9LivesLabs Security Review

Reviewers
0xnirlin, Lead
cats, Lead

1 Executive Summary

Chad Finance reached out and engaged with 9LivesLabs to review ChadFinance.

Repository	Commit
Chad Finance	04cf6a77964fb30e1bfb5093f8afdde2417559f0

Summary

Type of Project	CDP Stablecoin
Engagement Date	March 24th, 2024
Methods	Manual Review
Available Documentation	Low

Total Issues

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	3
Gas Optimizations and Informational	3

Contents

2 9LivesLabs

9LivesLabs is a team of smart contract security researchers comprising of 0xnirlin & cats. Together, we help secure the Web3 ecosystem. We offer security reviews and related services to Web3 projects.

3 Introduction

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a two person team. Any modifications to the code will require a new security review.

4 Findings

4.1 Medium Risk

4.1.1 No minimum borrow amount and incentive to liquidate small positions can create bad debt

Severity: Medium

Context: L2Vault.sol#L84-L97

Description:

There is currently no incentive in the protocol to liquidate small borrows that don't ever get repaid. Furthermore, there is no minimum amount to borrow either, as the available borrow amount corresponds to the liquidity available in the UniswapV3 NFT used as collateral:

```
function borrow(uint256 pos, uint256 amount) external {

  _checkOwnerAndtransferNFT(pos);
  _accrueInterest(pos);
  _addDebt(pos, amount);

  conjurer.conjure(msg.sender, amount);

  if(!checkPositionHealth(pos)){
      revert Vault__positionUnderwater();
  }

  emit Borrow(msg.sender, pos, amount);
}
```

Gas on mainnet is becoming evermore expensive so this is a real problem and due to all of this, the protocol can accumulate bad debt from small positions over time which users don't want to liquidate.

Recommendation: Set a minimum borrow amount which would counteract this.

Chad Finance: L2Vault will consistently be deployed on Layer 2, where gas fees are more affordable. Should there be any instances of bad debt involving smaller amounts, the protocol team will ensure timely liquidation of such positions

9LivesLabs: Acknowledged

4.2 Low Risk

4.2.1 Liquidator's payout should round down instead of up

Severity: Low

Context: L2Vault.sol#L311-L319

Description:

The liquidator's payout is rounded up. Currently in the contract it does not pose a risk, but it is always recommended to round up only for protocol side, while rounding down for clients. In case of future changes or integrations, this type of rounding might create issues.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged

4.2.2 Oracle cannot be deployed with latest version of OpenZeppelin's Ownable

Severity: Low

Context: `Oracle.sol`

Description:

The oracle contract is missing a constructor which is necessary when using the latest version of OZ's Ownable contract. The issue is that an address needs to be specified in the constructor's `Ownable(address)`, but since the oracle contract is missing one, compilation will revert and contract cannot be deployed. Since foundry did not install the dependencies, we could not assess which version is used. In that case we installed the latest versions where it is an issue.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged

4.2.3 No deadline when borrowing can execute function at a later time when the collateral is de-valued

Severity: Low

Context: `L2Vault.sol#L84-L97`

Description:

When users borrow from the protocol, they collaterize their Uniswap V3 NFT, whose value determines the amount they can borrow. The `maxDebt()` function is the one which values the NFT and outputs the available borrow amount:

```
    function maxDebt(uint256 tokenId) public view returns (uint256){
        (
            address token0,
            address token1,
            uint256 amount0,
            uint256 amount1,
            TokenInfo memory token0Info,
            TokenInfo memory token1Info
        ) = _getTokenAndAmounts(tokenId);


        uint256 coll = Math.min(token0Info.collateralFactor,
↪   token1Info.collateralFactor);

        if(coll == 0){
            return 0;
        }

        return Math.mulDiv(
            coll,
            _calculatePrice(
                amount0,
                amount1,
                uint256(token0Info.decimals),
                uint256(token1Info.decimals),
                token0,
                token1
            ),
            BASIS_POINT);
    }
```

But when users are borrowing, since there is no deadline, their transaction can get executed at a later time when the value of the collateral is less. What this allows is the following scenario:

Let's say for simplicity's sake the user can borrow 50% of their collateral value, so 2:1 ratio.

1. Alice transfers her NFT and borrows for 15% of the collateral value
2. Alice later wants to borrow some more funds, another 15% of her collateral value which would put her total borrow at 30%
3. There is no deadline on the borrow and Alice's transaction is filled at a later time where the value of her collateral has dropped

4. Instead of her borrow now being 30% of the collateral as planned, she is now filled at for example 45-48% of the collateral value
5. The borrow will still succeed as she is not considered underwater, but her position is dangerously close to the liquidation threshold

Recommendation: Allow deadline to be passed for borrows since the collateral of the NFT can drop.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged

4.3 Info/Gas Optimizations

4.3.1 Event emission is wrong when repaying debt

Severity: Info

Description:

When repaying debt and `amount <= interest`, at the end of the if statement, amount is zero'd out. Since the variable is later used in an event emission, it will always emit 0 amount repaid in this case.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged

4.3.2 Hardcoded values in `conjurer.sol`

Severity: Info

Description:

Hardcoded values in conjurer are not recommended in case of deploying on multiple chains. It is recommended to add them to the constructor instead.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged

4.3.3 `_getTick` function is redundant

Severity: Info

Description:

The `_getTick` function is currently not used anywhere in the protocol and redundant.

Chad Finance: Acknowledged

9LivesLabs: Acknowledged