AffineDefi Restaking Security Review

Version 2.0

02.05.2024-07-05.2024

Conducted by:
MaslarovK, Independent Security Researcher

Table of Contents

1 About MaslarovK

MaslarovK is an independent security researcher from Bulgaria. He has secured various protocols through private audits and public contests - Secured ~$5M in TVL.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	AffineDeFi
Repository	https://github.com/AffineLabs/contracts/
Commit hash	4d28ca86adab6b9a9e342044516265e0504e0e05
Resolution	173b930a3eb93d2104172595a868c6b3b8c73247
Documentation	N/A
Methods	Manual review & testing

Scope

vaults/restaking/AffineDelegator.sol
vaults/restaking/staking/AffineRestaking.sol
vaults/restaking/staking/DelegatorBeacon.sol
vaults/restaking/staking/IDelegator.sol
vaults/restaking/staking/UltraLRT.sol
vaults/restaking/staking/UltraLRTStorage.sol
vaults/restaking/staking/WithdrawalEscrowV2.sol

Issues Found

Critical risk	0
High risk	3
Medium risk	0
Low risk	1
Informational	1

5 Findings

5.1 High risk

5.1.1 Wrong amount passed in calculation in the AffineDelegator::delegate

Severity: *High risk*

Context: AffineDelegator.sol#L676

Description: In the `AffineDelegator::delegate`

```
function delegate(uint256 amount) external onlyVault {
        // take stETH from vault
        stETH.transferFrom(address(vault), address(this), amount);

        // deposit into strategy
        strategyManager.depositIntoStrategy(address(stEthStrategy), address(stETH),
            stETH.balanceOf(address(this)));

        // delegate to operator if not already
        if (!isDelegated) {
            _delegateToOperator();
        }
    }
```

`stETH.balanceOf(address(this)` is used when depositing into strategy instead of `amount`. However this is wrong because if `stETH.balanceOf(address(this)` is different than `amount` which is highly possible - it will mess the accounting in the `UltraLRT::delegateToDelegator`

```
function delegateToDelegator(address _delegator, uint256 amount) external onlyRole(
    HARVESTER) {
        IDelegator delegator = IDelegator(_delegator);

        DelegatorInfo memory info = delegatorMap[_delegator];

        if (!info.isActive) revert ReStakingErrors.InactiveDelegator();
        if (vaultAssets() < amount) revert ReStakingErrors.InsufficientLiquidAssets
            ();

        // delegate
        ERC20(asset()).approve(_delegator, amount);
        delegator.delegate(amount);

        info.balance += uint248(amount);
        delegatorMap[_delegator] = info;
        delegatorAssets += amount;
    }
```

Recommendation: Implement the following changes, I have described them in the comments:

```
function delegate(uint256 amount) external onlyVault {
        // take stETH from vault
        stETH.transferFrom(address(vault), address(this), amount);

        // deposit into strategy
        strategyManager.depositIntoStrategy(address(stEthStrategy), address(stETH),
            amount);
        // delegate to operator if not already
        if (!isDelegated) {
            _delegateToOperator();
        }
    }
```

Resolution: Fixed

5.1.2 Burning wrong amount of shares in UltraLRT::_withdraw

Severity: *High risk*

Context: UltraLRT.sol#L249

Description: In the `AffineDelegator::delegate`

```
    function _withdraw(address caller, address receiver, address owner, uint256
        assets, uint256 shares)
        internal
        override
    {
        if (caller != owner) {
            _spendAllowance(owner, caller, shares);
        }

        // If _asset is ERC777, ‘transfer‘ can trigger a reentrancy AFTER the
           transfer happens through the
        // ‘tokensReceived‘ hook. On the other hand, the ‘tokensToSend‘ hook, that
           is triggered before the transfer,
        // calls the vault, which is assumed not malicious.
        //
        // Conclusion: we need to do the transfer after the burn so that any
           reentrancy would happen after the
        // shares are burned and after the assets are transfered, which is a valid
           state.

        // TODO: calculate fees

        if (!canWithdraw(assets)) {
            // do withdrawal request
            _transfer(_msgSender(), address(escrow), shares);
            escrow.registerWithdrawalRequest(receiver, shares);
            // do immediate withdrawal request for user
            _liquidationRequest(assets);
            return;
        }
        _burn(owner, shares);

        uint256 assetsToReceive = Math.min(vaultAssets(), assets);

        if (assetsToReceive + ST_ETH_TRANSFER_BUFFER < assets) revert
           ReStakingErrors.InsufficientLiquidAssets();

        ERC20(asset()).safeTransfer(receiver, assetsToReceive);
        emit Withdraw(caller, receiver, owner, assetsToReceive, shares);
    }
```

When burning the shares, you are burning the amount corresponding to the assets, but after that if the `vaultAssets()< assets`, the amount of assets to transfer will be less than the one needed for the shares burned.

Recommendation: Calculate the shares when you know what are the `assetsToReceive`.

Resolution: Aknowledged

5.1.3 Wrong parameter passed to UltraLRT::_delegatorWithdrawRequest in UltraLRT::_liquidationRequest

Severity: *High risk*

Context: UltraLRT.sol#L305

Description: In the `UltraLRT::_liquidationRequest`

```
     function _liquidationRequest(uint256 assets) internal {
        for (uint256 i = 0; i < delegatorCount; i++) {
            IDelegator delegator = delegatorQueue[i];
            uint256 assetsToRequest = Math.min(delegator.withdrawableAssets(),
                assets);
            _delegatorWithdrawRequest(delegator, assetsToRequest);
            if (assetsToRequest == assets) {
                break;
            }
            assets -= assetsToRequest;
        }
    }
```

When calculating the `assetsToRequest`, if `assets < delegator.withdrawableAssets()` then `assetsToRequest = assets` and if `delegator.withdrawableAssets()< assets` then `assetsToRequest = delegator.withdrawableAssets()`. So practically, there is no scenario where `assets > delegator.withdrawableAssets()`, which will make the if check in `UltraLRT::_delegatorWithdrawRequest` impossible to happen

```
function _delegatorWithdrawRequest(IDelegator delegator, uint256 assets) internal {
        if (assets > delegator.withdrawableAssets()) revert ReStakingErrors.
            ExceedsDelegatorWithdrawableAssets();
        delegator.requestWithdrawal(assets);
    }
```

Recommendation: Change the function as follows, passing the right parameter:

```
     function _liquidationRequest(uint256 assets) internal {
        for (uint256 i = 0; i < delegatorCount; i++) {
            IDelegator delegator = delegatorQueue[i];
            uint256 assetsToRequest = Math.min(delegator.withdrawableAssets(),
                assets);
            _delegatorWithdrawRequest(delegator, assets);
            if (assetsToRequest == assets) {
                break;
            }
            assets -= assetsToRequest;
        }
    }
```

Resolution: Fixed

5.2 Low risk

5.2.1 Consider decreasing the maxDeposit for a user on every deposit

Severity: *High risk*

Context: TrotelCoinStakingV2.sol#L136-L140

Description: In the `UltraLRT::maxDeposit`, the amount is set to `type(uint256).max`, which is fine, but given the fact that the function can be overriden and different value may be set - would suggest decreasing it on every deposit with the amoun deposited.

```
function maxDeposit(address) public view virtual override returns (uint256) {
        return type(uint256).max;
    }
```

Resolution: Aknowledged

5.3 Informational

5.3.1 Consider refactoring the immutable veriables to constant as they are initialized upon declaration.

Severity: *High risk*

Context: TrotelCoinStakingV2.sol#L136-L140

Description:

```
IStrategyManager public immutable strategyManager = IStrategyManager(0
    x858646372CC42E1A627fcE94aa7A7033e7CF075A); // StrategyManager for Eigenlayer
    IDelegationManager public immutable delegationManager =
        IDelegationManager(0x39053D51B77DC0d36036Fc1fCc8Cb819df8Ef37A); //
            DelegationManager for Eigenlayer
    IStrategy public immutable stEthStrategy = IStrategy(0
        x93c4b944D05dfe6df7645A86cd2206016c51564D); // stETH strategy on Eigenlayer
```

Resolution: Fixed