions::admin::add_reward_pool::handler(ctx)
    }

    #[access_control(UpdateAuthority::validate(&ctx))]
    pub fn update_authority(
        ctx: Context<UpdateAuthority>,
        args: UpdateAuthorityArgs,
    ) -> Result<()> {
        instructions::admin::update_authority::handler(ctx, args)
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

FSA - First Stake Advantage

Criticality	Minor / Informative
Location	stake_pool.rs#194
Status	Acknowledged

Description

If rewards exist in the reward pool before staking begins, the first user to stake will be able to mint reward tokens equal to the entire pre-existing balance when a second user stakes. Specifically, the first staker receives 100% of the reward allocation at that moment, effectively minting rewards equivalent to the balance present in the vault prior to the start of staking. This behavior may be exploited and could lead to system manipulation.

```
pub fn recalculate_rewards_per_effective_stake<'info>(
        &mut self,
        remaining_accounts: &[AccountInfo<'info>],
        reward_vault_account_page_size: usize,
    ) -> Result<()> {
    ...
}
```

Recommendation

The reward vault balance must be carefully managed to align with the intended system design. In particular, the balance should increase progressively in accordance to the amount of tokens staked in the system. This ensures a fair distribution of rewards and mitigates the risk of unintended minting behavior or potential exploitation.

IRPI - Insecure Reward Pool Input

Criticality	Minor / Informative
Location	stake.rs#84 increase_stake.rs#75
Status	Acknowledged

Description

The contract relies on `ctx.remaining_accounts` to pass in all relevant `RewardPool` accounts for recalculating reward distribution. This design delegates responsibility to the caller to supply the correct accounts in the correct order, which introduces risks of misconfiguration or intentional manipulation. If the wrong set or sequence of accounts is provided, reward recalculation could behave incorrectly, leading to misallocated rewards, incorrect accounting, or silent failures that are difficult to detect on-chain.

```
    let stake_pool = &mut ctx.accounts.stake_pool;
     stake_pool.recalculate_rewards_per_effective_stake(
         &ctx.remaining_accounts,
         Stake::REMAINING_ACCOUNT_PAGE_SIZE,
     )?;
```

Recommendation

It is recommended to fetch or derive all relevant `RewardPool` accounts internally or through deterministic means instead of relying on user-supplied remaining accounts. If that is not feasible, implement strict validation logic to verify that the supplied accounts match the expected reward pools both in content and order. This ensures that reward calculations operate on trusted data and preserves the integrity of the staking and distribution process.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	create_stake_pool.rs#41 update_authority.rs.rs#29 mint_burn_redeem.rs#40
Status	Acknowledged

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
pub fn handler(ctx: Context<CreateStakePool>, args: CreateStakePoolArgs) ->
Result<()> {
...
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MOTV - Missing Owner Token Validation

Criticality	Minor / Informative
Location	withdraw.rs#16
Status	Acknowledged

Description

The contract does not perform runtime validation to ensure that the `owner_token_account` is correctly configured. Specifically, there is no check verifying that the account is owned by the `owner`, nor that it holds the correct token mint associated with the `stake_receipt`. This omission allows users to supply arbitrary token accounts, including accounts they control that use a different mint. As a result, token transfers during the withdrawal process could be redirected to unintended destinations or token types, compromising the integrity and correctness of reward or principal withdrawals.

```
#[account(mut)]
 pub owner_token_account: Account<'info, TokenAccount>,
```

Recommendation

It is recommended to include the following runtime validations in the `validate` function of the `Withdraw` instruction:

- Ensure that the `owner_token_account.owner` matches the `owner.key()`.
- Ensure that the `owner_token_account.mint` matches the `stake_receipt.mint`.

Adding these validations will ensure that the withdrawal can only be made to a legitimate and expected token account, preserving the integrity of the withdrawal mechanism.

Team Update

The team has acknowledged that this is not a security issue and states:

It is intentional that there is no owner check on the TokenAccount being withdrawn to. Given the owner of the StakeReceipt is a signer, they are allowed to enter an external TokenAccount. This is a feature that allows them to withdraw into a separate wallet other than the one they signed with. However, they MUST has signed with the wallet that owns the StakeReceipt. Separately, a mint check is not necessary as the Transfer CPI would fail if the mint does not match that of the from and to accounts.

MPC - Missing Period Check

Criticality	Minor / Informative
Location	create_stake_pool.rs#28
Status	Acknowledged

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically the contract does not ensure the cooldown period is not assigned the zero value. If such a value is used, the system may be exposed to manipulation and potential loss of funds.

```
   pub fn validate(_ctx: &Context<CreateStakePool>, args:
&CreateStakePoolArgs) -> Result<()> {
        require!(
            args.authority != Pubkey::default(),
            ErrorCode::InvalidAuthority
        );
        Ok(())
    }
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

Team Update

The team has acknowledged that this is not a security issue and states:

Since the cooldown function is restricted to administrators only and is immutable once applied, a cooldown value greater than zero will be set.

MRAV - Missing Reward Account Validations

Criticality	Minor / Informative
Location	stake_pool.rs#346
Status	Acknowledged

Description

The `mint_accounting_reward_tokens` function lacks critical runtime checks for several user-supplied accounts, leaving the reward distribution mechanism vulnerable to misdirection or spoofing:

1. Recipient Account Mismatch: The `owner_accounting_reward_token_info` is not checked for correct ownership (`owner.key()`) .
2. Burn-and-Redeem Destination Mismatch: In the `burn_and_redeem` branch, the destination SPL token account (`user_reward_token`) is not validated to belong to the caller or match the real `reward_pool.reward_mint` . This opens the door to redirection of real rewards.

```
pub fn mint_accounting_reward_tokens<'info>(
...
    let owner_accounting_reward_token_info =
    &remaining_accounts[remaining_accounts_index + 2];
    let cpi_accounts = MintTo {
    mint: accounting_reward_mint_info.clone(),
    to: owner_accounting_reward_token_info.clone(),
    authority: stake_pool_account.clone(),
    };
...
}
```

Recommendation

It is recommended to add the following validations:

- `require!(owner_accounting_reward_token_info.mint == accounting_reward_mint_info.key(), ...)`
- `require!(owner_accounting_reward_token_info.owner == owner.key(), ...)`
- `require!(user_reward_token.owner == owner.key(), ...)`
- `require!(user_reward_token.mint == reward_pool.reward_mint, ...)`

These checks are necessary to enforce correct and secure reward delivery, prevent misdirection of tokens, and preserve the integrity of both synthetic and real reward flows.

Team Update

The team has acknowledged that this is not a security issue and states:

1. This is an unnecessary check. The owner is a signature of the instruction, thus the program can assume that the owner_accounting_reward_token_info is the intended TokenAccount whether it is owned by the owner or not. The owner could allow reward tokens to be sent to another wallet if they so chose.
2. We do not make the case that the address receiving rewards MUST be the same as the owner of the StakeReceipt. As long as the owner of the StakeReceipt is the signer of the transaction, we know that their intention is to have the accounting tokens sent to an external TokenAccount. Separately, a Mint inconsistency would error within the Token program's Mint instruction and thus not required to be checked in the staking program.

MRPV - Missing Reward Pool Validation

Criticality	Minor / Informative
Location	stake.rs#54
Status	Acknowledged

Description

The `validate` function in the `Stake` instruction does not enforce the presence of at least one reward pool before allowing a user to stake assets. While internal checks such as `get_asset_by_mint` ensure the asset exists, they do not verify whether any reward pool is available to distribute rewards. This omission could lead to a misleading user experience where users are allowed to stake tokens without receiving any rewards, or where the staking operation proceeds under invalid economic conditions.

```
    pub fn validate(_ctx: &Context<Stake>, _args: &StakeArgs) -> Result<()> {
        // Validation for checking if the mint is in the list of Assets
happens in `get_asset_by_mint`.

        Ok(())
    }
```

Recommendation

It is recommended to add a validation check to ensure that at least one active reward pool exists before allowing staking to proceed. This guarantees that staking actions are meaningful and that reward calculations have valid targets. Adding such validation improves the reliability of the protocol and prevents users from unknowingly interacting with an incomplete or improperly configured reward system.

MSTRP - Missing Synthetic Token Redemption Path

Criticality	Minor / Informative
Location	claim_reward_tokens.rs#53
Status	Acknowledged

Description

The contract mints synthetic accounting tokens (used for tracking rewards) to users during reward claims but lacks a redemption mechanism that allows users to convert or burn these tokens in exchange for the actual underlying reward tokens. As a result, these synthetic tokens accumulate in user accounts without a way to redeem them for value. Additionally, the `mint_burn` logic only handles deltas during reward distribution, not full redemption, further reinforcing the lack of an exit path. This design creates a misleading impression that users have received rewards when, in reality, they hold non-redeemable synthetic balances.

```
// For each reward_pol, mint the reward tokens
    ctx.accounts.stake_pool.mint_accounting_reward_tokens(
        ctx.accounts.owner.to_account_info(),
        ctx.accounts.stake_pool.to_account_info(),
        ctx.accounts.token_program.to_account_info(),
        &ctx.accounts.stake_receipt,
        &ctx.remaining_accounts,
        ClaimRewardTokens::REMAINING_ACCOUNT_PAGE_SIZE,
        false,
    )?;
```

Recommendation

It is recommended to implement a clear and verifiable redemption mechanism that allows users to convert their synthetic reward tokens into real rewards. This should include explicit burn logic tied to minting of real reward tokens, along with proper accounting and validation to prevent abuse. Without such a mechanism, the synthetic rewards model remains incomplete and may confuse users or lead to loss of expected value.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	stake_pool.rs#377
Status	Acknowledged

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
// Transfer the reward tokens
let cpi_accounts = Transfer {
    from: reward_vault_info.clone(),
    to: user_reward_token.clone(),
    authority: stake_pool_account.clone(),
};
let cpi_ctx = CpiContext {
    accounts: cpi_accounts,
    remaining_accounts: vec![],
    program: token_program_info.clone(),
    signer_seeds: &[stake_pool_signer_seeds!(self)],
};
token::transfer(cpi_ctx, total_claimable)?;
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

Team Update

The team has acknowledged that this is not a security issue and states:

The claimable revenue will be entered automatically, as the DEX includes a function by which fee revenue is sent to the RewardPools for liquidity providers.

UNWR - Uniform NFT Weighting Risk

Criticality	Minor / Informative
Location	add_nft.rs#66
Status	Acknowledged

Description

The contract applies a fixed weight to each NFT asset without accounting for the collection's total supply or the number of NFTs actually staked. Unlike fungible tokens—where weight reflects the staked amount—NFTs are assigned a flat asset weight, which is then applied uniformly across all individual NFTs. This design leads to disproportionate reward distribution, where each NFT receives an equal share of the total NFT asset weight, regardless of how many NFTs exist or are staked. For example, if the NFT asset weight is set to 400/1000 and 10 NFTs are staked, each NFT effectively receives 400/1000. However, if the collection size is 5000, this approach over-allocates rewards relative to their intended share, potentially resulting in inflation or reward abuse.

```
#[derive(AnchorDeserialize, AnchorSerialize)]
pub struct AddNftArgs {
    pub weight_numerator: u64,
    pub weight_denominator: u64,
}

pub fn handler(ctx: Context<AddNft>, args: AddNftArgs) -> Result<()>
{
    let stake_pool = &mut ctx.accounts.stake_pool;

    let asset_weight = Asset::new(
        &ctx.accounts.mint.key(),
        args.weight_numerator,
        args.weight_denominator,
        None,
        Some(ctx.accounts.metadata.key),
    );
    stake_pool.set_next_asset(asset_weight)?;

    Ok(())
}
```

Recommendation

It is recommended to adjust the NFT asset weight calculation by dividing the assigned asset weight by the total supply (or total staked amount) of NFTs. This would ensure each NFT receives a proportional share of the assigned weight, aligning reward distribution with actual stake representation. Alternatively, separate logic should be implemented for NFT-based assets to normalise their contribution based on collection size, preventing disproportionate allocation of pool rewards.

Summary

Mage Labs contract implements a weighted staking and reward distribution mechanism supporting both fungible tokens and NFTs. This audit investigates security issues, business logic concerns, and potential improvements to ensure correctness, efficiency, and readiness for production deployment.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io