SOLIDIFIED

Audit Report for The Sandbox Estate Sale and Fee Distributor on September 16, 2020

Summary

Audit Report prepared by Solidified covering The Sandbox Estate Sale and Fee Distributor smart contracts (and their associated components).

Process and Delivery

Two (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on September 5th, 2020. Fixes were supplied by the team on September 15th and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

- EstateSaleWithFee.sol
- LandToken.sol
- FeeDistributor.sol

Supplied in the repository: https://github.com/thesandboxgame/sandbox-private-contracts

Notes

The audit was based on commit `f5cd7e4a31e443800338e9ada4456d7a83845fa5`, Solidity compiler version `0.6.5`.

UPDATE: Fixes were supplied in commit number `86ef96ba563e8e56991fa4fe143fa46f7435fa71`.

Intended Behavior

The estate sale smart contract implements the sale of a token representing land, identified by coordinates and size. Land details and pricing are supplied by the buyer and verified through a Merkle proof.
The fee distributor contract acts as a vault for fees received and allows authorized parties to withdraw their share of the fees.

SOLIDIFIED

Audit Report for The Sandbox Estate Sale and Fee Distributor on September 16, 2020

Executive Summary

Solidified found that the Sandbox contracts contain 1 minor issue, in addition to 1 informational note.

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

Issues found:

Critical	Major	Minor	Notes
0	0	4	5

Issues Found

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. Lower precision for percentages in fee distributor that stated

In `FeeDistrbutor.sol`, percentages are supposed to have 4 decimals, according to documentation, but the calculations implicitly reduce precision to 2.

Example: 100% = 10**4 = 10 = 10,000 → However with 4 decimals 100 % should be represented as 1,000,000

Recommendation
Adjust decimal calculations to the desired precision.

Update
The issue has been dealt with by renaming and correcting the documentation. Precision itself is sufficient.

2. Malleable signatures

In `SigUtil.sol`, signatures are verified but not checked for malleability. The built-in `ecrecover` function still allows malleable signatures with `s` values in the higher ranges.

Recommendation
Check for malleable signatures. An example can be found at https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/cryptography/ECDSA.sol

Update
The team considers malleable signatures not to be an issue for this particular use case but will consider the recommendations for future versions.

3. Duplicates in recipient shares are not considered

`FeeDistributor.sol`: Duplicate addresses in `recipientShares` are not considered when calculating the total percentage. Duplicates values will be overwritten in mapping, but the array will include them and the constructor will execute successfully. This will affect the complete fee distribution

Recommendation
It is recommended to check for duplicates in the constructor or to perform an addition instead of an assignment to avoid duplication.

Update
Fixed.

4. Commission transfer can revert purchase

`ReferralValidator.sol`: When transferring the referral amount, there is a possibility of the referral address halting the flow by throwing. Whilst there is no incentive to do this, this could occur unintentionally, because of there not being enough gas forwarded by transfer (smart contract wallets that perform some additional logic). This may cause the whole transaction to revert, blocking the purchase.

Recommendation
It is recommended to either favor the pull mechanism for the referral amount rather than pushing them to the referrer or using a low-level call to transfer values without reverts.

Update
The contract will only be used with the SAND token, which is protected against this issue.

Notes

5. Code Layout and Styling Does not Comply with Solidity Style Guide

The contracts use an unusual coding convention and source files might be harder to navigate for those expecting a conventional layout.

Recommendation
We recommend applying the official Solidity style guide:
https://solidity.readthedocs.io/en/v0.7.1/style-guide.html

6. Code duplication

The functions handleReferralWithETH and handleReferralWithERC20 in `ReferalValidator.sol` share a lot of the same logic. The code could be reduced to avoid duplication. Similarly, `SigUtil.sol` includes duplicate code that can be further simplified for better readability and maintenance. Since the function uses assembly code, it is better to maintain this using one function than duplicating it.

Recommendation
Simplify codebase by removing duplicate code.

7. Consider using the same compiler for all files

Both `pragma solidity 0.6.6` and `pragma solidity ^0.6.0` are used within the same codebase.

Recommendation
For consistency, it's recommended to use a single fixed version throughout all the contracts.

Update
Fixed.

8. Inconsistent permission checks in multiple places

The Admin contract already includes a modifier that checks for permission. This can be used in other functions to avoid the duplicate manual check and error messages.

Recommendation
Make use of the modifier defined in the Admin contract.

Update
This is a continuous decision. The team prefers `require` statements over modifiers in some situations.

9. Address type casting

The contract uses old-style type casting for addresses.

Recommendation
Starting solidity v0.6.0 an address can be converted to address payable by calling `payable(referrer)` rather than using `address(uint160(referrer))`.

Update
Fixed.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the 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.