SOLIDIFIED

Audit Report for The Sandbox - March 8, 2021

Summary

Audit Report prepared by Solidified covering the Sandbox Polygon staking, asset giveaway and land sale smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on September 8, 2021, and the results are presented here.

Audited Files

The following contracts were covered during the audit:

https://github.com/thesandboxgame/sandbox-smart-contracts/blob/master/src/solc_0.8/polygon/LiquidityMining/PolygonLandWeightedSANDRewardPool.sol

https://github.com/thesandboxgame/sandbox-smart-contracts/blob/master/src/solc_0.8/polygon/LiquidityMining/PolygonSANDRewardPool.sol

https://github.com/thesandboxgame/sandbox-smart-contracts/blob/master/src/solc_0.8/claims/AssetGiveaway/AssetGiveaway.sol

https://github.com/thesandboxgame/sandbox-smart-contracts/blob/master/src/solc_0.8/claims/MultiGiveaway/MultiGiveaway.sol

https://github.com/thesandboxgame/sandbox-smart-contracts/blob/master/src/solc_0.6/EstateSale/EstateSaleWithAuth.sol

Commit number: `306c134e7f94549f0b667566158f1d7a227e600fc`

Intended Behavior

The audited contracts implement the following functionalities:

- A liquidity mining solution for a staking token
- A giveaway contract that allows users to claim assets with a merkle proof
- A merkle proof-based land sale contract

Executive Summary

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	Medium	-
Test Coverage	High	-

Issues Found

Solidified found that the Sandbox contracts contain no critical issue, no major issues, 1 minor issue in addition to 2 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	PolygonLandWeightedSANDRewardPool.sol: A user could increase their staking contribution using loaned token	Minor	Pending
2	PolygonLandWeightedSANDRewardPool.sol: Confusing constant naming	Note	-
3	EstateSaleWithAuth.sol: Ambiguous signature verification in the function _checkAuthAndProofValidity()	Note	-

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. `PolygonLandWeightedSANDRewardPool.sol`: A user could increase their staking contribution using loaned NFTs

A user could temporarily loan `_multiplierNFToken` tokens before calling the `stake()` function to artificially increase his/her `contribution`.

Recommendation

The possible solutions to this issue could affect the use cases of the NFTs. If possible, consider restricting the transfer of the multiplier NFT by locking it in the ERC721 or transferring it to the staking contract. Another possible workaround is keeping track of the duration the NFT is held by the staker to calculate the reward.

Notes

2. `PolygonLandWeightedSANDRewardPool.sol`: Confusing constant naming

The contract declares a number of constants to be used throughout. Some of the naming conventions seem conflicting and make the contract hard to read. For example:

```
uint256 internal constant DECIMALS_9 = 1000000000;
```

suggest a fixed point multiplier of `10^9`, whereas

`uint256 internal constant NFT_FACTOR_6 = 10000`;

suggests a multiplier of `10^6`, but uses `10^4`.

Recommendation
Consider using more descriptive naming and declaring constants used in several contracts in one place.

3. `EstateSaleWithAuth.sol`: Ambiguous signature verification in the function `_checkAuthAndProofValidity()`

The function `_checkAuthAndProofValidity()` calculates the `hashedData` by doing a `keccak256` on `abi.encodePacked(..,assetIds,proof)` arrays. Doing the `abi.encodePacked()` on several arrays results in an ambiguous signature verification, since there is no way to tell which items belong to which array when verifying the signature.

Recommendation
Consider replacing `assetIds` with `keccak256(assetIds)` inside the call to `abi.encodePacked()`.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of TSB GAMING LTD or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.