Security assessment and code review

February 14 , 2022

Prepared for:
Equilibrium

Prepared by:
Karl Yu | HashCloak Inc

Table Of Contents

Executive Summary

From January 31 to February 14 2022, Equilibrium engaged HashCloak for an audit of their Vanilla v2 smart contracts. Vanilla v2 is an asset manager for DeFi tokens. It leverages Balancer Managed Pools in order to build a prediction market, Juicenet, for creating weighted portfolios of users' assets. The system is managed through a dual token system, in which the VNL token manages governance-related functionality and the JUICE token is used as a staking token for earning rewards.

The commit hash for the audited version is 3d80b38c05d225dbd8b4cd91d231d7ae3097addc. The scope for the audit was all files ending in **.sol** in the **contracts** folder.

Throughout the course of the audit, we familiarized ourselves with the key functionality of the Juicenet contracts . We also started a manual analysis of the contract and used automated, off-the-shelf tools as well.

We initially identified issues ranging from low severity to informational and provided recommendations to improve code quality and mitigations against several attacks. After discussing with the Equilibrium team , we updated our findings. We also provide some general recommendations for further improving the code base.

Severity	**Number of Findings**
Critical	0
High	0

Medium	0
Low	0
Informational	0

Overview

Vanilla v2 is a DeFi protocol that enables users to have their assets managed in a decentralized and permissionless way. Vanilla v2 provides both decentralized pools in which users can invest in and an active way for users to make decisions about the funds in these pools.

The Vanilla v2 system architecture is separated into 3 components:

- Vanilla DAO: The governance component for the protocol. It manages functionality related to fees, liquidity provision, and emergency handling.
- Juicenet: The component in charge of making decisions about what assets to invest in through a decentralized prediction market.
- Vanilla Pools: These are Balancer Managed Pools that use the parameters (i.e. portfolio weights) that are returned by the Juicenet for managing LP assets.

Scope

The scope for the audit was the following:

- All of the interfaces in contracts/interfaces used to provide interfaces that JuiceStaking.sol implements.
- JuiceStakerDelegateEIP712Util.sol in contracts for handling EIP712 related utilities. This contract is meant to be inherited by the JuiceStaking.sol contract.
- JuiceStaking.sol in contracts which contains the main functionality for the Vanilla v2 DeFi protocol

Additionally, we looked through the tests in contracts and the high-level test folder in order to better understand the codebase.

Any libraries such as the OpenZeppelin libraries used in the codebase were out of

scope and any third party DeFi protocols that Vanilla v2 relies on such as Balancer were also out of scope of the audit but were considered for failure cases that might affect the functioning of the Vanilla v2 contracts.

General Recommendations

Potential for Re-entrancy in deposit and withdrawal functions

Based on the fact that Juicenet's functionality is quite simple at the current stage , we suspect that when adding more functionality in the future , it will be quite vulnerable to the functions defined in `deposit()`, `doDeposit()`, `withdraw()`, `doWithdraw()` .

One of the attacks we've considered is reentrancy attack . Since these functions correctly apply the checks-effects-interactions pattern, if a re-entrancy were to occur, the attacker would not be able to either trick the deposit functions into depositing more assets or withdraw more assets than they own.

Impact: An malicious caller creates a contract and leverages an ERC20 implementation that allows for re-entrancy (the code snippet already assumes such a token exists) as follows to call these functions , which is possible to do re-entrancy.

```
Contract BadStaker {
var JuiceStaking = ...
function deposit(amount) {
    JuiceStaking.deposit(amount, address(this);
}
function withdraw(amount) {
    if JuiceStaking.unstakedBalance(address(this)) > 0 {
         JuiceStaking.withdraw(amount, address(this))
}}
```

Suggestion: Use function modifier to prevent re-entrancy .

Check return values of external calls

This codebase makes heavy use of external contracts and as such makes many external calls. We highly recommend that the return values of external calls are checked consistently throughout the codebase in order to fail gracefully and properly handle exceptions and reverts.

Check return values do not occur underflow/overflow

This codebase has quite a few arithmetic operations that potentially occur underflow/overflow when adding functionalities to the roles in Juicenet. We have checked all the involved arithmetic operations in the audited contract for correctness, and also recommend developers to be cautious about the results of such operations.