



Beradrome Security Audit Report

July 18, 2025



Contents

1 Introduction

[1.1 About Beradrome](#)

[1.2 Source Code](#)

[1.3 Revision History](#)

2 Overall Assessment

3 Vulnerability Summary

[3.1 Overview](#)

[3.2 Security Level Reference](#)

[3.3 Vulnerability Details](#)

4 Appendix

[4.1 About AstraSec](#)

[4.2 Disclaimer](#)

[4.3 Contact](#)

1 Introduction

1.1 About Beradrome

Beradrome is Berachain's native restaking & liquidity marketplace, enabling ecosystem projects to build deeper liquidity for less, liquidation free HONEY loans and a flywheel boosted by its Berachain validator set.

The protocol introduces an innovative token structure, encompassing BERO, hiBERO, and oBERO tokens, each presenting users with diverse advantages and motivations. The supply of BERO tokens is algorithmically controlled via a bonding curve mechanism, ensuring a stable foundation for BERO tokens while providing liquidity at market-driven rates.



1.2 Source Code

The following source code was reviewed during the audit:

- ▶ <https://github.com/Heesho/beradrome>
- ▶ Commit id: fac7b90b547af9e2ae2cb664ff4a4857833de93c

And this is the final version representing all fixes implemented for the issues identified in the audit:

- ▶ <https://github.com/Heesho/beradrome>
- ▶ Commit id: 274a6681533fc25873b3d79ab69d0f79ccf7f627

1.3 Revision History

Version	Date	Description	Final commit
v1.0	January 20, 2025	Initial Audit	c1a8fcd
v1.1	July 18, 2025	Fund	274a668

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Beradrome protocol. Throughout this audit, we identified a total of **9** issues spanning various severity levels. 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	2	1	—	1
Medium	4	1	—	3
Low	3	—	—	3
Informational	—	—	—	—
Total	9	2	—	7

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to its detailed page, or scroll down to view all details in sequence.

~~H-1~~

[Possible Bypass of onlyNewEpoch Check](#)

~~H-2~~

[Inflated VToken Amount By Leveraged Borrow in TOKEN](#)

~~M-1~~

[Possible Discarded Reward in HiveDistro::distributeRewards\(\)](#)

~~M-2~~

[Revisited Logic of Voter::reviveGauge\(\)](#)

~~M-3~~

[Potential Denial-of-Service in KodiakPlugin::withdrawTo\(\)](#)

M-4

[Potential Risks Associated with Centralization](#)

~~L-1~~

[Improved Rounding Operation in TOKEN](#)

~~L-2~~

[Revisited Validation of amountToken in sell\(\)](#)

~~L-3~~

[Incompatibility with Non-ERC20 Tokens](#)

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	Acknowledged
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

3.3.1 [H-1] Possible Bypass of onlyNewEpoch Check

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Voter.sol	Business Logic	High	Medium	Addressed

The [Voter](#) contract allows VToken holders to participate in voting. However, a restriction is enforced to ensure that a user can only vote once per epoch, which is implemented through the [onlyNewEpoch\(\)](#) modifier. Upon reviewing the implementation of [onlyNewEpoch\(\)](#), we identified a potential bypass that could allow a user to vote multiple times within the same epoch.

Specifically, after each vote, the contract records the timestamp of the user's last vote. When the same user attempts to vote again, the [onlyNewEpoch\(\)](#) modifier checks whether the user has already voted in the current epoch. However, under certain conditions, if a user votes within the first block of the current epoch, they can repeatedly vote within that same block or in any subsequent block within the same epoch.

This vulnerability arises due to the comparison logic in the [onlyNewEpoch\(\)](#) modifier. The current implementation uses a < operator to compare the current epoch's start timestamp with the user's last voting timestamp. As a result, if the user votes exactly at the start of the epoch (i.e., within the first block), the comparison fails to prevent multiple votes within the same epoch.

beradrome - Voter.sol

```
89  modifier onlyNewEpoch(address account) {
90      if ((block.timestamp / DURATION) * DURATION < lastVoted[account]) revert Voter__AlreadyVotedThisEpoch();
91      _;
92  }
```

Remediation To address this issue, we recommend changing the comparison operator in the [onlyNewEpoch\(\)](#) modifier from < to <= to ensure that users are strictly limited to one vote per epoch, including within the first block of the epoch.

3.3.2 [H-2] Inflated VToken Amount By Leveraged Borrow in TOKEN

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
TOKEN.sol	Business Logic	High	Medium	Acknowledged

The TOKEN contract enables users to purchase TOKEN using BASE and also allows users to borrow BASE against VTOKEN collateral. Furthermore, the VTOKEN contract allows users to deposit TOKEN to mint VTOKEN. Based on this, users can exploit the protocol to repeatedly perform the following operations to accumulate a disproportionate amount of VTOKEN:

1. Buy TOKEN using BASE from the TOKEN contract.
2. Deposit TOKEN in the VTOKEN contract to mint VTOKEN.
3. Borrow BASE from the TOKEN contract using the newly minted VTOKEN as collateral.

This process enables users to leverage a limited amount of BASE to significantly inflate their VTOKEN holdings, granting them a larger share of TOKEN fee rewards and disproportionate voting power in governance decisions.

```
beradrome - TOKEN.sol

302 function borrow(uint256 amountBase)
303 {
304     address account = msg.sender;
305     uint256 credit = getAccountCredit(account);
306     if (credit < amountBase) revert TOKEN__ExceedsBorrowCreditLimit();
307     debts[account] += amountBase;
308     debtTotal += amountBase;
309     uint256 feeBASE = amountBase * BORROW_FEE / DIVISOR;
310     emit TOKEN__Borrow(account, amountBase);
311     BASE.safeTransfer(FEES, feeBASE);
312     BASE.safeTransfer(account, amountBase - feeBASE);
313     return true;
314 }
```

Remediation We recommend conducting a comprehensive review of the TOKEN economic model to address the risks associated with leveraged borrowing.

Response By Team When the user buys TOKEN to leverage up, it pushes the price up, so the LTV becomes lower for each subsequent buy and borrow.

3.3.3 [M-1] Possible Discarded Reward in HiveDistro::distributeRewards()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
HiveDistroFactory.sol	Coding Practices	Medium	Medium	Addressed

The `HiveDistro::distributeRewards()` function is responsible for transferring various reward tokens from the contract to the `HiveRewarder` contract for distribution to users. To ensure that the rewards produce a meaningful output, the function includes a preliminary check to verify that the current reward token balance in the contract exceeds a predefined threshold (`DURATION`). This check, i.e., `if (balance > DURATION)`, aims to prevent the distribution of insignificant reward amounts.

However, upon reviewing the implementation, we identified a potential issue. After verifying that the balance exceeds `DURATION`, the function deducts a fee from the reward token balance. If the deducted fee is sufficiently large, it could reduce the remaining balance to a value below `DURATION`. Consequently, when the balance is sent to the `HiveRewarder` contract via the `notifyRewardAmount()` function, it may fail to generate an effective reward distribution due to the insufficient balance.

```
beradrome - HiveDistroFactory.sol

51 function distributeRewards(address[] calldata rewardTokens) external nonReentrant {
52     for (uint256 i = 0; i < rewardTokens.length; i++) {
53         uint256 balance = IERC20(rewardTokens[i]).balanceOf(address(this));
54         if (balance > DURATION) {
55             uint256 fee = balance * IHiveFactory(hiveFactory).rewardFee() / DIVISOR;
56             balance -= fee;
57
58             IERC20(rewardTokens[i]).safeTransfer(IHiveToken(hiveToken).treasury(), fee * 10 / 100);
59             IERC20(rewardTokens[i]).safeTransfer(IHiveFactory(hiveFactory).protocol(), fee * 90 / 100);
60
61             IERC20(rewardTokens[i]).safeApprove(hiveRewarder, 0);
62             IERC20(rewardTokens[i]).safeApprove(hiveRewarder, balance);
63             IHiveRewarder(hiveRewarder).notifyRewardAmount(rewardTokens[i], balance);
64             emit HiveDistro__RewardDistributed(rewardTokens[i], balance);
65         }
66     }
67 }
```

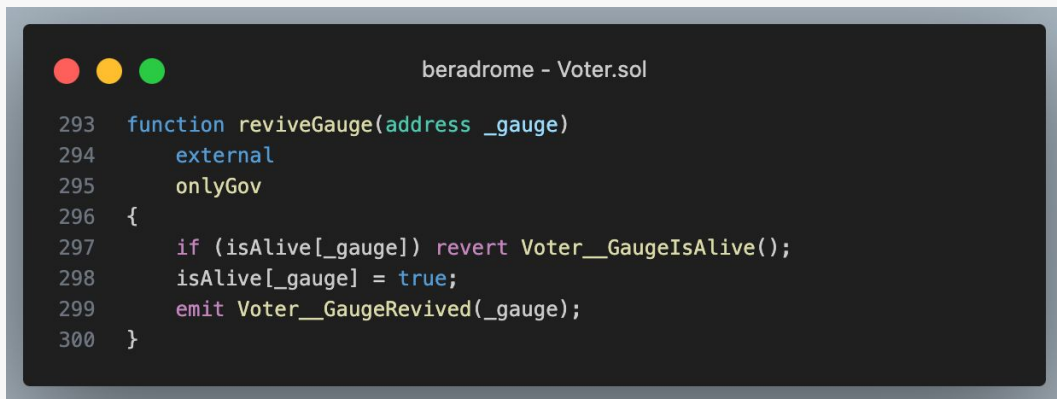
Remediation Check the remaining reward balance after deducting the fee, and only proceed with transferring the balance to the `HiveRewarder` contract if the remaining balance exceeds the predefined `DURATION` threshold.

3.3.4 [M-2] Revisited Logic of Voter::reviveGauge()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Voter.sol	Business Logic	Medium	Medium	Addressed

The [Voter](#) contract implements a mechanism that allows the governance (gov) to deactivate and later revive a gauge. A deactivated gauge ceases to accumulate OTOKEN rewards until it is revived through the [reviveGauge\(\)](#) function. However, upon reviewing the [reviveGauge\(\)](#) function, we identified that it does not properly update the reward index of the gauge to the latest value. As a result, a revived gauge may retroactively accumulate rewards from prior periods, leading to an unintended reward allocation.

To prevent the accumulation of unexpected rewards by a revived gauge, we recommend invoking the [_updateFor\(\)](#) function for the gauge within the [reviveGauge\(\)](#) function. This will ensure that the gauge's reward index is updated to the latest value upon revival, thereby allowing the gauge to accumulate new rewards only from that point onward.



```
beradrome - Voter.sol

293 function reviveGauge(address _gauge)
294     external
295     onlyGov
296 {
297     if (isAlive[_gauge]) revert Voter__GaugeIsAlive();
298     isAlive[_gauge] = true;
299     emit Voter__GaugeRevived(_gauge);
300 }
```

Remediation Timely invoking the [_updateFor\(\)](#) function for the revived gauge within the [reviveGauge\(\)](#) function to update its reward index to the latest value.

Response By Team The [reviveGauge\(\)](#) function is not used and will be removed from the [Voter](#) contract to avoid complexity.

3.3.5 [M-3] Potential Denial-of-Service in KodiakPlugin::withdrawTo()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
KodiakPluginFactory.sol	Business Logic	Medium	Medium	Addressed

The [KodiakPlugin](#) plugin is designed to lock users' deposits of a specified token into a farm to earn rewards. During the withdrawal operation, the [withdrawTo\(\)](#) function follows a specific flow: it first withdraws all locked tokens from the farm, transfers the requested withdrawal amount to the user, and then re-locks the remaining tokens back into the farm.

However, if the remaining token balance after the user's withdrawal is zero, the subsequent attempt to lock a zero balance to the farm could revert the transaction. This would cause the entire withdrawal operation to fail, effectively resulting in a denial-of-service (DoS) scenario for the user.

beradrome - KodiakPluginFactory.sol

```
90  function withdrawTo(address account, uint256 amount)
91      public
92      override
93  {
94      ICommunalFarm(farm).withdrawLockedAll();
95      super.withdrawTo(account, amount);
96      uint256 balance = IERC20(getToken()).balanceOf(address(this));
97      IERC20(getToken()).safeApprove(farm, 0);
98      IERC20(getToken()).safeApprove(farm, balance);
99      ICommunalFarm(farm).stakeLocked(balance, 0);
100 }
```

Remediation Add a validation check to ensure the remaining token balance is greater than zero before calling the [stakeLocked\(\)](#) function.

3.3.6 [M-4] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

In the [Beradrome](#) protocol, the existence of a privileged owner account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged account.

```
beradrome - Voter.sol
283 function killGauge(address _gauge)
284     external
285     onlyGov
286 {
287     if (!isAlive[_gauge]) revert Voter__GaugeIsDead();
288     isAlive[_gauge] = false;
289     claimable[_gauge] = 0;
290     emit Voter__GaugeKilled(_gauge);
291 }
292
293 function reviveGauge(address _gauge)
294     external
295     onlyGov
296 {
297     if (isAlive[_gauge]) revert Voter__GaugeIsAlive();
298     isAlive[_gauge] = true;
299     emit Voter__GaugeRevived(_gauge);
300 }
```

Remediation To mitigate the issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team The owner will be the team multisig and can be transitioned to governance in the future if required.

3.3.7 [L-1] Improved Rounding Operation in TOKEN

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
TOKEN.sol	Mathematics	Low	Medium	Addressed

The `buy()` function in the TOKEN contract allows users to purchase TOKEN from the bonding curve market reserves using BASE. During this process, the function calculates the `newMrTOKEN` value based on the bonding curve formula. However, we notice that the calculation employs a default rounding-down operation. As a result, the `newMrTOKEN` value may be slightly (by 1 WEI) lower than expected. Consequently, the output TOKEN amount, calculated as `outTOKEN = mrrTOKEN - newMrTOKEN`, becomes slightly higher than expected. This discrepancy leads to a minor loss of TOKEN from the bonding curve market reserves, reducing the reserves' overall health.

```
beradrome - TOKEN.sol

179 function buy(uint256 amountBase, uint256 minToken, uint256 expireTimestamp, address toAccount, address provider)
180     external
181     nonReentrant
182     nonZeroInput(amountBase)
183     nonExpiredSwap(expireTimestamp)
184     returns (bool)
185 {
186     uint256 feeBASE = amountBase * SWAP_FEE / DIVISOR;
187     uint256 newMrBASE = (mrvBASE + mrrBASE) + amountBase - feeBASE;
188     uint256 newMrTOKEN = (mrvBASE + mrrBASE) * mrrTOKEN / newMrBASE;
189     uint256 outTOKEN = mrrTOKEN - newMrTOKEN;
190
191     if (outTOKEN < minToken) revert TOKEN__ExceedsSwapSlippageTolerance();
192
193     mrrBASE = newMrBASE - mrvBASE;
194     mrrTOKEN = newMrTOKEN;
195
196     emit TOKEN__Buy(msg.sender, toAccount, amountBase);
197     ...
198 }
```

Remediation Revise the rounding mechanism in the `buy()` function. Specifically, the calculation of `newMrTOKEN` should utilize rounding up rather than rounding down to maintain the long-term stability of the bonding curve market reserves.

3.3.8 [L-2] Revisited Validation of amountToken in sell()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
TOKEN.sol	Business Logic	Low	Medium	Addressed

In the TOKEN contract, the `sell()` function allows users to sell TOKEN for BASE. Before proceeding, the function checks whether the input `amountToken` exceeds the maximum sellable amount, as determined by the `getMaxSell()` function. This ensures that the market reserves can handle the transaction.

However, upon reviewing the `getMaxSell()` function, we identified that it calculates the maximum sellable amount based on the $k = xy$ invariant without accounting for the swap fee deducted from the input `amountToken`. Consequently, the calculated maximum input amount is smaller than expected, potentially limiting users' ability to sell their tokens effectively.

beradrome - TOKEN.sol

```
354 function getMaxSell() public view returns (uint256) {
355     return (mrrTOKEN * mrrBASE / mrvBASE);
356 }
```

Remediation Improve the `getMaxSell()` function to incorporate the swap fee in its calculation. This adjustment will ensure that the maximum sellable amount reflects the actual market conditions and prevents unnecessary limitations on user transactions.

3.3.9 [L-3] Incompatibility with Non-ERC20 Tokens

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Compatibility	Low	Medium	Addressed

The BribePot contract contains a vulnerability related to the use of non-standard ERC20 tokens. Specifically, the `approve()` function, as defined within the IERC20 interface, expects a boolean return value, but non-standard ERC20 tokens may not return this value, potentially causing the function to fail. This issue is observed in the `distribute()` function where `IERC20(token).approve(address(bribe), balance)` is called directly. The problem may also exist in other files where `approve()` is used without proper handling.

beradrome - AuctionFactory.sol

```
27 function distribute() external {
28     address token = IFund(fund).asset();
29     address bribe = IFund(fund).getBribe();
30     uint256 balance = IERC20(token).balanceOf(address(this));
31     if (balance > IBribe(bribe).left(address(token))) {
32         IERC20(token).approve(address(bribe), 0);
33         IERC20(token).approve(address(bribe), balance);
34         IBribe(bribe).notifyRewardAmount(address(token), balance);
35     }
36 }
```

Remediation It is recommended to replace the direct use of `approve()` with the `safeApprove()` function from the OpenZeppelin library to safely handle non-standard ERC20 tokens and mitigate the risk of function failure.

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	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI