AstraSec

UwU ICV3 & MFDV3

Security Audit Report

July 11, 2024

Contents

1 | Introduction

1.1 About UwU Lend

`UwU Lend` is a decentralized non-custodial liquidity market protocol where users can participate as depositors, borrowers or LP stakers. The new implementation of the `IncentivesControllerV3 (ICV3)` and `MultiFeeDistributionV3 (MFDV3)` contracts includes several key changes. These updates introduce features such as adding a blacklist for calling `claimReceiver()`, minting only UwU tokens as rewards for `UwU Lend` users, and imposing a 50% penalty for early exits.

1.2 Audit Scope

The following source code was reviewed in the audit:

- https://github.com/Test-Land/uwu-contracts/pull/22
- commit: 17b8afd
- files: `MultiFeeDistributionV3.sol`, `IncentivesControllerV3.sol` and the test scripts in the `tasks/operations/impls-v3` directory

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/Test-Land/uwu-contracts/pull/22
- commit: 33b939d

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `UwU ICV3 & MFDV3`. Throughout this audit, we identified a total of 4 issues spanning various severity levels. All the issues have been properly fixed or acknowleged by the team. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	1	-	-	1
Low	3	3	-	-
Informational	-	-	-	-
Total	4	3	-	1

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ Timely _massUpdatePools() in addPool()

L-1 Potential Delay of New Emission

L-2 Revisited Setup of IncentivesControllerV3

L-3 Potential Risks Associated with Centralization

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[M-1] Timely _massUpdatePools() in addPool()

Target	Category	IMPACT	LIKELIHOOD	STATUS
IncentivesControllerV3.sol	Business Logic	Medium	Medium	Addressed

In the `IncentivesControllerV3 (ICV3)` contract, the `addPool()` function is utilized by the pool configurator to add a new pool into `ICV3`. The addition of a new pool with the specified `_allocPoint` parameter affects the reward distribution among all existing pools. Therefore, it is essential to invoke the `_massUpdatePools()` function to settle all distributed rewards for all pools before the new pool begins to share in the rewards. However, upon reviewing the implementation of the `addPool()` function, we notice that it does not consistently invoke the `_massUpdatePools()` function.

To elaborate, we show below the code snippet of the `addPool()` function. Specifically, it calls the `_updateEmissions()` function (line 120) to check and update the emission. In the `_updateEmissions()` function, it checks if a new emission can be started and calls the `_massUpdatePools()` function (line 376) only when a new emission can be started. This means that if the new emission cannot be started, the `_massUpdatePools()` function will not be invoked.

As a result, if the `_massUpdatePools()` function is not invoked within the `addPool()` function, the new pool will increase the `totalAllocPoint`, thereby reducing the amount of rewards distributed to the existing pools.

IncentivesControllerV3::addPool()

```
116 function addPool(address _token, uint _allocPoint) external {
117   require(_token != address(0), 'token cannot be zero address');
118   require(isPoolConfigurator[msg.sender], 'only pool configurator can add pools'
        );
119   require(poolInfo[_token].lastRewardTime == 0, 'pool already registered');
120   _updateEmissions();
121   totalAllocPoint = totalAllocPoint.add(_allocPoint);
122   registeredTokens.push(_token);
123   poolInfo[_token] = PoolInfo(...);
124   emit PoolAdded(_token, _allocPoint);
125 }
```

IncentivesControllerV3::_updateEmissions()

```
371 function _updateEmissions() internal {
372   uint length = emissionSchedule.length;
373   if (startTime != 0 && length != 0) {
374     EmissionPoint memory e = emissionSchedule[length - 1];
```

```
    if (block.timestamp.sub(startTime) > e.startTimeOffset) {
      _massUpdatePools();
      rewardsPerSecond = uint(e.rewardsPerSecond);
      emissionSchedule.pop();
    }
  }
}
```

Remediation Revisit the implementation of the `addPool()` function to ensure that the `_massUpdatePools()` function can be consistently invoked.

Response By Team This recommendation has been accepted by the team.

[L-1] Potential Delay of New Emission

Target	Category	IMPACT	LIKELIHOOD	STATUS
IncentivesControllerV3.sol	Business Logic	Low	Low	Acknowledged

In the `IncentivesControllerV3` contract, new emissions are initiated by the first call to the `_updateEmissions()` function (line 121) after the new emission's start time has expired. Our analysis shows that the `_updateEmissions()` function can only be triggered by certain operations, such as `addPool()`, `claim()`, or `handleAction()`. This dependency on user operations may delay the timely initiation of new emissions if no such operations occur after the new emission's start time.

If the new emission cannot be started in a timely manner, the contract will continue using the legacy reward rate (`rewardsPerSecond`) to distribute rewards, which may not be intended.

IncentivesControllerV3::_updateEmissions()

```
116 function _updateEmissions() internal {
117   uint length = emissionSchedule.length;
118   if (startTime != 0 && length != 0) {
119     EmissionPoint memory e = emissionSchedule[length - 1];
120     if (block.timestamp.sub(startTime) > e.startTimeOffset) {
121       _massUpdatePools();
122       rewardsPerSecond = uint(e.rewardsPerSecond);
123       emissionSchedule.pop();
124     }
125   }
126 }
```

Remediation Check and start the new emission promptly once its start time arrives.

Response By Team This is not applicable to us. For UwU Lend, this is not an issue as we're updating the emissions schedule twice for the remaining years, both times for the same schedule. So, the old rewardsPerSecond is the same as the new rewardsPerSecond. We won't have emissions miscalculation caused by late triggered _updateEmissions().

[L-2] Revisited Setup of IncentivesControllerV3

Target	Category	IMPACT	LIKELIHOOD	STATUS
update-v3.ts	Business Logic	Low	Low	Acknowledged

As the last step to complete the update of the new IncentivesControllerV3, the setup() function is called. This function setups the contract with the existing pools and emissions schedule from the previous IncentivesController (ICV2) contract, as well as parameters such as startTime, rewardsPerSecond, and mintedTokens.

However, our analysis indicates that there is also a need to migrate the claimReceiver from ICV2. Without this migration, users would need to set claimReceiver again in ICV3. Our study suggests that claimReceiver can be migrated from ICV2 when either _initiateUserInfo() or _initiateUserBaseClaimable() is invoked by users in ICV3.

Moreover, it's suggested to set the blacklist for all the attacker's addresses. This would prevent the attacker from setting the claimReceiver in ICV3 and claiming the accumulated rewards associated with his addresses.

IncentivesControllerV3::setup()

```
function setup() external onlyOwner {
  require(!setuped, 'already setuped');
  uint length = incentivesController.poolLength();
  for (uint i = 0; i < length; i++) {
    address token = incentivesController.registeredTokens(i);
    IChefIncentivesController.PoolInfo memory oldInfo = incentivesController.
        poolInfo(token);
    poolInfo[token] = PoolInfo(...);
    registeredTokens.push(token);
    totalAllocPoint = totalAllocPoint.add(poolInfo[token].allocPoint);
  }
  _copyEmissionSchedule();
  startTime = incentivesController.startTime();
  rewardsPerSecond = incentivesController.rewardsPerSecond();
  mintedTokens = incentivesController.mintedTokens();
  maxMintableTokens = incentivesController.maxMintableTokens();
  setuped = true;
}
```

Remediation Properly migrate the `claimReceiver` from ICV2 and set the blacklist for all attacker's addresses.

Response By Team We have no users who set different addresses as claim receivers, so we don't need to take any extra steps. `setBlacklist()` in `setup()` for the attacker is also not an issue for us, as we have no rewards for any of the pools the attacker is currently in, so there are no rewards to claim. We can safely deploy, blacklist the attacker first, and then, in the future, give emissions to the variableDebt tokens the attacker holds safely.

[L-3] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

In the `IncentivesControllerV3` and `MultiFeeDistributionV3` contracts, the presence of the privileged owner account introduces risks of centralization, as they hold significant control and authority over critical operations governing the protocol. Our analysis shows that the owner is pointing to the address 0xb8416EaC2155E9636b5f728dd29810bf7e3bC20d, which is a Genosis multisig (2/4). The multisig owner account greatly mitigates the risk, but we still need to highlight the representative functions potentially affected by the privileges of the owner account.

Examples of Privileged Operations

```
function setMinters(address[] calldata _minters) external onlyOwner {
  delete minters;
  for (uint i = 0; i < _minters.length; i++) {
    minters.add(_minters[i]);
  }
}

function setPoolConfigurator(
  address _poolConfigurator,
  bool _isPoolConfigurator
) external onlyOwner {
  _setPoolConfigurator(_poolConfigurator, _isPoolConfigurator);
}

function setBlacklist(address _user, bool _isBlacklisted) external onlyOwner {
  blacklisted[_user] = _isBlacklisted;
  emit Blacklisted(_user, _isBlacklisted);
}

function setRewardMinter(IMultiFeeDistribution _miner) external onlyOwner {
  rewardMinter = _miner;
}
```

Remediation Properly manage the owner with the multisig account, and it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been acknowleged by the team.

4 | Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI