SOLIDIFIED

Audit Report for Stakehouse - November 16, 2021

Summary

Audit Report prepared by Solidified covering the Stakehouse smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on November 16, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/bsn-eng/Stakehouse-Solidified

Commit number: `298b1e9fbed3e0ca8e891bff61c4c641a794c926`

UPDATE: Fixes received on November 18, 2021
Final commit number: `8a9c43659995f4132d751bdcbf1ebd39291b0035`

Intended Behavior

Stakehouse is a protocol that uses on-chain registries of ETH2 validators for trustless and permissionless liquidity abstraction of ETH2 staked assets.

Findings

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	High	-
Test Coverage	High	-

Issues Found

Solidified found that the Stakehouse contracts contain 1 critical issue, no major issues, 1 minor issue, and 2 informational notes.

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

Issue #	Description	Severity	Status
1	TransactionManager.sol: Function _addDepositToQueue() does not reset stakeHouseMemberQueue after depositing ETH to DepositContract	Critical	Resolved
2	CarefulMath.sol, Exponential.sol and ExponentialNoError.sol will not return arithmetic overflow/underflow errors as intended	Minor	Resolved
3	skLOOTFactory.sol: The function skLootItemClaim() contains pseudo random outcome generation	Note	-
4	Code Cleanup	Note	Resolved

Critical Issues

1. `TransactionManager.sol`: Function `_addDepositToQueue()` does not reset `stakeHouseMemberQueue` after depositing ETH to `DepositContract`

The function `_addDepositToQueue()` does not reset the value of `stakeHouseMemberQueue[_stakeHouse][_publicKey]` after it has deposited the `depositAmount` to `DepositContract`. This can allow an attacker to repetitively buy SLOTs with ETH that they did not deposit (after their initial 1 ETH deposit), potentially draining the entire contract balance.

Recommendation

Set `stakeHouseMemberQueue[_stakeHouse][_publicKey]` to zero after the corresponding `depositAmount` has been deposited to `DepositContract`.

Status

Resolved

Major Issues

No major issues have been found.

Minor Issues

2. `CarefulMath.sol`, `Exponential.sol` and `ExponentialNoError.sol` will not return arithmetic overflow/underflow errors as intended

Solidity compiler version `0.8.9` has built-in arithmetic overflow-underflow protection. Thus the behavior of `CarefulMath.sol`, `Exponential.sol` and `ExponentialNoError.sol` will be different from the original Compound implementation which used Solidity compiler version `^0.5.16`.

Recommendation

To preserve the same behaviour while using a new Solidity compiler, consider using the `unchecked` keyword for all actions which could result in arithmetic overflow or underflow.

Status

Resolved

Informational Notes

3. skLOOTFactory.sol: The function skLootItemClaim() contains pseudo random outcome generation

Function `skLootItemClaim()` contains pseudo random outcome generation. A smart contract could predict the outcome of `skLootItemClaim()` function and choose either to join StakeHouse and claim the item or wait so it could be done later in another block when the pseudo-random outcome is more favorable.

Recommendation

Consider using Chainlink VRF to obtain random numbers

4. Code Cleanup

- dETH.sol: Contract declares unused constant: `BATCH_MINT_AMOUNT`. Recommendation: Consider removing.

Status

Resolved

Disclaimer