Cyberscope

Audit Report

BrushO Network

January 2025

Repository	https://github.com/aitoothbrush/brusho-program-library/tree/main/programs/reward-distributor
Commit	906cd89f4c20cbd563b0235bfdd23f488d81f0e0
Audited by	© cyberscope

Table of Contents

Table of Contents	**1**
Risk Classification	**3**
Review	**4**
Audit Updates	4
Source Files	4
Overview	**6**
Findings Breakdown	**7**
Diagnostics	**8**
MRLM - Missing Rate Limiting Mechanism	10
Description	10
Recommendation	10
MLE - Missing Log Events	12
Description	12
Recommendation	12
UCDB - Unsafe Canopy Data Bounds	13
Description	13
Recommendation	13
HOL - Hardcoded Oracle Limit	14
Description	14
Recommendation	14
IOA - Index Out-of-Bounds Access	15
Description	15
Recommendation	15
IPR - Inefficient Proof Representation	16
Description	16
Recommendation	16
IASV - Insufficient Account Space Validation	17
Description	17
Recommendation	17
IOV - Insufficient Operation Validation	18
Description	18
Recommendation	18
IODV - Insufficient Oracle Data Validation	19
Description	19
Recommendation	19
MDEM - Missing Descriptive Error Message	20
Description	20
Recommendation	20
MEH - Missing Error Handling	21

Description 21
Recommendation 21
MIV - Missing Input Validation 22
Description 22
Recommendation 22
MNLC - Missing Name Length Check 23
Description 23
Recommendation 23
OBA - Out of Bounds Access 24
Description 24
Recommendation 24
PVF - Period Validation Flaw 25
Description 25
Recommendation 25
PIO - Potential Integer Overflow 26
Description 26
Recommendation 26
UB - Unchecked Borrow 27
Description 27
Recommendation 27
VNT - Variable Naming Typo 28
Description 28
Recommendation 28
Summary 29
Disclaimer 30
About Cyberscope 31

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/aitoothbrush/brusho-program-library/tree/main/programs/reward-distributor
Commit	906cd89f4c20cbd563b0235bfdd23f488d81f0e0

Audit Updates

Initial Audit	20 Jan 2025

Source Files

Filename	SHA256
error.rs	659e8afa9787ea92f2478c761eb9a7ed6bc4d0cba0848a7e982bcf30f2398a24
instructions/set_canopy_data.rs	bbaf34685622e92893938a24a65ae8a390c3d721e9c8a5fdb11eed56df38fe66
instructions/create_canopy.rs	dd4939cdb8b95288cae3066ce68bb359790188793e5664417e2a6f60a4c5ea98
instructions/claim_rewards.rs	8439e73e2e66f145c44865d86bdaa496576e8d2be5a80688b5e4b0c382914b74
instructions/create_distributor.rs	e2dba3a7889f5291b14aab467a006e9a0104959b287bb6ff3165147bca0712b8
instructions/create_distribution_tree.rs	7d653c367bafd8cdb6daac39fd16d661774d97fb8a788f2d59851f183deff569
instructions/activate_distribution_tree.rs	701680c70942dde3d9d9a0b23989bf7fe9698aaadded577660e5fa7979144df0

instructions/mod.rs	5956fb034a1d63b7cee8f55480c358afd66a819d67abd2f43ec81a47f9fce412
instructions/report_oracle.rs	d1cabb9ad583d5c159a6e4a44cdd5b1a2a632cd9ce01749c4a6bce6a8624eead
instructions/update_distributor.rs	2cf494863a4b18f8e6b9053cc5ee404e7da7585b76ac5021009b16555ab02a10
instructions/close_canopy.rs	8b76211e170e5685f5e8f48138bce0e7bf441f792ed996981a3bb0b97581daa7
lib.rs	9d34e8f2c48fdcdd0a834bbe5f0224b71f5d1f43a45094350a4e2ba98c0c8232
circuit_breaker.rs	8e1b9f4d91d705d122a83b98b8fe1c361b76b78925515ded64886c1a4c84575a
canopy.rs	e6260876d5e48893acb73a4bc3b97aec1dc1a28853bfcba5b079d23f041362c3
merkle_proof.rs	df172e15279a965dbfdfc8b5b8538d708f704f8b15753fff96ae0753ff7c31e3
compressed_nfts.rs	e419aa12e89b6b04250df8881f6c1fa5e32c8886b2936378162a8ed60094cfe2
state.rs	42903b7e9fba2c866584fa8ad64fb5effc45820a709b8af92cb1f41e0521e7ee

Overview

The contract is a decentralized reward distribution system built on Solana, designed to allocate tokens to eligible NFT holders securely and efficiently. It uses cryptographic techniques such as Merkle tree proofs and compressed NFTs to validate ownership and eligibility, ensuring scalability and reducing costs.

At the heart of the system is the **Distributor**, which manages reward allocations, tracks the current distribution period, and integrates governance controls. A **DistributionTree** is used to organize reward data for specific periods, relying on reports from oracles to validate the state of the Merkle tree. The contract ensures consensus by requiring a majority of oracle votes to determine the correct Merkle tree root and maximum depth.

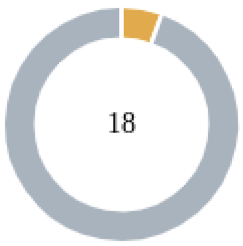
Users claim rewards by submitting proofs of ownership for their NFTs. The contract verifies these proofs using the **Canopy**, a caching mechanism that stores upper levels of the Merkle tree to optimize validation by reducing the size of proofs. After successful validation, rewards are transferred to the user’s account, with security mechanisms in place to prevent fraud.

The contract uses a **Circuit Breaker** to enforce limits on token transfers, ensuring that rewards are distributed within predefined thresholds. This mechanism protects the system from unexpected or malicious token movements by halting operations if certain conditions are breached.

Governance is integral to the contract, allowing trusted authorities to configure and update key parameters, such as adding oracles, updating the Distributor, or managing canopy data. This ensures accountability and adaptability in the system’s operations.

In summary, the contract provides a secure and efficient framework for managing rewards in decentralized applications. It combines NFT-based ownership validation, cryptographic security, governance, and transfer controls to deliver a scalable solution for reward distribution.

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	17

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	1	0	0
●	Minor / Informative	0	17	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	MRLM	Missing Rate Limiting Mechanism	Acknowledged
⚪	MLE	Missing Log Events	Acknowledged
⚪	UCDB	Unsafe Canopy Data Bounds	Acknowledged
⚪	HOL	Hardcoded Oracle Limit	Acknowledged
⚪	IOA	Index Out-of-Bounds Access	Acknowledged
⚪	IPR	Inefficient Proof Representation	Acknowledged
⚪	IASV	Insufficient Account Space Validation	Acknowledged
⚪	IOV	Insufficient Operation Validation	Acknowledged
⚪	IODV	Insufficient Oracle Data Validation	Acknowledged
⚪	MDEM	Missing Descriptive Error Message	Acknowledged
⚪	MEH	Missing Error Handling	Acknowledged
⚪	MIV	Missing Input Validation	Acknowledged
⚪	MNLC	Missing Name Length Check	Acknowledged
⚪	OBA	Out of Bounds Access	Acknowledged

●	PVF	Period Validation Flaw	Acknowledged
●	PIO	Potential Integer Overflow	Acknowledged
●	UB	Unchecked Borrow	Acknowledged
●	VNT	Variable Naming Typo	Acknowledged

MRLM - Missing Rate Limiting Mechanism

Criticality	Medium
Location	claim_rewards.rs#L105
Status	Acknowledged

Description

The `claim_rewards` function lacks a mechanism to limit the frequency of claims from the same user or account within a short timeframe. This absence of rate-limiting exposes the contract to potential exploitation, such as repeated invocation of the function in a short period. An attacker could abuse this to:

- Exhaust system resources by creating high-frequency claim requests.
- Attempt brute force attacks to bypass validation logic.
- Exploit any unaddressed vulnerabilities that may arise during the execution of the function.

Without a cooldown or rate-limiting mechanism, this issue could lead to degraded performance, higher gas costs for legitimate users, and potential system downtime.

```
pub fn claim_rewards<'info>(
    ctx: Context<'_, '_, 'info, 'info, ClaimRewards<'info>>,
    args: ClaimRewardsArgs,
) -> Result<()> {
    ...
}
```

Recommendation

It is recommended to implement a rate-limiting mechanism for the `claim_rewards` function. This could include recording the last claim timestamp for each recipient and enforcing a cooldown period before subsequent claims are allowed. For example:

- Add a `last_claim_timestamp` field to the `Recipient` struct.
- Validate the elapsed time since the last claim before processing a new one.

- Enforce a minimum cooldown duration, ensuring that claims cannot be repeated within this period.

This approach will enhance the system's resilience against abuse, safeguard against brute force attacks, and maintain system performance for all users.

MLE - Missing Log Events

Criticality	Minor / Informative
Location	close_canopy.rs
Status	Acknowledged

Description

The `close` and `close_canopy` functions transfer lamports and close accounts, but they do not emit events to record these actions. Without proper logging, it becomes challenging to track or audit account closures and lamport transfers. This lack of transparency can hinder debugging efforts and make it difficult to ensure compliance with operational or security standards.

```
pub fn close<'info>(info: AccountInfo<'info>, sol_destination:
AccountInfo<'info>) -> Result<()> {
    // Account closure logic, but no logging
    ...
}
```

Recommendation

It is recommended to add event emissions for actions such as account closures and lamport transfers. For instance, an `AccountClosed` event can log the account being closed, the recipient of the lamports, and the amount transferred. This will improve the transparency and auditability of the contract.

UCDB - Unsafe Canopy Data Bounds

Criticality	Minor / Informative
Location	set_canopy_data.rs#L27
Status	Acknowledged

Description

The `set_canopy_data` function does not validate the `args.offset` or the length of `args.bytes` before performing the slicing operation on the `canopy_data` account's data. If the range `(args.offset)..(args.offset + args.bytes.len())` exceeds the size of the `canopy_data` account, it will result in a runtime panic. This oversight allows buggy or malicious inputs to cause crashes, disrupting the program's normal operations and potentially affecting other transactions in the same block.

```
data[(args.offset) as usize..(args.offset) as usize + args.bytes.len()]
    .copy_from_slice(&args.bytes);
```

Recommendation

It is recommended to validate the `args.offset` and the combined size of `args.bytes` to ensure they fall within the bounds of the `canopy_data` account's data length. Adding a validation will prevent runtime panics, ensuring the function handles inputs safely and robustly under all conditions. This will improve the program's reliability and safeguard against disruptions caused by invalid inputs.

HOL - Hardcoded Oracle Limit

Criticality	Minor / Informative
Location	create_distributor.rs#l19
Status	Acknowledged

Description

The `MAX_ORACLES_COUNT` is hard-coded with a value of 5, which limits the number of oracles that can be included in the program. While this value might be sufficient for current use cases, it restricts the scalability and flexibility of the program. If a larger number of oracles is required in the future, the program will need to be upgraded, which could introduce unnecessary overhead and risks. By hard-coding this limit, the program cannot dynamically adapt to changes in requirements without significant effort.

```
pub const MAX_ORACLES_COUNT: usize = 5;
```

Recommendation

It is recommended to store the maximum oracle count as part of the program state instead of hard-coding it. This dynamic configuration would allow for easier updates to the oracle count without requiring a program upgrade. Implementing this change would enhance the program's scalability and flexibility to accommodate future needs.

IOA - Index Out-of-Bounds Access

Criticality	Minor / Informative
Location	report_oracle.rs#L14
Status	Acknowledged

Description

The function does not validate whether the `args.index` provided in `ReportOracleArgs` is within the bounds of the `distributor.oracles` or `distribution_tree.oracle_reports` arrays. If `args.index` exceeds the bounds of these arrays, it could cause a runtime panic when attempting to access the corresponding element. This would result in the entire transaction failing, which could disrupt program functionality and user experience.

```
constraint = distributor.oracles[args.index as usize] == authority.key()
@ RdError::Authorization,

distribution_tree.oracle_reports[args.index as usize] =
Option::Some(args.report);
```

Recommendation

It is recommended to add explicit bounds checks for `args.index` to ensure it falls within the valid range of indices for both `distributor.oracles` and `distribution_tree.oracle_reports`. This will prevent out-of-bounds access and ensure the function operates safely under all conditions.

IPR - Inefficient Proof Representation

Criticality	Minor / Informative
Location	merkle_proof.rs#L26
Status	Acknowledged

Description

The `proof` is currently represented as a `Vec<[u8; 32]>`, which involves heap allocations that may be unnecessary for fixed-depth Merkle trees. This representation increases memory usage and transaction costs, particularly when dealing with large proofs. If the Merkle tree depth is known and constant, the use of a dynamic vector introduces inefficiency.

```
pub fn verify(proof: Vec<[u8; 32]>, root: [u8; 32], leaf: [u8; 32],
index: u32) -> bool {
    recompute(leaf, &proof, index) == root
}
```

Recommendation

It is recommended to use a fixed-size array for proof representation if the Merkle tree depth is known at compile time. This will eliminate unnecessary heap allocations and reduce memory usage and transaction costs. For dynamic-depth Merkle trees, consider optimising proof representation to balance flexibility and efficiency.

IASV - Insufficient Account Space Validation

Criticality	Minor / Informative
Location	create_distribution_tree.rs#L15
Status	Acknowledged

Description

The space allocation for the `distribution_tree` account depends on the value of `MAX_ORACLES_COUNT`. Specifically, the total size includes the size of `OracleReport` multiplied by `MAX_ORACLES_COUNT`. While this calculation ensures sufficient space for the maximum number of oracles, there is no runtime check to validate that `MAX_ORACLES_COUNT` is within a reasonable range. If this value is set too high, it could result in an account size that exceeds Solana's maximum limits or wastes resources. This may lead to transaction failures during account creation or inefficient resource usage.

```
space = 8 + 4 + std::mem::size_of::<DistributionTree>() +
std::mem::size_of::<OracleReport>() * MAX_ORACLES_COUNT,
```

Recommendation

It is recommended to add a runtime check to ensure that the number of oracles defined by `MAX_ORACLES_COUNT` falls within a safe and reasonable range. This will prevent the creation of excessively large accounts and ensure compatibility with Solana's maximum account size constraints, reducing the risk of transaction failures and resource wastage.

IOV - Insufficient Operation Validation

Criticality	Minor / Informative
Location	canopy.rs#L88
Status	Acknowledged

Description

The `fill_in_proof_from_canopy` and `get_cached_path_length` functions perform operations on the canopy data without sufficient validation of indices or input parameters. Specifically, indices such as `cached_idx` are used to access the canopy array without explicit bounds checks, and parameters like `max_depth` are not validated for extreme values or edge cases (e.g., zero-length canopy arrays or excessively large tree depths). This lack of input validation can result in out-of-bounds access, runtime panics, or undefined behaviour, potentially causing program crashes or memory corruption.

```
let cached_idx = if shifted_index % 2 == 0 {
    shifted_index + 1
} else {
    shifted_index - 1
};
if canopy[cached_idx] == EMPTY {
    ...
}
```

Recommendation

It is recommended to implement comprehensive bounds checks for all indices and parameters. Ensure that all array accesses (e.g., `cached_idx`) are within valid ranges. Additionally, validate input parameters such as `max_depth` to confirm they fall within logical and supported limits. These measures will enhance the program's resilience against invalid inputs and prevent runtime disruptions.

IODV - Insufficient Oracle Data Validation

Criticality	Minor / Informative
Location	activate_distribution_tree.rs#L34
Status	Acknowledged

Description

The contract only verifies that `oracle_choice()` returns a `Some` value without validating the contents or ensuring that the oracle data is correct or trustworthy. This lack of validation exposes the contract to potential manipulation or the inclusion of invalid data, potentially leading to an incorrect or vulnerable contract state.

```
require!(distributin_tree.oracle_choice().is_some(),
RdError::InvalidOracleReports);
```

Recommendation

It is recommended to add stricter validation checks on the value returned by `oracle_choice()`. Ensure that only trusted oracles provide data and that the data is validated against predefined rules or criteria to maintain contract integrity.

MDEM - Missing Descriptive Error Message

Criticality	Minor / Informative
Location	error.rs#L4
Status	Acknowledged

Description

The `RdError` enum defines various error codes that are integral to the program's error handling. However, all error messages in the `#[msg]` attributes are left empty. This omission makes it difficult for developers, auditors, or users to understand the root cause of an error when it occurs. Without descriptive messages, debugging issues becomes challenging, and the program loses a critical mechanism for communicating errors clearly and effectively. This lack of transparency can lead to inefficiencies and increased development or operational costs.

```
pub enum RdError {
    #[msg("")]
    Authorization,
    #[msg("")]
    InvalidAsset,
    #[msg("")]
    ...
}
```

Recommendation

It is recommended to provide clear and concise descriptions for each error in the `#[msg]` attributes. These messages should explain the nature of the error and, where applicable, provide guidance on resolving the issue. By adding meaningful error messages, the program can improve debugging efficiency, enhance user experience, and provide better documentation for its error codes.

MEH - Missing Error Handling

Criticality	Minor / Informative
Location	merkle_proof.rs#L18
Status	Acknowledged

Description

The `hash_to_parent` function directly uses the `hashv` function to compute the parent node of a Merkle tree without implementing any error handling. Although `hashv` is generally reliable, the absence of error handling means that any unexpected issues, such as invalid input data, could result in silent failures or runtime panics. This could disrupt the Merkle tree verification process and lead to incorrect results or undefined behaviour.

```
let parent = if is_left {
    hashv(&[node, sibling])
} else {
    hashv(&[sibling, node])
};
node.copy_from_slice(parent.as_ref());
```

Recommendation

It is recommended to implement error handling around the `hashv` operation to ensure that any issues are caught and handled gracefully. For instance, check the validity of the inputs before performing the hash operation and propagate meaningful error messages if the operation fails. This will enhance the robustness of the Merkle tree logic.

MIV - Missing Input Validation

Criticality	Minor / Informative
Location	merkle_proof.rs#L7
Status	Acknowledged

Description

The functions in this file lack explicit validation for their inputs. For example, the `proof` array is not validated to ensure that its length matches the expected depth of the Merkle tree, and the `index` is not checked to confirm it falls within the valid range `[0, 2^depth - 1]`. If the `proof` length is incorrect or the `index` is out of range, the `recompute` function could produce incorrect results or behave unexpectedly. This absence of input validation compromises the reliability of the Merkle tree verification process and increases the risk of logical errors.

```
pub fn recompute(leaf: Node, proof: &[Node], index: u32) -> Node {
    let mut current_node = leaf;
    for (depth, sibling) in proof.iter().enumerate() {
        hash_to_parent(&mut current_node, sibling, index >> depth & 1 ==
0);
    }
    current_node
}
```

Recommendation

It is recommended to validate the inputs before processing. Ensure that the `proof` array has the correct number of nodes for the expected Merkle tree depth and that the `index` lies within the valid range. Adding these checks will improve the reliability of the Merkle tree operations and safeguard against undefined behaviour caused by invalid inputs.

MNLC - Missing Name Length Check

Criticality	Minor / Informative
Location	create_distributor.rs#L85
Status	Acknowledged

Description

The `args.name` field is used as a seed in the PDA derivation for the `distributor` account. While there is a validation check to ensure the name length does not exceed 32 bytes, this validation occurs separately from the PDA derivation logic. This introduces a potential risk where if the name length exceeds 32 bytes, PDA derivation could fail unexpectedly. Such failures could lead to inconsistencies and disruptions in the distributor initialization process.

```
require!(args.name.len() <= 32, RdError::InvalidDistributorName);

seeds = ["distributor".as_bytes(), realm.key().as_ref(),
rewards_mint.key().as_ref(), args.name.as_bytes()],
```

Recommendation

It is recommended to directly enforce or truncate the length of `args.name` before using it as a seed for PDA derivation. This ensures that PDA derivation will always succeed, even if the name length approaches its maximum limit. By validating or truncating the name length as part of the seed generation process, the program can maintain consistent and reliable behaviour.

OBA - Out of Bounds Access

Criticality	Minor / Informative
Location	canopy.rs#L83
Status	Acknowledged

Description

The `fill_in_proof_from_canopy` function computes indices like `node_idx` and `cached_idx` to access elements in the canopy array. These indices are derived from input parameters and calculations but are not explicitly checked against the bounds of the array. If these indices exceed the size of the canopy array, it could lead to runtime panics or invalid memory access. This issue disrupts program execution and poses a significant risk, especially if the function is invoked with unexpected or malicious inputs.

```
let mut node_idx = ((1 << max_depth) + index) >> (max_depth - path_len);
let shifted_index = node_idx as usize - 2;
let cached_idx = if shifted_index % 2 == 0 {
    shifted_index + 1
} else {
    shifted_index - 1
};
if canopy[cached_idx] == EMPTY {
    ...
}
```

Recommendation

It is recommended to add explicit bounds checks before accessing the canopy array. Validate that `node_idx` , `shifted_index` , and `cached_idx` are within the valid range of indices for the canopy. This will prevent runtime panics and ensure the function operates safely, even when handling edge cases or unexpected inputs. Comprehensive bounds validation will improve the reliability and robustness of canopy-related operations.

PVF - Period Validation Flaw

Criticality	Minor / Informative
Location	report_oracle.rs#L31
Status	Acknowledged

Description

The `require!` statement in the `report_oracle` function ensures that the `distribution_tree.period` is greater than the `distributor.current_period`. While this validation prevents reports from being submitted for the current or past periods, it does not verify whether `distribution_tree.period` aligns with the expected next period (`distributor.current_period + 1`). A misconfigured `distribution_tree` could allow reports to be accepted for unintended or incorrect periods, disrupting the program's logic.

```
require!(
    distribution_tree.period > distributor.current_period,
    RdError::CannotReportAtPresent
);
```

Recommendation

It is recommended to enhance the validation logic to ensure that `distribution_tree.period` matches the expected next period (`distributor.current_period + 1`). This additional check will prevent misconfigured `distribution_tree` accounts from being used and ensure that reports are only accepted for the intended periods.

PIO - Potential Integer Overflow

Criticality	Minor / Informative
Location	close_canopy.rs#L7
Status	Acknowledged

Description

The `checked_add` operation is used to safely add lamports to the `sol_destination` account, preventing overflow. However, the `unwrap()` call following this operation assumes that the addition will never fail. If an overflow occurs, the `unwrap()` call will panic, disrupting the program's execution. Although this does not present a direct security vulnerability in this specific case, it could cause unexpected behaviour or transaction failure under certain conditions.

```
**sol_destination.lamports.borrow_mut() =
    dest_starting_lamports.checked_add(info.lamports()).unwrap();
```

Recommendation

It is recommended to replace the `unwrap()` call with explicit error handling. This will ensure that the program gracefully handles cases where the addition fails, avoiding unexpected panics and improving the contract's robustness.

UB - Unchecked Borrow

Criticality	Minor / Informative
Location	claim_rewards.rs#L165
Status	Acknowledged

Description

The `canopy_data` account is borrowed using `try_borrow_data()` without a fallback mechanism. If the borrow fails due to data corruption or concurrency issues, it could cause a runtime panic.

```
if !ctx.accounts.canopy_data.data_is_empty() {
    fill_in_proof_from_canopy(
        &ctx.accounts.canopy_data.try_borrow_data()?,
        oracle_report.max_depth,
        args.distribution_args.index,
        &mut distribution_tree_proof,
    )?;
}
```

Recommendation

It is recommended to add error handling for the `try_borrow_data()` call to gracefully handle scenarios where the account data cannot be borrowed. Return a meaningful error instead of causing a runtime panic.

VNT - Variable Naming Typo

Criticality	Minor / Informative
Location	activate_distribution_tree.rs#L26
Status	Acknowledged

Description

The contract contains a typo in the variable name `distributin_tree` instead of the correct `distribution_tree`. This inconsistency could lead to confusion among users or developers and increase the likelihood of introducing bugs during future modifications or debugging.

```
let distributin_tree = &ctx.accounts.distribution_tree;

require_eq!(
    distributin_tree.period,
    distributor.current_period + 1,
    RdError::IllegalPeriod
);
```

Recommendation

It is recommended to fix the typo by renaming the variable `distributin_tree` to `distribution_tree` for consistency and clarity.

Summary

The Reward Distributor contract implements a decentralized reward distribution mechanism leveraging Merkle tree proofs, compressed NFTs, and governance controls to allocate tokens to eligible NFT holders securely. This audit investigates potential security vulnerabilities, business logic flaws, and areas for scalability, fraud prevention, and operational efficiency improvement. The team has acknowledged the findings.

Disclaimer

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.

The Cyberscope team

cyberscope.io