AstraSec

UwU Recovery

Security Audit Report

June 29, 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. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an over-collateralized (perpetually) fashion. LP stakers provide liquidity and receive revenue share when staking their LP tokens.

1.2 About The Exploit

On June 10, 2024, UwU Lend was exploited across three different transactions on the Ethereum Mainnet due to the flawed price oracle of sUSDe reserve, which resulted in a loss of over 5272 ETH, totaling approximately $23 million.

UTC Time	Exploit Transaction
Jun-10-2024 12:05:59 PM	0x242a0f...96408b
Jun-10-2024 12:06:35 PM	0xb3f067...017376
Jun-10-2024 12:12:35 PM	0xca1bbf...0d6ac3

After the exploit, the UwU Lend team promptly paused the protocol to conduct an investigation. Once the root cause was identified and the vulnerability mitigated, they unpaused the protocol to repay part of the bad debt left by the exploit. However, since the attacker still held a large amount of uSUSDE assets, he had the opportunity to borrow the repaid reserves in transaction 0x9235e0...216110. Ultimately, the UwU Lend team took emergency actions to freeze the protocol through the following operations:

- Reset the interest rate strategy: 0x457a8c...e5bc88.
- Pause the protocol: 0x0cef76...84d35d.
- Freeze all the reserves: 0xca85c9...af3b46.

- Disable the emissions for all reserves: 0x454a2a...a2d4b3.

Afterward, the UwU Lend team developed a process with the target to safely recover the entire protocol and involved AstraSec to audit the recovery process.

1.3 Audit Scope

The following source code was reviewed in the audit:

- https://github.com/Test-Land/uwu-contracts/pull/21
- commit: e6b7abd
- audited files: all the modified solidity smart contract and the test scripts in the tasks/operations/recovery directory

Please note that we assume the price oracle is robust and reliable, and that asset prices are provided in a timely manner. The implementation of the price oracle itself is not included in the scope of this audit.

Additionally, the following recovery process was reviewed in the audit:

- Step 1: Simulate all changes using Tenderly.
- Step 2: Add a dummy reserve to fix totalAllocPoint.
- Step 3: Upgrade uSUSDE (AToken) and variableDebtSUSDE (variable debt token) contracts to an intermediate version, clearing all the attacker's ATokens, along with the variableDebt token loans to himself.
- Step 4: Revert AToken and variableDebt token contracts of sUSDe reserve to their previous versions.
- Step 5: Unpause the protocol but keep each reserve individually frozen.
- Step 6: Update the affected reserves' Interest Rate Strategies.
- Step 7: Restart emissions to the tokens, as they should be.
- Step 8: Gradually repay the attacker's debt via the repay() function.
- Step 9: Revert to the original Interest Rate Strategies for recovered reserves.

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `UwU Recovery` project. By employing auxiliary tool techniques to supplement our thorough manual code review, we found the audited source files to be well structured and engineered, and there's no critical issue detected at the code level. However we have discovered the following recommendations for the recovery process.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	1	-	-	1
Low	3	2	-	1
Informational	-	-	-	-
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

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

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] Step-2: Rescue of Attacker's Rewards after totalAllocPoint is fixed

Target	Category	IMPACT	LIKELIHOOD	STATUS
IncentivesControllerV2.sol	Business Logic	Medium	Medium	Addressed

After the exploit, `UwU Lend` team disabled the emissions for all reserves in the `IncentivesControllerV2` contract. Since all pools emissions were set to 0, the `totalAllocPoint` was also set to 0, creating the division-by-0 error. This error will block any user operations on the protocol, thereby also block the recovery process needed to burn the attacker's uSUSDE and variableDebtSUSDE. To address this error, the recovery process plans to add a DUMMY reserve with positive allocation points to the protocol.

However, once the `totalAllocPoint` is fixed, users, including the attacker, can claim their accumulated rewards in the `IncentivesControllerV2` contract. Our analysis shows that the privileged owner can invoke the `setClaimReceiver()` function to set a new receiver for the attacker and then invoke the `claim()` function to direct all the attacker's rewards to the new receiver. By doing so, the attacker's accumulated rewards can be rescued.

IncentivesControllerV2::setClaimReceiver()

```
  function setClaimReceiver(address _user, address _receiver) external {
    require(msg.sender == _user  msg.sender == owner());
    claimReceiver[_user] = _receiver;
  }
```

IncentivesControllerV2::claim()

```
function claim(address _user, address[] calldata _tokens) external {
  for (uint i = 0; i < _tokens.length; i++) {
    initiateUserInfo(_user, _tokens[i]);
  }
  initiateUserBaseClaimable(_user);
  _updateEmissions();
  uint256 pending = userBaseClaimable[_user];
  userBaseClaimable[_user] = 0;
  uint256 _totalAllocPoint = totalAllocPoint;
  for (uint i = 0; i < _tokens.length; i++) {
    PoolInfo storage pool = poolInfo[_tokens[i]];
    require(pool.lastRewardTime > 0);
    _updatePool(pool, _totalAllocPoint);
    UserInfo storage user = userInfo[_tokens[i]][_user];
    uint256 rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12);
```

```
        pending = pending.add(rewardDebt.sub(user.rewardDebt));
        user.rewardDebt = rewardDebt;
      }
      _mint(_user, pending);
    }
```

Remediation Properly rescue the accumulated rewards of the attacker in the `IncentivesControllerV2` contract.

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

[L-1] Step-7: New Rewards Accumulated for Attacker after Emission Restored

Target	Category	IMPACT	LIKELIHOOD	STATUS
IncentivesControllerV2.sol	Business Logic	Low	Low	Acknowledged

In the Step-7 of the recovery process, emissions to the reserves will be restarted as they should be. Users will be able to earn new rewards for their LPs, represented by uToken or variableDebtToken. However, it is important to note that the attacker can also earn these rewards until the bad debt is fully repaid.

Our analysis shows that we can move all the debt tokens of the attacker to the team address, which will repay the debt for the attacker. This approach effectively prevents the attacker from accumulating rewards during the recovery process.

IncentivesControllerV2::claimableReward()

```
function claimableReward(
  address _user,
  address[] calldata _tokens
) external view returns (uint[] memory) {
  uint256[] memory claimable = new uint256[](_tokens.length);
  for (uint256 i = 0; i < _tokens.length; i++) {
    address token = _tokens[i];
    PoolInfo memory pool = poolInfo[token];
    UserInfo memory user;
    if (userInfoInitiated[token][_user]) {
      user = userInfo[token][_user];
    } else {...}
    uint256 accRewardPerShare = pool.accRewardPerShare;
    uint256 lpSupply = pool.totalSupply;
    if (block.timestamp > pool.lastRewardTime && lpSupply != 0) {
      uint256 duration = block.timestamp.sub(pool.lastRewardTime);
      uint256 reward = duration.mul(rewardsPerSecond).mul(pool.allocPoint).div(
          totalAllocPoint);
      accRewardPerShare = accRewardPerShare.add(reward.mul(1e12).div(lpSupply));
    }
    claimable[i] = user.amount.mul(accRewardPerShare).div(1e12).sub(user.
        rewardDebt);
  }
  return claimable;
}
```

Remediation Move all the debt tokens of the attacker to the team address to prevent the attacker from obtaining accumulated rewards, and then repay the debt gradually.

Response By Team We came up with an agreed solution which works, which is to zero out borrow emissions until the bad debt is fully repaid. We prefer the previously decided upon method to avoid risks associated with upgrading and reverting a half dozen contracts so that he can earn same 0 rewards we plan to give borrowers.

[L-2] Step-5: Proper Handling of Users Under Liquidation

Target	Category	IMPACT	LIKELIHOOD	STATUS
N/A	Business Logic	Low	Low	Acknowledged

Due to the exploit, the utilization rate of the impacted reserves increased to approximately 100%, resulting in higher borrow interest rates for these reserves. The higher borrow interest rates may push borrowers to be liquidatable. Fortunately, the `UwU Lend` team promptly reset the interest strategy to curb the worsening situation.

However, after the protocol was paused, borrowers were unable to repay their debt, which may also push them underwater due to price violations. Based on this, it is suggested to closely monitor the users' health factors and plan necessary solutions to protect these users from being liquidated or to develop a reimbursement plan for these users after they are liquidated.

Remediation Closely monitor the users who become at risk of liquidation before the protocol is unpaused and take appropriate actions to protect these users from being liquidated or provide reimbursement after they are liquidated.

Response By Team We have been monitoring the situation closely, and no accounts are currently under-collateralized or close to becoming a problem. Our solution is to offer to reimburse those who were profitably liquidated by us shortly after relaunch, offer to otc back to them on chain when they open a support ticket.

[L-3] Revisited Liquidation Threshold and Bonus Configuration in UwU

Target	Category	IMPACT	LIKELIHOOD	STATUS
LendingPool.sol	Configuration	Low	Low	Addressed

A robust protocol requires not only high-quality contract implementation but also proper configuration to ensure its smooth operation. The UwU protocol is forked from AAVE which has been fully audited by top auditors and has been running smoothly for years. However, the configuration of some key parameters for the WETH reserve makes the protocol more prone to incurring bad debts.

Specifically, when a borrower is liquidated, the profit of the liquidator is paid in the collateral of the borrower. To prevent introducing bad debt to the protocol from liquidating the debts of a borrower, we need to ensure that: `C*LT*(1+LB)<= C` (where `C` is the collateral value of the borrower, `LT` is the liquidation threshold of the collateral reserve, and `LB` is the liquidation bonus of the collateral reserve). Based on this, we can derive the formula: `R = 1-LT*(1+LB)`, where `R` represents the safe ratio that protects the protocol from incurring bad debts due to liquidation. In theory, the larger the `R`, the less likely the protocol is to incur bad debts.

In the following, we show configuration of the liquidation threshold parameter and the liquidation bonus parameter for the WETH reserve in UwU and AAVE.

	Liquidation Threshold	**Liquidation Bonus**	**Anti-BadDebt Buffer**
UwU	90%	10%	1%
AAVE	83%	5%	13%

Based on this configuration and the above formula, we can get `R` values of 1% and 13% respectively, indicating that UwU is more likely to incur bad debts than AAVE.

Remediation Revisit the configuration of the liquidation threshold and the liquidation bonus for all reserves in UwU.

Response By Team This recommendation has been accepted 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

4.3 Contact

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