DAC.digital

Gdańsk, 6.04.2023

VAIOT Limited
Mosta, MST 1180, Malta
Suite 1, Level 2,
Cornerstone Business Centre,
16th September Square

Smart contract review report

1. Review Scope

The scope review encompassed the evaluation of the "StakingRewards.sol" smart contract written in Solidity using version 0.8.18. The review was conducted on 04.03.2023. The objective of the audit was to assess the solidity code and identify any potential security and performance vulnerabilities.

2. Observations/Finding

The Solidity code appears to be robust, and no critical issues have been identified in terms of security or performance. The smart contract under review is designed to be used for staking a single asset. While deploying the contract onto the blockchain we can determine which ERC20 token will be used for staking and earning rewards purposes. To prevent reentrancy attacks, the contract uses ReentrancyGuard from OpenZepelin.

The staking logic is based on the popular and well-known Synthetix StakingReward contract with two major changes.

To begin with, the smart contract under review allows determining the duration of the next pool while setting reward for the next pool. If the previous reward pool has not yet finished, the new duration is set, and the old one is overridden, while also recalculating the reward rate based on the token amount transferred for the new reward pool.

Furthermore, the contract's logic has been improved with the introduction of a withdrawal grace period. This ensures that addresses staking their assets are unable to

withdraw their funds for a certain duration after initiating the withdrawal request. If the funds are withdrawn before the end of this period, a 10% fee is deducted from their funds and paid to a special fee collector address.
This portion of the code is well-written and does not introduce any risks in terms of performance and security. The rest of the logic is the same as the well-known and battle-tested StakingReward contract from Synthetix.

The smart contract consists of public and restricted methods that can only be called by the owner of the contract.

Restricted methods include "notifyRewardAmount," used for setting rewards and the duration of the next reward pool, "changeStakeLimit," used for setting the token limit available to stake for one address, "changePoolLimit," used for setting the pool limit indicating how many tokens can be transferred to the smart contract, "transferOwnership" used to change the owner of the smart contract, "changeGracePeriod" used for modifying the duration of time for which users are prohibited from withdrawing their assets after they have initialized the withdrawal request, and "recoverERC20," allowing for returning ERC20 tokens accidentally sent to the contract.

The remaining contract methods are publicly available, with noteworthy ones being "stake," used for passing a specific amount of tokens to the smart contract to earn rewards, "initializeWithdrawal" used for initiating a withdrawal request with a particular token amount if the user no longer wishes to earn rewards or would like to reduce their reward rate, "claimWithdrawal" which permits users to claim their tokens after the grace period has concluded, and "withdrawReward" used for withdrawing tokens earned by staking.

During the review, only one minor-to-medium issue was found, which also exists in the Synthetix smart contract code. The issue is not an exploitable bug and is also not a "must fix" issue. In certain instances, a fraction of the rewards may remain unutilized within the contract. Whenever a protocol initiates a reward cycle, it aims to distribute a predetermined amount of incentives during that period. If a certain amount remains unused, and a new reward cycle is not initiated, that amount remains dormant inside the contract. As there will likely be a delay between the first stake and the reward cycle initiation, it would be advisable to address this matter.

3. Recommendations

From a profitability perspective, it would be beneficial to address the issue of tokens that are dormant inside the contract. It is recommended to define "finishAt" in the first "stake()" that is done after "notifyRewardAmount()" when total deposits are zero. Furthermore, it will be necessary to invoke the logic from the "updateRewards" modifier once again after setting the value of "finishAt".

In terms of utility, the addition of a "pausable" feature to a smart contract provides increased flexibility for managing the contract's operations and mitigating the risk of irreversible errors or losses. This can enhance the contract's robustness and reliability, contributing to the trust and confidence of its users.

4. Reviewers

Patryk Wojciechowski - is an experienced software developer specializing in frontend technologies and Ethereum blockchain solutions. With expertise in Web3, he has the ability to build decentralized applications that interact with smart contracts. Additionally, he has a strong background in backend development, allowing for a well-rounded skill set and the ability to build end-to-end solutions.