Cyberscope

Audit Report

StoryChain

April 2024

Repository	https://github.com/storychainai/storychain_contracts
Address	733a02565c99f7d9d396586ae26c2705c294bab7
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/storychainai/storychain_contracts
Commit	733a02565c99f7d9d396586ae26c2705c294bab7

Audit Updates

Initial Audit	19 Apr 2024

Source Files

Filename	SHA256
StoryChain.sol	a4193e3cb144fb2171481cdd0542fdc26db625391b7c6573ebeb1caedf6462c8
STRYToken.sol	fbcec02a1a5ef5488d737a7fd91384c3ccdaaea031e0a35699627195ebded1d2
DividendReceiver.sol	feeef4108de2f53a952216c87b4aa5de70a86fba054bda9babaf64ac49749609

Overview

StoryChain contract

The StoryChain contract is described as a decentralized platform for creating and managing stories, particularly with a focus on integrating artificial intelligence and community interaction in the storytelling process. The functionalities are broadly categorized into story creation, competition management, and administrative tasks.

Story Creation and Management

At its core, the contract allows users to create stories using various inputs such as tokens for payment (supporting cryptocurrency and BNB), prompts for AI models, titles, and types of stories ranging from collaborative efforts to competitions. Each story creation function (`createStory` and `createWhitelistedStory`) logs details like the AI model and style to be used, and the number of pages intended for the story. Notably, the `createWhitelistedStory` function includes an additional feature to specify which users can continue the story, enhancing privacy and exclusivity for special projects. The story progression is facilitated through functions like `continueStory` for ongoing stories and `enterCompetition` for stories designated as competitions, emphasizing flexibility in how narratives are developed and expanded upon by the community.

Competition Handling

The platform uniquely incorporates features to manage story-based competitions. This is evident from functions like `createCompetition` , which sets up a new competitive story event including parameters such as duration, entry fee, and rewards distribution among voters and the treasury. Competitions are meticulously managed through functions that allow for the continuation of stories within the competitive framework (`enterCompetition`), and the execution of competitions (`executeCompetition`) which includes distributing rewards based on off-chain voting results verified by a Merkle tree proof. This dual approach of on-chain management with off-chain voting ensures a scalable and flexible system for community-driven story competitions.

Administrative Functions

Administrative capabilities are robust, involving settings adjustments and operational controls by authorized personnel only. This includes setting token allowances, adjusting financial parameters like fees and reward distributions (`setStoryFromBnbRate` , `setTokenAllowed`), and critical functions like `claimTreasury` for fund management. These administrative functions ensure that the platform remains adaptable and secure, capable of evolving with its user base and the broader blockchain ecosystem.

DividendReceiver contract

The DividendReceiver abstract contract, outlines a comprehensive system designed for managing dividends, treasury functions, and reward distributions within a decentralized platform. The contract leverages the ERC20 standard for safe token handling and incorporates advanced structures and error handling mechanisms to maintain robust financial interactions among users.

Dividend and Treasury Management

The "DividendReceiver" abstract contract is designed to manage dividends and treasury operations within a blockchain-based platform, ensuring equitable distribution of dividends based on user holdings and secure treasury management. It incorporates features such as real-time dividend calculation, secure token handling using the SafeERC20 library, and robust error handling to maintain financial integrity. Through functions that allow users to claim dividends, view treasury balances, and handle funds securely, the contract supports a transparent and fair economic environment, crucial for building trust and stability in decentralized platforms.

Roles

DEFAULT_ADMIN_ROLE

The DEFAULT_ADMIN_ROLE can interact with the following functions:

- function claimTreasury

SETTER_ROLE

The SETTER_ROLE can interact with the following functions:

- function setStoryFromBnbRate
- function setTokenAllowed
- function setFee
- function setNftShare
- function setSystemCostFee
- function setConvertFeesToStry
- function setBaseURI
- function skipNonce
- function migrateNFT

OPERATOR_ROLE

The OPERATOR_ROLE can interact with the following functions:

- function updateMultiple
- function createCompetition
- function executeCompetition

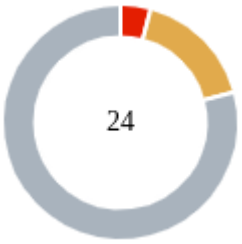
Users

The users can interact with the following functions:

- function createStory
- function createWhitelistedStory
- function enterCompetition
- function continueStory
- function claimVotingReward

- function claimDividend
- function injectTreasury

Findings Breakdown



Critical	1
Medium	4
Minor / Informative	19

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	4	0	0	0
Minor / Informative	19	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	UCC	Unauthorized Competition Creation	Unresolved
●	BCC	Bypassed Competition Checks	Unresolved
●	IRD	Incomplete Reward Distribution	Unresolved
●	IRC	Inconsistent Rejection Criteria	Unresolved
●	ITV	Inefficient Token Validation	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IAH	Inadequate Accuracy Handling	Unresolved
●	ISV	Inadequate StoryType Validation	Unresolved
●	MPC	Merkle Proof Centralization	Unresolved
●	MFN	Misleading Function Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MNV	Missing Nonce Verification	Unresolved
●	MTV	Missing Token Validation	Unresolved
●	MU	Modifiers Usage	Unresolved

●	PBV	Percentage Boundaries Validation	Unresolved
●	PUR	Potential Underflow Risk	Unresolved
●	PARS	Potentially Arbitrary Rate Set	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	UED	Unutilized Error Declarations	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

UCC - Unauthorized Competition Creation

Criticality	Critical
Location	StoryChain.sol#L268,293
Status	Unresolved

Description

The contract currently allows users to create a story with `storyType` set to 4, which corresponds to a `Competition`, using the `createStory` or `createWhitelistedStory` functions. These functions can be accessed by any user as long as the token used is allowed (tokenIsAllowed modifier), and crucially, it does not restrict the number of pages (numPages) that can be set for a Competition story. As a result, a user can bypass the intended administrative control and set up a Competition with just one page, which is inconsistent with the typical use case envisioned for Competitions, generally expected to be initiated through the `createCompetition` function by an operator with specific permissions.

```
    function createStory(
        address token,
        string calldata prompt,
        string calldata title,
        uint64 storyType,
        uint64 imageAIid,
        uint64 imageStyleId,
        uint64 numPages
    ) external payable tokenIsAllowed(token) {
        _createStory(
            token,
            prompt,
            title,
            storyType,
            imageAIid,
            imageStyleId,
            numPages
        );
    }

    function createWhitelistedStory(
        address token,
        string calldata prompt,
        string calldata title,
        uint64 storyType,
        uint64 imageAIid,
        uint64 imageStyleId,
        uint64 numPages,
        address[] calldata whitelist
    ) external payable tokenIsAllowed(token) {
        ...

        _createStory(
            token,
            prompt,
            title,
            storyType,
            imageAIid,
            imageStyleId,
            numPages
        );
    }
```

Recommendation

It is recommended to introduce safeguards within the `_createStory` function to prevent the unauthorized creation of Competition stories by general users. Specifically, adding a check to ensure that storyType 4 (Competition) can only be selected if the function caller has the appropriate role (e.g., OPERATOR_ROLE) would be prudent. This can be implemented by extending the existing role-checking mechanisms to this function or by adding a conditional statement that explicitly checks the user's role when `storyType` is set to 4. These measures will ensure that only authorized personnel can initiate Competitions, thereby maintaining the integrity and the controlled environment intended for such events.

BCC - Bypassed Competition Checks

Criticality	Medium
Location	StoryChain.sol#L901
Status	Unresolved

Description

The contract currently permits users to both create a story with the `COMPETITION` type and subsequently invoke the `enterCompetition` function directly. This design allows the `enterCompetition` function to execute successfully without the necessary checks to verify whether the story indeed qualifies as a competition under the correct and intended circumstances. Typically, such validations include confirming that the story has been set up by an authorized operator and meets specific criteria for a competition. The absence of these checks could lead to improper use of the competition mechanism, potentially resulting in unauthorized entries and disruptions in the intended flow of story competitions.

```
    function _continueStory(
        address token,
        string calldata prompt,
        uint256 storyNonce
    ) internal nonReentrant {
        Story storage story = s_stories[storyNonce];
```

Recommendation

It is recommended to implement comprehensive validations within the `enterCompetition` function to ensure that only stories that have been properly set up and authorized as competitions can be accessed and entered by users. Additionally, modifying the `createStory` function to restrict the creation of stories with the `COMPETITION` type to authorized users only would further strengthen the process and ensure compliance with the intended competitive framework. These changes will help maintain the integrity of competition stories and prevent the potential exploitation of the system.

IRD - Incomplete Reward Distribution

Criticality	Medium
Location	StoryChain.sol#L545
Status	Unresolved

Description

The contract is designed to distribute a `totalReward`, which is the sum of `baseReward` and `collectedFees`, among various stakeholders according to specified percentages (voters, treasury, prize winners). However, if the combined percentages allocated to voters, the treasury, and prize winners do not sum to 100%, the remaining portion of the `totalReward` is not redistributed or returned but remains unallocated within the contract. This could result in funds being locked within the contract,unable to be re-distributed.

```
uint totalReward = competition.baseReward +
competition.collectedFees;
competition.perVoterReward = uint64(
    ((totalReward * competition.votersPercentage) / 100) /
numVoters
);

_addTreasury(
    competition.token,
    (totalReward * competition.treasuryPercentage) / 100
);

for (uint i = 0; i < nftIds.length; i++) {
    cumilativePercentage += prizePercentages[i];
    if (cumilativePercentage > 100) {
        revert StoryChain__PercentagesCantBeMoreThan100();
    }
    uint256 nftId = nftIds[i];
    address winner = _ownerOf(nftId);
    if (winner == address(0)) revert StoryChain__NoSuchNFT();
    //SET winner and nfts as arrays
    competition.winner = winner;
    competition.winnerNft = uint128(nftIds[i]);
```

Recommendation

It is recommended to include a validation check in the `executeCompetition` function to ensure that the sum of the percentages allocated for voters, treasury, and prize winners equals 100%. This check can prevent the scenario where funds are unintentionally withheld from distribution. Additionally, it could be beneficial to implement a fallback mechanism to handle any unallocated rewards, such as redistributing them back or rolling them over to the next competition. This approach will enhance the fairness and transparency of the reward distribution process and ensure that all allocated funds are handled appropriately.

IRC - Inconsistent Rejection Criteria

Criticality	Medium
Location	StoryChain.sol#L990
Status	Unresolved

Description

The contract includes the `_rejectPrompt` function, which is designed to reject stories based on certain conditions. Specifically, the function rejects any story that either has no pages or is not of the `COMPETITION` or `COLLAB` type. This creates a significant inconsistency since `COMPETITION` and `COLLAB` stories that have more than one page are exempt from rejection, while other story types with more than one page can still be rejected. This inconsistency can lead to unfair treatment of different story types and might not align with the intended operational logic of the contract.

```
if (
    story.pages == 0 ||
    (story.storyType != COMPETITION_STORY_TYPE &&
        story.storyType != COLLAB_STORY_TYPE)
) story.creationRejected = true;
```

Recommendation

It is recommended to standardize the rejection criteria across all story types by modifying the `_rejectPrompt` function to apply consistent conditions for rejection. This could be done by either making the number of pages a non-factor in the rejection process or by ensuring that all story types, including `COMPETITION` and `COLLAB`, are evaluated under the same page count conditions. Implementing uniform rules for rejection will enhance fairness, transparency, and user trust by ensuring that all stories are treated equitably regardless of their type. This change will align the contract's operations with its intended logic and prevent any potential biases in story processing.

ITV - Inefficient Token Validation

Criticality	Medium
Location	StoryChain.sol#L326
Status	Unresolved

Description

The contract is currently employing a simple if statement to validate the token address in the `enterCompetition` function. While this checks that the token address matches the one associated with the `storyNonce`, it does not verify whether the `storyNonce` itself represents a legitimate entry that was initialized correctly via the `createCompetition` function. This could potentially allow users to pass parameters that appear legitimate but may not actually correlate with a valid competition context. This oversight in validation could lead to unauthorized access or misuse of the competition entering process, as users might exploit this loophole to pass arbitrary or incorrect `storyNonce` values that the contract will treat as legitimate.

```
    function enterCompetition(
        address token,
        string calldata prompt,
        uint256 storyNonce,
        uint64 imageAIid,
        uint64 imageStyleId
    ) external payable tokenIsAllowed(token) {
        if (s_competition[storyNonce].token != token)
            revert StoryChain__CompetitionTokenIsDifferent();
```

Recommendation

It is recommended to reconsider the validation strategy for the parameters passed to the `enterCompetition` function. Specifically, the contract should include checks to ensure that the `storyNonce` provided by the user corresponds to a nonce that was indeed initialized through the `createCompetition` function and that the associated token address matches the one stored in the competition record. This could involve implementing a mapping to track valid `storyNonce` values or adding additional conditions in the validation logic to verify both the `storyNonce` and the token comprehensively. Such

measures will enhance security and integrity by preventing the misuse of the function with invalid competition entries.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	StoryChain.sol#L416,437,661
Status	Unresolved

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.

```
   function setStoryFromBnbRate(uint _rate) external
onlyRole(SETTER_ROLE) {
        s_stryFromBnbRate = _rate;
        emit SetSTRYRate(_rate);
    }

    function updateMultiple(
        uint[] calldata promptEntryIds,
        uint[] calldata rejectPromptEntryIds
    ) external onlyRole(OPERATOR_ROLE) {
        for (uint i = 0; i < promptEntryIds.length; i++) {
            _updateStory(promptEntryIds[i]);
        }
        for (uint i = 0; i < rejectPromptEntryIds.length; i++)
{
            _rejectPrompt(rejectPromptEntryIds[i]);
        }
    }

  function migrateNFT(
        uint256 migrationId,
        address author,
        string memory _title,
        uint _storyType,
        uint _imageAIid,
        uint _imageStyleId
    ) external onlyRole(SETTER_ROLE) {
```

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.

IAH - Inadequate Accuracy Handling

Criticality	Minor / Informative
Location	StoryChain.sol#L576
Status	Unresolved

Description

The contract is set up to distribute a `totalReward` among users, including a proportion to winners based on predefined percentages. However, the calculation of these rewards does not consider decimal places accurately due to the use of integer arithmetic, which truncates decimals. This could lead to discrepancies in the distribution, especially noticeable as the number of winners increases. Without proper handling of decimal values, smaller reward portions might not be allocated correctly, potentially leading to undistributed funds or minor underpayments to winners.

```
for (uint i = 0; i < nftIds.length; i++) {
    cumilativePercentage += prizePercentages[i];
    if (cumilativePercentage > 100) {
        revert StoryChain__PercentagesCantBeMoreThan100();
    }
    uint256 nftId = nftIds[i];
    address winner = _ownerOf(nftId);
    if (winner == address(0)) revert StoryChain__NoSuchNFT();
    //SET winner and nfts as arrays
    competition.winner = winner;
    competition.winnerNft = uint128(nftIds[i]);
    _addBalance(
        competition.token,
        winner,
        (totalReward * prizePercentages[i]) / 100
    );
}
```

Recommendation

It is recommended to incorporate better precision in the reward calculations to handle decimals accurately. This could be achieved by modifying the reward distribution logic to include fixed-point arithmetic or by utilizing a more refined mathematical approach, such as

the use of a library designed for safe mathematical operations with support for decimals. Additionally, implementing a rounding mechanism to ensure that total payouts correctly sum up to the `totalReward` can help in maintaining the integrity and fairness of the distribution process. This adjustment will ensure that all participants receive their rightful share of the rewards, irrespective of the total number of winners or the granularity of the percentage allocations.

ISV - Inadequate StoryType Validation

Criticality	Minor / Informative
Location	StoryChain.sol#L263,293
Status	Unresolved

Description

The contract is currently lacking a validation check for the `storyType` parameter within the `createStory` and `createWhitelistedStory` functions. This parameter, which is supposed to accept predefined values (0: collab, 1-2: solo, 3: love letter, 4: competition), does not have any enforcement mechanism to ensure that the values passed to it are within these accepted limits. As a result, users could potentially pass a value outside of this range, which may not be handled correctly by the contract's logic, leading to unpredictable behaviors or errors in the story creation process.

```
    * @param   storyType  0: collab, 1-2: solo, 3: love letter, 4:
competition
     * @param   imageAIid  index of imageAI, to be passed to AI model
     * @param   imageStyleId  index of imageStyle, to be passed to AI
model
     * @param   numPages  number of pages to be created
     */
    function createStory(
        address token,
        string calldata prompt,
        string calldata title,
        uint64 storyType,
    ...
    function createWhitelistedStory(
        address token,
        string calldata prompt,
        string calldata title,
        uint64 storyType,
```

Recommendation

It is recommended to add additional checks at the beginning of the functions to validate that the `storyType` parameter is within the acceptable range (0 to 4). This can be efficiently achieved by using a simple conditional statement. Implementing this validation will prevent the submission of invalid numbers for the `storyType`, enhancing the robustness and reliability of the contract. This change will ensure that all story creation requests conform to the intended usage scenarios and maintain the contract's integrity.

MPC - Merkle Proof Centralization

Criticality	Minor / Informative
Location	StoryChain.sol#L366,527
Status	Unresolved

Description

The contract uses a Merkle Proof mechanism in order to define many applicable addresses. The verification process is based on an off-chain configuration. The contract owner is responsible for updating the in-chain “Merkle Root” in order to validate correctly the provided message.

Additionally the `numVoters` varaible is calculated off-chain and passed as parameter instead of a built-in contract variable which stores the corresponding variable.

```
function claimVotingReward(
        uint competitionNonce,
        bytes32[] memory proof
    ) external {
        ...
        if (
            !MerkleProof.verify(
                proof,
                s_competitionProofs[competitionNonce],
                leaf
            )
        ) revert StoryChain__ProofFailed();

        ...
    }
 function executeCompetition(
        uint256 storyNonce,
        uint256[] calldata nftIds,
        uint256[] calldata prizePercentages,
        bytes32 proofRoot,
        uint256 numVoters
    )
        external
        onlyRole(OPERATOR_ROLE)
    {
        ...

        uint cumilativePercentage =
competition.treasuryPercentage +
            competition.votersPercentage;
        s_competitionProofs[storyNonce] = proofRoot;
```

Recommendation

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasize that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MFN - Misleading Function Naming

Criticality	Minor / Informative
Location	StoryChain.sol#L440
Status	Unresolved

Description

The contract is using a misleading function naming conventions which may confuse users. The function `_updateStory` , used within the `updateMultiple` method, is essentially designed to accept and process Story identifiers (IDs). However, the name `_updateStory` might imply a broader or different functionality, such as updating various attributes of a story, whereas its actual implementation is focused solely on processing an ID for an update mechanism. This could lead to misunderstandings about the function's purpose and misuse in the context of the contract's operations.

```
for (uint i = 0; i < promptEntryIds.length; i++) {
    _updateStory(promptEntryIds[i]);
}
for (uint i = 0; i < rejectPromptEntryIds.length; i++) {
    _rejectPrompt(rejectPromptEntryIds[i]);
```

Recommendation

It is recommended to rename the `_updateStory` function to more accurately reflect its purpose and functionality. A suggested new name could be `_acceptStory` , which clearly indicates that the function handles story updates by their IDs. This renaming would enhance code readability and clarity, reducing the potential for errors or misinterpretation by future developers or during further code audits. Additionally, updating the documentation to describe what the function specifically does can further prevent confusion.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	StoryChain.sol#L366,421,527,634
Status	Unresolved

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.

```
    function claimVotingReward(
        uint competitionNonce,
        bytes32[] memory proof
    ) external {
        ...
    }

    function setTokenAllowed(
        address token,
        bool state
    ) external onlyRole(SETTER_ROLE) {
        s_allowedTokens[token] = state;
    }

    function executeCompetition {
    ...
    }

    function skipNonce(uint _nonce) external
onlyRole(SETTER_ROLE) {
        if (_nonce < s_nonce) revert
StoryChain__CantSkipToOldNonce();
        s_nonce = _nonce;
    }
```

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.

MNV - Missing Nonce Verification

Criticality	Minor / Informative
Location	StoryChain.sol#L350
Status	Unresolved

Description

The contract is currently failing to perform a necessary check to verify the existence of the `storyNonce` before proceeding with the execution of the `continueStory` function. This omission allows for the possibility that the function could be invoked with an invalid `storyNonce`, which does not correspond to any previously initialized story context. Such an oversight can lead to various issues, including the execution of functions with non-existent data references, potentially causing errors or inconsistencies within the contract's operation and state management.

```
    function continueStory(
        address token,
        string calldata prompt,
        uint256 storyNonce
    ) external payable tokenIsAllowed(token) {
        _continueStory(token, prompt, storyNonce);
    }
```

Recommendation

It is recommended to add additional checks within the `continueStory` function to ensure that the `storyNonce` passed as a parameter has been legitimately established in the contract's state. This could be implemented by validating the `storyNonce` against a stored record of valid nonces, such as using a mapping to track active or initialized nonces. This check will safeguard against potential misuse and enhance the integrity of the function's execution.

MTV - Missing Token Validation

Criticality	Minor / Informative
Location	StoryChain.sol#L464
Status	Unresolved

Description

The contract is currently not enforcing a check to verify the legitimacy of the token address used in the `createCompetition` function. While the `enterCompetition` function includes the `tokenIsAllowed` modifier to ensure that only permitted tokens are used, the `createCompetition` function allows any token address to be set, potentially including those that are not approved or recognized by the contract. This inconsistency could lead to scenarios where competitions are created with tokens that cannot be used or accepted in subsequent operations, leading to errors and complications in managing competition entries and rewards.

```
    function createCompetition(
        ...
        address token,
        uint rewardAmount
    )
        external
        payable
        onlyRole(OPERATOR_ROLE)
    {
```

Recommendation

It is recommended to use the `tokenIsAllowed` modifier within the `createCompetition` function to maintain consistency and security across the contract. Implementing this check will ensure that all tokens used within the competitions are vetted and approved, thus preventing the use of unauthorized tokens that could affect the competition's integrity and the contract's overall functionality. This adjustment will help in upholding the contract's standards and in avoiding potential disputes or operational issues related to token usage.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	StoryChain.sol#L334,371,540
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
if (s_competition[storyNonce].token != token)
    revert StoryChain__CompetitionTokenIsDifferent();
if (s_votingRewardClaimed[competitionNonce][msg.sender])
    revert StoryChain__AlreadyClaimed();
if (s_competition[competitionNonce].perVoterReward == 0)
    revert StoryChain__NoRewardToClaim();
if (s_stories[storyNonce].storyType != COMPETITION_STORY_TYPE)
    revert StoryChain__NotACompetition();
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	StoryChain.sol#L498,499,604
Status	Unresolved

Description

The contract utilizes variables for percentage-based calculations that are required for its operations. These variables are involved in multiplication and division operations to determine proportions related to the contract's logic. If such variables are set to values beyond their logical or intended maximum limits, it could result in incorrect calculations. This misconfiguration has the potential to cause unintended behavior or financial discrepancies, affecting the contract's integrity and the accuracy of its calculations.

```
competition.votersPercentage = votersPercentage;
competition.treasuryPercentage = treasuryPercentage;
...
s_feeNftShare = _nftShare;
```

Recommendation

To mitigate risks associated with boundary violations, it is important to implement validation checks for variables used in percentage-based calculations. Ensure that these variables do not exceed their maximum logical values. This can be accomplished by incorporating `require` statements or similar validation mechanisms whenever such variables are assigned or modified. These safeguards will enforce correct operational boundaries, preserving the contract's intended functionality and preventing computational errors.

PUR - Potential Underflow Risk

Criticality	Minor / Informative
Location	StoryChain.sol#L1026
Status	Unresolved

Description

The contract is vulnerable to an underflow condition in the `_dealFees` function where the `promptEntry.fees` is subtracted by `competition.entryFee`. If `competition.entryFee` is set to a value higher than `promptEntry.fees`, it could result in an underflow, leading to incorrect and excessively high fee values being processed. This arithmetic underflow can manipulate the fee logic unpredictably, potentially causing significant issues in fee handling and distribution within the contract's operations.

```
_dealFees(
    promptEntry.token,
    promptEntry.fees - competition.entryFee
);
```

Recommendation

It is recommended to implement an explicit check before the subtraction operation to ensure that `promptEntry.fees` is always greater than or equal to `competition.entryFee`. This can be achieved by adding a conditional statement such as require `(promptEntry.fees >= competition.entryFee, "Entry fee exceeds prompt fee");` prior to the subtraction. This precaution will prevent underflow errors and ensure that fee calculations are handled securely and accurately.

PARS - Potentially Arbitrary Rate Set

Criticality	Minor / Informative
Location	StoryChain.sol#L415
Status	Unresolved

Description

The contract is currently structured to allow addresses with the `SETTER_ROLE` to change the `s_stryFromBnbRate`, which dictates the BNB/STRY exchange rate. This setup poses significant risks as it grants the capability to potentially manipulate the fee values arbitrarily. Since the rate can dramatically influence the economic parameters of the contract, such as transaction costs and token valuation, the ability to alter this rate without stringent checks or balances can lead to abuse of power, market manipulation, or unintentional errors impacting the system's economic stability.

```
    function setStoryFromBnbRate(uint _rate) external
onlyRole(SETTER_ROLE) {
        s_stryFromBnbRate = _rate;
        emit SetSTRYRate(_rate);
    }
```

Recommendation

It is recommended to introduce additional safeguards and oversight mechanisms for setting the `s_stryFromBnbRate` exchange rate. To mitigate potential risks of arbitrary or malicious rate settings, the implementation of a multi-signature requirement for this function could be beneficial. Alternatively, transitioning to a decentralized oracle for rate determination could provide a more robust and tamper-resistant mechanism for rate updates. This change aims to ensure that rate adjustments are made more transparently and based on reliable, market-reflective data, thereby protecting the system from manipulation and promoting fair economic practices.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	StoryChain.sol#L198
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event UpdateOperator(address indexed operator, bool status);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	StoryChain.sol#L421
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setTokenAllowed(
        address token,
        bool state
    ) external onlyRole(SETTER_ROLE) {
        s_allowedTokens[token] = state;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	StoryChain.sol#L789
Status	Unresolved

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.

The contract is programmed to handle operations under the assumption that the tokens represented in its records are available in the actual token balances. However, discrepancies are arising because the contract's internal functions, such as `_addTreasury` and `_removeTreasury`, modify the theoretical token distribution without verifying against the actual on-chain balances. This can lead to situations where the contract assumes more tokens are available for distribution or withdrawal than what is truly present, potentially resulting in failed transactions or financial inconsistencies if the token supply is lower than expected.

```
_addTreasury(token, amountToShare);
addDividend(address(i_stryToken), nftRewards *
s_stryFromBnbRate);
_removeTreasury(
    address(i_stryToken),
    nftRewards * s_stryFromBnbRate
);
```

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 presale 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.

It is recommended to implement validation checks within the treasury-related functions to verify that the actual token balances align with the contract's internal representations before executing any additions or deductions. This could involve querying the token balance with a call to the token contract’s balance function prior to executing transactions that affect the treasury. By ensuring that sufficient tokens are indeed available and that the records accurately reflect the real state, the contract can avoid execution errors and maintain financial accuracy and integrity. This proactive approach will help prevent transaction failures and safeguard the contractual operations from discrepancies between recorded and actual token amounts.

UED - Unutilized Error Declarations

Criticality	Minor / Informative
Location	StoryChain.sol#L59
Status	Unresolved

Description

The contract declares custom errors which are not being utilized anywhere within the contract's logic. The contract's codebase indicates that the error is defined but never actually used in any of the function calls or condition checks. This leads to unnecessary bloating of the contract’s bytecode, potentially increasing deployment and runtime costs due to increased gas usage. Furthermore, the presence of unused code such as this can lead to confusion and misinterpretation of the contract's intended logic and error handling capabilities.

```
error StoryChain__LessThanFee();
error StoryChain__StoryOnlyAcceptsProposal();
error StoryChain__NotOpenToProposals();
error StoryChain__NoProposalOnThisIndex();
error StoryChain__VotingPeriodIsNotOver();
error StoryChain__NoProposalsForTheStory();
error StoryChain__EnteredPageIsNotCurrentPage();
error StoryChain__TransferFailed();
error StoryChain__NotAnOpenEndingStory();
error StoryChain__NotAClosedStory();
error StoryChain__NotCorrectNumOfPages();
error StoryChain__UseEnterCompetition();
```

Recommendation

It is recommended to remove the error declarations from the contract if there are no future plans to incorporate it into the contract’s logic. This action will help in optimizing the contract by reducing the bytecode size, thereby potentially lowering the costs associated with deploying and interacting with the contract. Moreover, cleaning up unused code will improve the overall readability and maintainability of the contract’s codebase. If there is a

possibility that this error could be used in future implementations, it is advised to document its intended use clearly to avoid any confusion.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	StoryChain.sol#L155,156,157,158,159,160,161,162,163,164,165,166,167,169,170,171,173,174,400,554,564,572,580,590,599,612,632,633,634,635,696,737 DividendReceiver.sol#L63,64,65,87,103,171,186
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 public s_fee
uint256 public s_feeSystemCost
uint256 public s_feeNftShare
uint256 public s_nonce
uint256 public s_promptIndex
string public s_baseURI
bool public s_convertFeesToStry
mapping(uint256 => Story) public s_stories
mapping(uint256 => PromptEntry) public s_promptEntries
mapping(uint256 => bool) public s_storyWhitelisted
mapping(uint256 => mapping(address => bool)) public
s_storyWhitelist
mapping(uint => mapping(address => bool)) public
s_votingRewardClaimed
mapping(uint256 => bool) public s_migrations
mapping(uint256 => Competition) public s_competition

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	StoryChain.sol#L1038 DividendReceiver.sol#L151,202,227,234,242
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _increaseBalance(
    address account,
    uint128 value
  ) internal override(ERC721Upgradeable,
ERC721EnumerableUpgradeable) {
    super._increaseBalance(account, value);
  }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	StoryChain.sol#L751,758,759
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 nftRewards = (amountToShare * s_feeNftShare) / 100
_removeTreasury(address(i_stryToken), nftRewards *
s_stryFromBnbRate)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

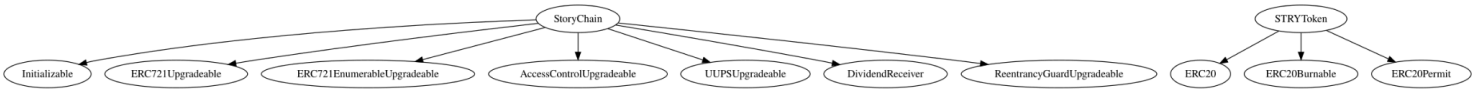
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
StoryChain	Implementation	Initializable, ERC721Upgradeable, ERC721EnumerableUpgradeable, AccessControlUpgradeable, UUPSUpgradeable, DividendReceiver, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createStory	External	Payable	tokenIsAllowed
	createWhitelistedStory	External	Payable	tokenIsAllowed
	enterCompetition	External	Payable	tokenIsAllowed
	continueStory	External	Payable	tokenIsAllowed
	claimVotingReward	External	✓	-
	setStoryFromBnbRate	External	✓	onlyRole
	setTokenAllowed	External	✓	onlyRole
	updateMultiple	External	✓	onlyRole
	createCompetition	External	Payable	onlyRole
	executeCompetition	External	✓	onlyRole
	setFee	External	✓	onlyRole
	setNftShare	External	✓	onlyRole

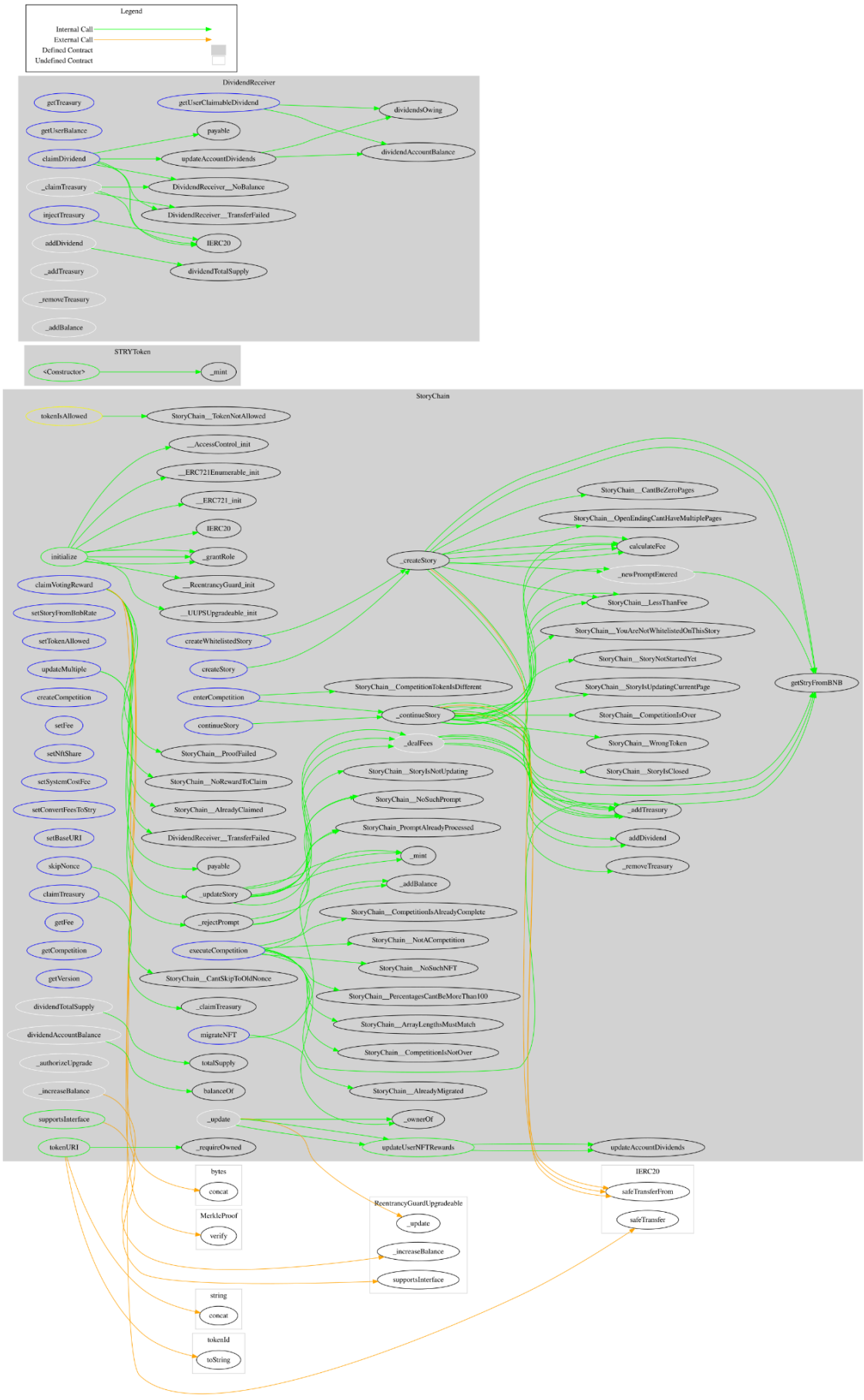
	setSystemCostFee	External	✓	onlyRole
	setConvertFeesToStry	External	✓	onlyRole
	setBaseURI	External	✓	onlyRole
	skipNonce	External	✓	onlyRole
	claimTreasury	External	✓	onlyRole
	migrateNFT	External	✓	onlyRole
	getFee	External		-
	getCompetition	External		-
	getVersion	External		-
	updateUserNFTRewards	Public	✓	-
	calculateFee	Internal		
	getStryFromBNB	Internal		
	dividendTotalSupply	Internal		
	dividendAccountBalance	Internal		
	_dealFees	Internal	✓	
	_newPromptEntered	Internal	✓	
	_createStory	Internal	✓	nonReentrant
	_continueStory	Internal	✓	nonReentrant
	_rejectPrompt	Internal	✓	
	_updateStory	Internal	✓	
	_update	Internal	✓	
	_authorizeUpgrade	Internal	✓	onlyRole
	_increaseBalance	Internal	✓	

	tokenURI	Public		-
	supportsInterface	Public		-
STRYToken	Implementation	ERC20, ERC20Burnable, ERC20Permit		
		Public	✓	ERC20 ERC20Permit
DividendReceiver	Implementation			
	getTreasury	External		-
	getUserClaimableDividend	External		-
	getUserBalance	External		-
	claimDividend	External	✓	-
	dividendTotalSupply	Internal		
	dividendAccountBalance	Internal		
	addDividend	Internal	✓	
	dividendsOwing	Internal		
	updateAccountDividends	Internal	✓	
	_claimTreasury	Internal	✓	
	injectTreasury	External	✓	-
	_addTreasury	Internal	✓	
	_removeTreasury	Internal	✓	
	_addBalance	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
DividendReceiver
getTreasury
getUserClaimableDividend
dividendsOwing
getUserBalance
payable
claimDividend
updateAccountDividends
dividendAccountBalance
_claimTreasury
DividendReceiver__NoBalance
injectTreasury
DividendReceiver__TransferFailed
addDividend
IERC20
_addTreasury
dividendTotalSupply
_removeTreasury
_addBalance
STRYToken
<Constructor>
_mint
StoryChain
tokenIsAllowed
StoryChain__TokenNotAllowed
__AccessControl_init
__ERC721Enumerable_init
__ERC721_init
IERC20
initialize
_grantRole
claimVotingReward
__ReentrancyGuard_init
setStoryFromBnbRate
__UUPSUpgradeable_init
setTokenAllowed
createWhitelistedStory
updateMultiple
createStory
createCompetition
enterCompetition
setFee
continueStory
setNftShare
StoryChain__ProofFailed
setSystemCostFee
StoryChain__NoRewardToClaim
setConvertFeesToStry
StoryChain__AlreadyClaimed
setBaseURI
DividendReceiver__TransferFailed
skipNonce
payable
claimTreasury
_updateStory
getFee
_rejectPrompt
getCompetition
executeCompetition
getVersion
StoryChain__CantSkipToOldNonce
dividendTotalSupply
_claimTreasury
dividendAccountBalance
migrateNFT
_authorizeUpgrade
totalSupply
_increaseBalance
balanceOf
supportsInterface
_update
tokenURI
_requireOwned
_createStory
StoryChain__CantBeZeroPages
StoryChain__OpenEndingCantHaveMultiplePages
calculateFee
_newPromptEntered
StoryChain__LessThanFee
StoryChain__YouAreNotWhitelistedOnThisStory
StoryChain__StoryNotStartedYet
StoryChain__CompetitionTokenIsDifferent
StoryChain__StoryIsUpdatingCurrentPage
getStryFromBNB
_continueStory
StoryChain__CompetitionIsOver
_dealFees
StoryChain__WrongToken
StoryChain__StoryIsNotUpdating
StoryChain__StoryIsClosed
StoryChain__NoSuchPrompt
_addTreasury
StoryChain__PromptAlreadyProcessed
addDividend
_mint
_removeTreasury
_addBalance
StoryChain__CompetitionIsAlreadyComplete
StoryChain__NotACompetition
StoryChain__NoSuchNFT
StoryChain__PercentagesCantBeMoreThan100
StoryChain__ArrayLengthsMustMatch
StoryChain__CompetitionIsNotOver
StoryChain__AlreadyMigrated
_ownerOf
updateUserNFTRewards
updateAccountDividends
bytes
concat
MerkleProof
verify
ReentrancyGuardUpgradeable
_update
_increaseBalance
supportsInterface
IERC20
safeTransferFrom
safeTransfer
string
concat
tokenId
toString

Summary

The StoryChain contract is designed to support a blockchain-based storytelling platform that allows users to create, manage, and participate in story-driven competitions. It integrates features such as AI-driven content generation, user-specific whitelists, and financial operations like fee handling and reward distributions. This audit investigates security issues, business logic concerns and potential improvements.

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.

The Cyberscope team

https://www.cyberscope.io