



Audit Report for The Sandbox Starter Pack Sale - October 02, 2020

Summary

Audit Report prepared by Solidified covering The Sandbox Starter Pack sale 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 29th, 2020 and the results are presented here.

Fixes were provided on October 1st, 2020, and have marked in this report

Audited Files

The following contracts were covered during the audit:

- src/StarterPack/PurchaseValidator.sol
- src/StarterPack/StarterPackV1.sol
- src/BaseWithStorage/ERC20Group.sol
- src//BaseWithStorage/ERC20SubToken.sol
- src/Catalyst/CatalystDataBase.sol
- src/Catalyst/CatalystToken.sol
- src/Catalyst/CatalystValue.sol
- src/Catalyst/ERC20GroupCatalyst.sol
- src/Catalyst/ERC20GroupGem.sol
- src/Catalyst/GemToken.sol

Supplied in the private repository:

<https://github.com/thesandboxgame/sandbox-private-contracts>

Notes

The audit was based on commit `fc5ea804fddf8e94683405f39630ad61a16f120`, Solidity compiler version `0.6.5`.

Updates: `056475df3bb961f478503e7e631644e82016e103`



Audit Report for The Sandbox Starter Pack Sale - October 02, 2020

Intended Behavior

The starter pack is used to sell starter packs, which are bundles of ERC20 tokens (organized in ERC20 group tokens) to users of the Sandbox game.

All sales are validated by signatures provided by an authorized address.

Executive Summary

Solidified found that the Sandbox contracts contain 2 major issues, 2 minor issues, in addition to 4 informational notes.

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	2	2	4

Issues Found

Critical Issues

No critical issues have been found.

Major Issues

1. Unsecure source of randomness

The computation for simulating pseudo-random numbers in `CatalystDataBase.sol`, depends purely on user-supplied values passed as arguments to `getValues()`. This means that the caller can pre-compute values that are advantageous.

Recommendation

Unpredictable values, such as the number of a future blockhash can be used to generate random numbers. In general, be aware that the security of the randomness has an upper bound on the value of the asset.

Update

Team explanation: “The function that uses a random parameter is only present in the contract to immortalize its rules. The parameter passed is actually a future blocknumber. When an Asset is minted, an event specifies the block number used for the calculation, that number is then passed to compute the value via that function.”

2. StarterPackV1.sol: Maker can update priceFeed

The StarterPackV1 uses Maker’s medianizer to read the ETH/USD value pair, but this address could be upgraded by Maker and the price stops updating, affecting all sale prices. This has happened in the past to Maker provided oracles.

Recommendation

A possible solution is for the admin to be able to change the medianizer address. Another one is to set a default price in case of problems with the feed.

Update

Team explanation: “While DAI was part of the initial scope, we are unlikely to enable DAI. Also if the medianizer were going to be shut down, we can redeploy a new contract with a different medianizer..”

Minor Issues**3. `SatrtterPackV1.sol`: Change can be sent to relayer**

When an exceeding amount is sent to the contract, the outstanding amount is returned to `msg.sender` which could be a meta transaction relayer. This can cause issues since `transfer()` only forwards a small amount of gas, which might not be enough to perform the necessary calculations to return to the original user.

Recommendation

It's recommended to use a low-level call to send funds, but it must be protected against reentrancy issues.

Update: Fixed

4. Potential Integer Overflow in `StartPackV1.sol`

`StarterPackV1.sol` - L286 & L290: Possible integer overflow if quantities and prices are large enough. The function `_calculateTotalPriceInSand()` does not include enough guards to check a possible overflow during total price calculation.

Recommendation

It is recommended to check for overflows after adding each item price to `totalPrice` or use `SafeMath` while accumulating prices.

Update: Fixed

Notes

5. StarterPackV1.sol: No input validation in price calculation

The function `_calculateTotalPriceInSand()` doesn't do any sanitization on the inputs it receives. A possible problem is for `catalystIds` and `catalystQuantities` to be of different lengths, which might cause unintended behaviours.

Update: Fixed

6. Potential gas saving on price calculation

Instead of every purchase calling the function to `_priceSelector()`, consider changing the dynamics of price changes by the admin calling the contract a second time to make the new price effective. This way every purchase will simply read the saved value, saving quite a lot of gas in the long term.

7. Get prices can return inconsistent switch time

`SarterPackV1.sol` L235 - L251: Switch time value can be inconsistent depending on the time the function is called. If the `getPrices` function is called after `_priceChangeDelay` but before someone purchases, `switchTime` is the time when the switch happened. But if someone has made a purchase, the `switchTime` shows 0. It is recommended to show consistent values without depending on external factors not directly related to this function.

Update

Team explanation: "The current behavior is what our backend/frontend expect and did not see a strong reason to change it."

8. Consider using alias or constants instead of magic numbers

In file `StarterPackV1.sol` in function `purchaseWithDAI()` a magic variable is used:

```
uint256 DAIRequired = amountInSand.mul(DAI_PRICE).div(1000000000000000000);
```



Audit Report for The Sandbox Starter Pack Sale - October 02, 2020

A better approach to use the alias `ether` to be sure of decimal places instead of counting zeros, which could be mistakenly changed during development, and is hard to notice. Another good practice is to export it to constant variables, so it can be properly named:

```
const DECIMAL_PLACES = 1 ether
```

And then

```
uint256 DAIRequired = amountInSand.mul(DAI_PRICE).div(DECIMAL_PLACES);
```

Update: Fixed



Audit Report for The Sandbox Starter Pack Sale - October 02, 2020

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.