// HALBORN

Opal Finance - Protocol

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **January 08th, 2024 - February 12th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	01/15/2024
0.2	Document Update	02/12/2024
0.3	Draft Review	02/12/2024
0.4	Draft Review	02/12/2024
1.0	Remediation Plan	03/04/2024
1.1	Remediation Plan Review	03/11/2024
1.2	Remediation Plan Review	03/12/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

The protocol manages the distribution of rewards obtained by omnipools.

Opal Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on January 08th, 2024 and ending on February 12th, 2024. The security assessment was scoped to the smart contracts provided in the OpalProtocol/contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 6 weeks for the engagement and assigned a full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, that were mostly addressed by Opal Finance. The main ones were the following:

- Restrict the approve() and swapForGem() functions of the Omnipool contract to the RewardManager.
- Set the transaction lock in the depositFor() function of the Omnipool contract for the recipient, not the function caller.
- Fix the getUSDPrice() function of the BPTOracle to handle tokens with non-standard token decimals, and do not assume the price feed precision.
- Fix the usage of the continuous statement in the for loops to prevent infinite execution.
- Fix the signature validation in the permit() function of the LiquidityGauge contract.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Opal Contracts

- Repository: OpalProtocol/contracts
- **Commit ID** : 3109328ed9bb647e98de08beb5999f464702aba5
- Smart contracts in scope:
  - src/pools/BPTOracle.sol
  - src/pools/Omnipool.sol
  - src/pools/OmnipoolController.sol
  - src/pools/OpalLpToken.sol
  - src/tokenomics/EscrowedToken.sol
  - src/tokenomics/GaugeController.sol
  - src/tokenomics/MinterEscrow.sol
  - src/tokenomics/VoteLocker.sol
  - src/tokenomics/GaugeFactory.sol
  - src/tokenomics/Minter.sol
  - src/tokenomics/LiquidityGauge.sol
  - src/tokenomics/GemMinterRebalancingReward.sol
  - src/RewardManager.sol
- Last remediation commit ID: e710f6cd208da85853fc1de877a8627fe5bd81bf

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.
- New features/implementations after/within the 3109328 & e710f6c commit IDs.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	1	5	4	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) ERC20 TOKENS CAN BE DRAINED FROM OMNIPOOL	Critical (10)	SOLVED - 02/09/2024
(HAL-02) LACK OF AUTHORIZATION CHECK IN SWAPFORGEM	Critical (10)	SOLVED - 02/08/2024
(HAL-03) WITHDRAWAL DELAY CAN BYPASSED	Critical (10)	SOLVED - 02/07/2024
(HAL-04) GETUSDPRICE INCORRECTLY HANDLES TOKEN DECIMALS	Critical (10)	SOLVED - 02/25/2024
(HAL-05) IMPROPER IMPLEMENTATION OF THE MINIMAL PROXY STANDARD	Critical (10)	SOLVED - 02/09/2024
(HAL-06) IMPROPER LOOP IMPLEMENTATIONS	High (7.5)	SOLVED - 02/09/2024
(HAL-07) LACK OF SIGNATURE VALIDATION	Medium (5.0)	SOLVED - 02/09/2024
(HAL-08) OPALLPTOKEN DECIMALS ARE NOT SET CORRECTLY	Medium (5.0)	SOLVED - 02/07/2024
(HAL-09) LACK OF STALENESS CHECK IN GETUSDPRICE	Medium (5.0)	SOLVED - 02/25/2024
(HAL-10) APPROVE IS INCOMPATIBLE WITH NON-STANDARD ERC20 TOKENS	Medium (5.0)	SOLVED - 02/05/2024
(HAL-11) USING TRANSFER INSTEAD OF SAFETRANSFER	Medium (5.0)	SOLVED - 02/05/2024
(HAL-12) PRICE FEED PRECISION IS ASSUMED IN GETUSDPRICE	Low (3.4)	SOLVED - 03/04/2024
(HAL-13) IMPROPER HANDLEDEPEGGEDPOOL IMPLEMENTATION	Low (3.4)	SOLVED - 03/04/2024
(HAL-14) PRICE FEED ORACLE ADDRESS CANNOT BE UPDATED	Low (2.5)	SOLVED - 03/04/2024
(HAL-15) MINUNDERLYINGRECEIVED INCLUDES THE FEES IN OMNIPOOL	Low (2.5)	SOLVED - 02/11/2024
(HAL-16) DOMAINSEPARATOR CANNOT BE REGENERATED	Informational (1.7)	SOLVED - 02/09/2024

(HAL-17) CHECKS-EFFECTS-INTERACTIONS PATTERN IS NOT FOLLOWED IN DEPOSITFOR AND WITHDRAW	Informational (1.7)	ACKNOWLEDGED
(HAL-18) LACK OF EMERGENCY STOP PATTERN IMPLEMENTATION	Informational (1.7)	ACKNOWLEDGED
(HAL-19) LACK OF ZERO ADDRESS CHECKS	Informational (1.7)	ACKNOWLEDGED
(HAL-20) REDUNDANT LOCK CHECK IN DEPOSIT	Informational (0.0)	SOLVED - 02/07/2024
(HAL-21) HARDCODED CONFIGURATION AND ADDRESSES	Informational (0.0)	SOLVED - 03/04/2024
(HAL-22) UNUSED CODE	Informational (0.0)	SOLVED - 02/24/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) ERC20 TOKENS CAN BE DRAINED FROM OMNIPOOL - CRITICAL(10)

Description:

This approve function of the Omnipool contract is used by the RewardManager to withdraw the reward tokens from the contract and distribute them to the users. However, it was identified that the function lacks any authorization check. By calling this function, anyone can withdraw the LP and reward tokens from the contract.

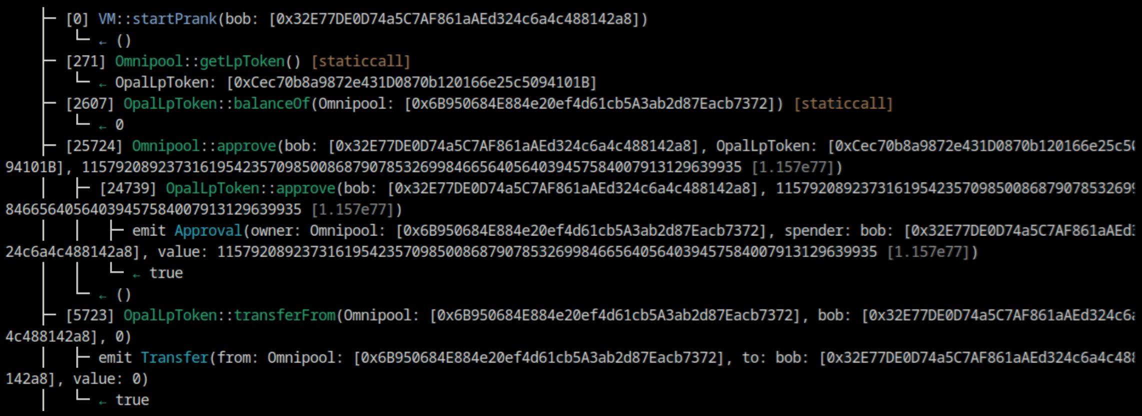
Code Location:

Listing 1: src/pools/Omnipool.sol

```
    function approve(address addr, address token, uint256 amount)
 ↳ external {
        // Transfer the rewards to the user
        IERC20 erc20 = IERC20(token);
        erc20.approve(addr, amount);
    }
```

Proof of Concept:

1. Users deposit funds into the pool.
2. Bob calls the Omnipool's approve function and authorizes himself to transfer out the LP tokens from the contract.
3. Bob transfers out the LP tokens from the contract.



```
    ├─ [0] VM::startPrank(bob: [0x32E77DE0D74a5C7AF861aAEd324c6a4c488142a8])
    │   └─ ← ()
    ├─ [271] Omnipool::getLpToken() [staticcall]
    │   └─ ← OpalLpToken: [0xCec70b8a9872e431D0870b120166e25c5094101B]
    ├─ [2607] OpalLpToken::balanceOf(Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372]) [staticcall]
    │   └─ ← 0
    ├─ [25724] Omnipool::approve(bob: [0x32E77DE0D74a5C7AF861aAEd324c6a4c488142a8], OpalLpToken: [0xCec70b8a9872e431D0870b120166e25c5
94101B], 115792089237316195423570985008687907853269984665640564039457584007913129639935 [1.157e77])
    │   ├─ [24739] OpalLpToken::approve(bob: [0x32E77DE0D74a5C7AF861aAEd324c6a4c488142a8], 11579208923731619542357098500868790785326
846656405640394575840079131296399935 [1.157e77])
    │   │   ├─ emit Approval(owner: Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372], spender: bob: [0x32E77DE0D74a5C7AF861aAEd
24c6a4c488142a8], value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 [1.157e77])
    │   │   └─ ← true
    │   └─ ← ()
    ├─ [5723] OpalLpToken::transferFrom(Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372], bob: [0x32E77DE0D74a5C7AF861aAEd324c6
4c488142a8], 0)
    │   ├─ emit Transfer(from: Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372], to: bob: [0x32E77DE0D74a5C7AF861aAEd324c6a4c48
142a8], value: 0)
    │   └─ ← true
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to restrict the approve() function to the RewardManager contract.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit ab517b0 by restricting the function to the reward manager contract.

4.2 (HAL-02) LACK OF AUTHORIZATION CHECK IN SWAPFORGEM - CRITICAL(10)

Description:

The RewardManager claims the extra rewards from the Omnipool contract by swapping the underlying reward tokens to GEM tokens. However, it was identified that the swapForGem function in the Omnipool contract lacks any authorization check. By calling this function, any caller can swap the tokens (e.g., LP pool tokens) stored in the contract to GEM reward tokens. This results in burning the LP tokens of the Omnipool resulting in loss of funds. The large swap can also be exploited using a sandwich attack to create profit for the attacker.

Code Location:

The swapForGem() function lacks authorization:

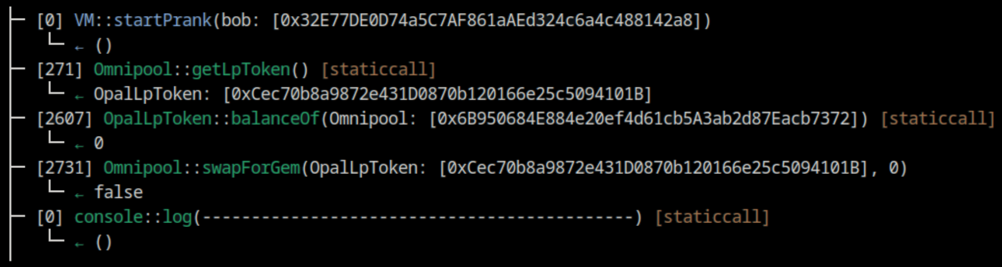
Listing 2: src/pools/Omnipool.sol

```
    function swapForGem(address _token, uint256 _amountIn)
  ↳ external returns (bool) {
        bytes32 poolId = extraRewardPools[_token];
        if (poolId == bytes32(0) || wethToGemPoolId == bytes32(0))
  ↳  {
            return false;
        }

        IERC20 erc20Token = IERC20(_token);
        erc20Token.approve(address(balancerVault), _amountIn);
```

Proof of Concept:

1. Users deposit funds into the pool.
2. Bob calls the swapForGem function to swap the LP tokens to GEM reward tokens.
3. Bob sandwiches the swap operation and realizes a considerable profit.



```
├─ [0] VM::startPrank(bob: [0x32E77DE0D74a5C7AF861aAEd324c6a4c488142a8])
│   └─ ← ()
├─ [271] Omnipool::getLpToken() [staticcall]
│   └─ ← OpalLpToken: [0xCec70b8a9872e431D0870b120166e25c5094101B]
├─ [2607] OpalLpToken::balanceOf(Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372]) [staticcall]
│   └─ ← 0
├─ [2731] Omnipool::swapForGem(OpalLpToken: [0xCec70b8a9872e431D0870b120166e25c5094101B], 0)
│   └─ ← false
├─ [0] console::log(------------------------------------------) [staticcall]
│   └─ ← ()
│
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to restrict the swapForGem() function to the RewardManager contract.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit c6e26f9 by restricting the function to the reward manager contract.

4.3 (HAL-03) WITHDRAWAL DELAY CAN BYPASSED - CRITICAL(10)

Description:

Users can't deposit and withdraw in the same block in the Omnipool contract. However, it was identified that this restriction could be bypassed by depositing from another user to the recipient address with the depositFor() function and then withdrawing from the recipient in the same block. This allows users to generate rewards in the same block and potentially drain the rewards from the protocol.

Code Location:

The withdrawal lock is only checked and updated for the msg.sender:

Listing 3: src/pools/Omnipool.sol

```
    function depositFor(uint256 _amountIn, address _depositFor, uint256 _minLpReceived) public {
        if (lastTransactionBlock[msg.sender] == block.number) {
            revert CantDepositAndWithdrawSameBlock();
        }

        uint256 underlyingPrice = bptOracle.getUSDPrice(address(underlyingToken));
```

Listing 4: src/pools/Omnipool.sol (Line 267)

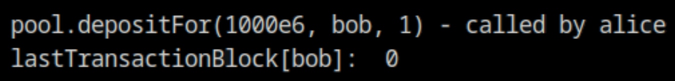
```
260         _handleRebalancingRewards(
261             msg.sender,
262             beforeTotalUnderlying,
263             afterTotalUnderlying,
264             beforeAllocatedPerPool,
265             afterAllocatedPerPool
266         );
267         lastTransactionBlock[msg.sender] = block.number;
268     }
```

Listing 5: src/pools/Omnipool.sol

```
    function withdraw(uint256 _amountOut, uint256 _minUnderlyingReceived) external override {
        if (lastTransactionBlock[msg.sender] == block.number) {
            revert CantDepositAndWithdrawSameBlock();
        }
```

Proof of Concept:

1. The depositFor() function is used to deposit funds for Bob from different addresses (e.g., using transaction batching or a smart contract).
2. The lastTransactionBlock is not updated for Bob.
3. Bob can withdraw in the same transaction to perform a sandwich attack.



```
pool.depositFor(1000e6, bob, 1) - called by alice
lastTransactionBlock[bob]:  0
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to set the transaction lock in the depositFor() function for the recipient, not the msg.sender.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 289ccaa by setting the lock for the recipient.

4.4 (HAL-04) GETUSDPRICE INCORRECTLY HANDLES TOKEN DECIMALS - CRITICAL(10)

Description:

The getUSDPrice function of the BPTOracle contract returns the USD price of the parameter token with 18 decimals precision. However, it was identified that the function could not handle tokens with different decimals than 6 or 18, resulting in incorrect price data in those cases.

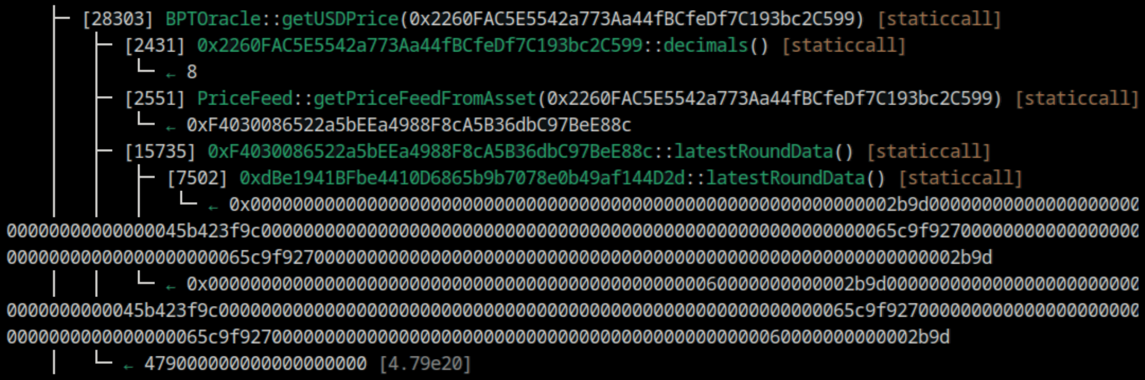
Code Location:

Listing 6: src/pools/BPTOracle.sol (Lines 217-219)

```
211     function getUSDPrice(address token) public view returns (
  ↳ uint256 priceInUSD) {
212         uint256 decimals = ERC20(token).decimals();
213
214         AggregatorV3Interface priceFeed = IPriceFeed(
  ↳ priceFeedAddress).getPriceFeedFromAsset(token);
215         if (address(priceFeed) == address(0)) revert
  ↳ PriceFeedNotFound();
216         (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
217         if (decimals < 18) {
218             return uint256(priceInUSDInt) * 10 ** (18 - decimals -
  ↳  2);
219         }
220         return uint256(priceInUSDInt) * 1e10;
221     }
```

Proof of Concept:

Incorrect price is calculated for bitcoin (4.79e20 instead of 4.79e22):



```
    ├─ [28303] BPTOracle::getUSDPrice(0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599) [staticcall]
    │   ├─ [2431] 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599::decimals() [staticcall]
    │   │   └─ ← 8
    │   ├─ [2551] PriceFeed::getPriceFeedFromAsset(0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599) [staticcall]
    │   │   └─ ← 0xF4030086522a5bEEa4988F8cA5B36dbC97BeE88c
    │   ├─ [15735] 0xF4030086522a5bEEa4988F8cA5B36dbC97BeE88c::latestRoundData() [staticcall]
    │   │   ├─ [7502] 0xdBe1941BFbe4410D6865b9b7078e0b49af144D2d::latestRoundData() [staticcall]
    │   │   │   └─ ← 0x0000000000000000000000000000000000000000000000000000000000002b9d000000000000000000
000000000000000045b423f9c0000000000000000000000000000000000000000000000000000000065c9f92700000000000000000
0000000000000000000065c9f9270000000000000000000000000000000000000000000000000000000000002b9d
    │   │   └─ ← 0x00000000000000000000000000000000000000000000000600000000000002b9d0000000000000000000000
0000000000045b423f9c0000000000000000000000000000000000000000000000000000000065c9f927000000000000000000000
000000000000000065c9f92700000000000000000000000000000000000000000000000600000000000002b9d
    │   └─ ← 479000000000000000000 [4.79e20]
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

It is recommended that the getUSDPrice() function of the BPTOracle contract be modified to handle tokens with non-standard token decimals.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit e710f6c by normalizing the returned amounts to 18 decimals.

4.5 (HAL-05) IMPROPER IMPLEMENTATION OF THE MINIMAL PROXY STANDARD - CRITICAL(10)

Description:

It was identified that the LiquidityGauge contract improperly utilized the EIP-1167: Minimal Proxy Standard, as it sets the values of non-immutable state variables in its constructor. These values are not copied into the clones, and therefore, the LiquidityGauge deployed by the factory will have uninitialized state variables that cannot be changed later.

Code Location:

The registryContract and registryAccess state variables are not immutable:

Listing 7: src/tokenomics/LiquidityGauge.sol

```
52     IRegistryContract public registryContract;
53     IRegistryAccess public registryAccess;
```

However, they are initialized in the constructor():

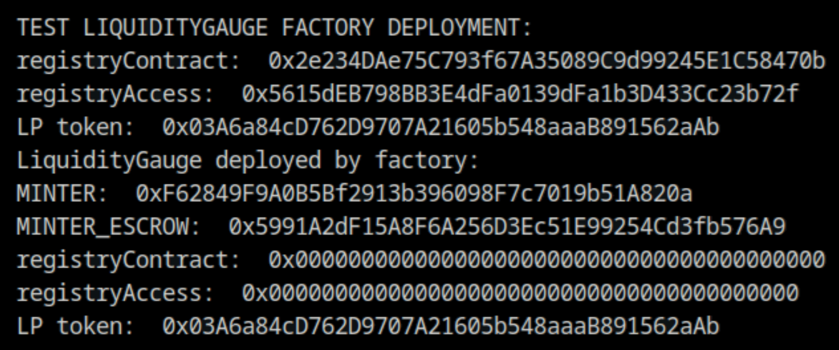
Listing 8: src/tokenomics/LiquidityGauge.sol (Lines 140-141)

```
131     constructor(
132         address _minter,
133         address _minterEscrow,
134         address _vlToken,
135         address _registryContract
136     ) {
137         MINTER = _minter;
138         MINTER_ESCROW = _minterEscrow;
139         VL_TOKEN = _vlToken;
140         registryContract = IRegistryContract(_registryContract);
141         registryAccess = IRegistryAccess(registryContract.
    ↳ getContract(CONTRACT_REGISTRY_ACCESS));
142
```

```
        GAUGE_CONTROLLER = registryContract.getContract(
↳ CONTRACT_GAUGE_CONTROLLER);

        lpToken = address(0);
    }
```

Proof of Concept:



```
TEST LIQUIDITYGAUGE FACTORY DEPLOYMENT:
registryContract:  0x2e234DAe75C793f67A35089C9d99245E1C58470b
registryAccess:  0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
LP token:  0x03A6a84cD762D9707A21605b548aaaB891562aAb
LiquidityGauge deployed by factory:
MINTER:  0xF62849F9A0B5Bf2913b396098F7c7019b51A820a
MINTER_ESCROW:  0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9
registryContract:  0x0000000000000000000000000000000000000000
registryAccess:  0x0000000000000000000000000000000000000000
LP token:  0x03A6a84cD762D9707A21605b548aaaB891562aAb
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to only initialize immutable state variables in the contractor.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 7a52757 by adding the immutable modifier to the state variables.

4.6 (HAL-06) IMPROPER LOOP IMPLEMENTATIONS - HIGH (7.5)

Description:

It was identified that the following functions do not always increase the loop counter before continuing to the next cycle. If the condition having the continue statement executes, the loop cycle is repeated forever, and the functions will eventually revert after exhausting all gas. This can cause the protocol to enter a denial of service state because these functions are used in several places in the protocol, and the contracts cannot be upgraded.

src/tokenomics/EscrowedToken.sol
- claimAll()
src/tokenomics/Minter.sol
- mintMultiple()
src/tokenomics/VoteLocker.sol
- totalSupplyAtEpoch()
- getReward()

Code Location:

The following is an example from the VoteLocker contract. If the condition is met, the loop will never end, and the function will eventually revert:

Listing 9: src/tokenomics/VoteLocker.sol (Line 952)

```
    function totalSupplyAtEpoch(uint256 _epoch) public view
 ↳ returns (uint256 supply) {
        uint256 epochStart = uint256(_epochs[0].date).add(uint256(
 ↳ _epoch).mul(rewardsDuration));
        if (epochStart >= block.timestamp) revert FutureEpoch();
Fix t
        uint256 cutoffEpoch = epochStart.sub(lockDuration);
        uint256 lastIndex = _epochs.length - 1;

        uint256 epochIndex = _epoch > lastIndex ? lastIndex :
 ↳ _epoch;

        for (uint256 i = epochIndex + 1; i > 0;) {
            Epoch memory e = _epochs[i - 1];
            if (e.date == epochStart) {
                continue;
            } else if (e.date <= cutoffEpoch) {
                break;
            } else {
                supply += e.supply;
            }
            unchecked {
                --i;
            }
        }
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (7.5)

Recommendation:

It is recommended to increase the loop counter before continuing to the next cycle.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 7a52757.

4.7 (HAL-07) LACK OF SIGNATURE VALIDATION - MEDIUM (5.0)

Description:

It was identified that the permit() function of the LiquidityGauge contract improperly utilizes the isValidSignatureNow() function. Instead of the hash and signature, the domainSeparator and the structHash values are passed to this function. This results in failing the signature check every time. It is also noted that the permit() function has no signature parameter.

Code Location:

The signatuere based validation is implemented improperly in the permit() function:

Listing 10: src/tokenomics/LiquidityGauge.sol (Line 323)

```
    function permit(address owner, address spender, uint256 value,
 ↳ uint256 deadline)
        public
        virtual
    {
        if (owner == address(0)) revert AddressZero();
        if (block.timestamp > deadline) revert SignatureExpired();

        uint256 userNonce = nonces[owner];
        bytes memory structHash = abi.encode(
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender,
 ↳ value, userNonce, deadline))
        );

        bool signerOkay = SignatureChecker.isValidSignatureNow(
 ↳ owner, domainSeparator, structHash);

        // bytes32 _hash = ECDSA.recover(domainSeparator,
 ↳ structHash);

        if (!signerOkay) revert SignatureInvalid();

        allowance[owner][spender] = value;
        nonces[owner]++;
    }
```

The isValidSignatureNow() function has different parameterization:

Listing 11: lib/openzeppelin-contracts/contracts/utils/cryptography/SignatureChecke

```
    function isValidSignatureNow(address signer, bytes32 hash,
↳ bytes memory signature) internal view returns (bool) {
        (address recovered, ECDSA.RecoverError error, ) = ECDSA.
↳ tryRecover(hash, signature);
        return
            (error == ECDSA.RecoverError.NoError && recovered ==
↳ signer) ||
            isValidERC1271SignatureNow(signer, hash, signature);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to add a signature validation to the permit() function.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 720a833.

4.8 (HAL-08) OPALLPTOKEN DECIMALS ARE NOT SET CORRECTLY - MEDIUM (5.0)

Description:

It was identified that configuring the decimals in the constructor of the OpalLpToken contract is not working, and all tokens will have 18 decimals. This might result in calculation errors, as the underlying assets of the pools can have different decimals.

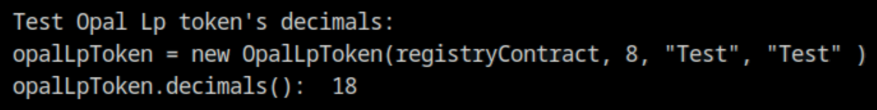
Code Location:

The decimals are configured in the constructor:

Listing 12: src/pools/OpalLpToken.sol (Line 45)

```
36     constructor(
37         address _registryContract,
38         uint8 _decimals,
39         string memory name,
40         string memory symbol
41     ) ERC20(name, symbol) {
42         registryContract = IRegistryContract(_registryContract);
43         registryAccess = IRegistryAccess(registryContract.
   ↳ getContract(CONTRACT_REGISTRY_ACCESS));
44 
45         __decimals = _decimals;
46     }
```

Proof of Concept:



```
Test Opal Lp token's decimals:
opalLpToken = new OpalLpToken(registryContract, 8, "Test", "Test" )
opalLpToken.decimals():  18
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to fix the OpalLpToken contract by overriding the decimals() function to correctly show the configured value.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 289ccaa.

4.9 (HAL-09) LACK OF STALENESS CHECK IN GETUSDPRICE - MEDIUM (5.0)

Description:

The getUSDPrice function of the BPTOracle contract returns the USD price of the parameter token. However, it was identified that the function does not check whether the received data is out of date and valid.

Code Location:

The priceFeed does not check whether the received data is out of date and valid:

Listing 13: src/pools/BPTOracle.sol (Line 216)

```
211     function getUSDPrice(address token) public view returns (
    ↳ uint256 priceInUSD) {
212         uint256 decimals = ERC20(token).decimals();
213
214         AggregatorV3Interface priceFeed = IPriceFeed(
    ↳ priceFeedAddress).getPriceFeedFromAsset(token);
215         if (address(priceFeed) == address(0)) revert
    ↳ PriceFeedNotFound();
216         (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
217         if (decimals < 18) {
218             return uint256(priceInUSDInt) * 10 ** (18 - decimals -
    ↳  2);
219         }
220         return uint256(priceInUSDInt) * 1e10;
221     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to reject prices older than the threshold corresponding to the heartbeat of the price feed.

The staleness threshold should correspond to the heartbeat of the oracle’s price feed.

On L2 chains like Arbitrum, it is also recommended to check whether the L2 Sequencer is down to avoid stale pricing data that appears fresh.

References:

Check the timestamp of the latest answer

L2 Sequencer Uptime Feeds

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit e710f6c.

4.10 (HAL-10) APPROVE IS INCOMPATIBLE WITH NON-STANDARD ERC20 TOKENS - MEDIUM (5.0)

Description:

Some tokens do not correctly implement the EIP20 standard, and their approve function returns void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will always revert. Tokens that do not correctly implement the latest EIP20 spec, like USDT on Ethereum, will be unusable in the mentioned contracts as they revert the transaction because of the missing return value.
Some tokens also require that the allowance be set to 0 before issuing a new approve call. Calling the approve function when the allowance is not zero reverts the transaction with these types of tokens.

Code Location:

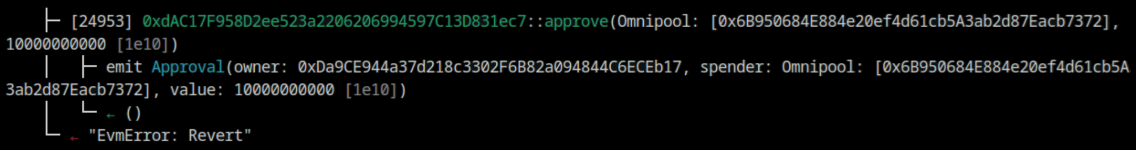
Example usage of the approve function in the protocol:

Listing 14: src/pools/Omnipool.sol (Line 233)

```
226     function depositFor(uint256 _amountIn, address _depositFor, uint256 _minLpReceived) public {
227         if (lastTransactionBlock[msg.sender] == block.number) {
228             revert CantDepositAndWithdrawSameBlock();
229         }
230
231         uint256 underlyingPrice = bptOracle.getUSDPrice(address(underlyingToken));
232
233         underlyingToken.approve(address(auraRewardPoolDepositWrapper), _amountIn);
234
235         (
236             uint256 beforeTotalUnderlying,
237             uint256 beforeAllocatedBalance,
238             uint256[] memory beforeAllocatedPerPool
```

```
            ) = _getTotalAndPerPoolUnderlying(underlyingPrice);
```

Proof of Concept:



```
    ├ [24953] 0xdAC17F958D2ee523a2206206994597C13D831ec7::approve(Omnipool: [0x6B950684E884e20ef4d61cb5A3ab2d87Eacb7372],
10000000000 [1e10])
    │   ├ emit Approval(owner: 0xDa9CE944a37d218c3302F6B82a094844C6ECEb17, spender: Omnipool: [0x6B950684E884e20ef4d61cb5A
3ab2d87Eacb7372], value: 10000000000 [1e10])
    │   └ ← ()
    └ ← "EvmError: Revert"
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to use OpenZeppelin's SafeERC20 and the forceApprove() function to also handle non-standard-compliant tokens.

References:

OpenZeppelin's SafeERC20

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 6cba12e by using OpenZeppelin's SafeERC20 and the forceApprove() function.

4.11 (HAL-11) USING TRANSFER INSTEAD OF SAFETRANSFER - MEDIUM (5.0)

Description:

It was identified that several functions in the contracts use the IERC20Metadata and IERC20 interfaces to interact with tokens. However, the interface expects the transfer function to have a return value on success. It is important to note that the transfer functions of some tokens (e.g., USDT, BNB) do not return any values, so these tokens are incompatible with the current version of the contracts.

Code Location:

Example usage of the transferFrom function in the protocol:

Listing 15: src/pools/Omnipool.sol (Line 244)

```
    function depositFor(uint256 _amountIn, address _depositFor,
 ↳ uint256 _minLpReceived) public {
        if (lastTransactionBlock[msg.sender] == block.number) {
            revert CantDepositAndWithdrawSameBlock();
        }

        uint256 underlyingPrice = bptOracle.getUSDPrice(address(
 ↳ underlyingToken));

        underlyingToken.approve(address(
 ↳ auraRewardPoolDepositWrapper), _amountIn);

        (
            uint256 beforeTotalUnderlying,
            uint256 beforeAllocatedBalance,
            uint256[] memory beforeAllocatedPerPool
        ) = _getTotalAndPerPoolUnderlying(underlyingPrice);

        uint256 exchangeRate = _exchangeRate(beforeTotalUnderlying
 ↳ );

        // Transfer underlying token to this contract
        underlyingToken.transferFrom(msg.sender, address(this),
↳ _amountIn);

        _depositToAura(beforeAllocatedBalance,
↳ beforeAllocatedPerPool, _amountIn);
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to use OpenZeppelin's SafeERC20 wrapper with the IERC20 and IERC20Metadata interfaces to make the contracts compatible with currencies that return no value.

References:

OpenZeppelin's SafeERC20

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 6cba12e5 by using OpenZeppelin's SafeERC20 wrapper.

4.12 (HAL-12) PRICE FEED PRECISION IS ASSUMED IN GETUSDPRICE - LOW (3.4)

Description:

The getUSDPrice function of the BPTOracle contract returns the USD price of the parameter token with 18 decimals precision. However, it was identified that the function assumes that the price value returned by the price feed always has 8 decimals. This is not necessarily true for all assets. For example, ETH pairs usually have 18 decimals. Some other pairs, like AMPL/USD, also have 18 decimals.

Code Location:

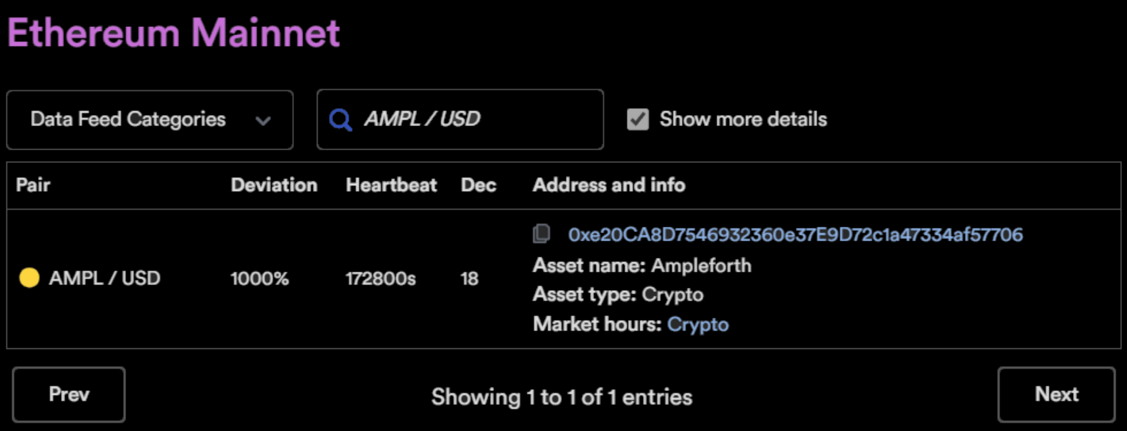
The getUSDPrice function assumes that the price value returned by the price feed always has 8 decimals:

Listing 16: src/pools/BPTOracle.sol (Line 216)

```
211     function getUSDPrice(address token) public view returns (uint256 priceInUSD) {
212         uint256 decimals = ERC20(token).decimals();
213
214         AggregatorV3Interface priceFeed = IPriceFeed(priceFeedAddress).getPriceFeedFromAsset(token);
215         if (address(priceFeed) == address(0)) revert PriceFeedNotFound();
216         (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
217         if (decimals < 18) {
218             return uint256(priceInUSDInt) * 10 ** (18 - decimals - 2);
219         }
220         return uint256(priceInUSDInt) * 1e10;
221     }
```

Proof of Concept:

AMPL/USD price feed information from the Chainlink documentation:


Ethereum Mainnet

Data Feed Categories | AMPL / USD | ✓ Show more details

Pair	Deviation	Heartbeat	Dec	Address and info
● AMPL / USD	1000%	172800s	18	0xe20CA8D7546932360e37E9D72c1a47334af57706 Asset name: Ampleforth Asset type: Crypto Market hours: Crypto

Prev | Showing 1 to 1 of 1 entries | Next

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (3.4)

Recommendation:

It is recommended to query the price feed in the getUSDPrice function of the BPTOracle contract to get the exact number of decimals for the price value and adjust it if it is necessary.

References:

Price Feed Contract Addresses

Remediation Plan:

SOLVED: The Opal Finance team solved the issue by implementing suggestion.

4.13 (HAL-13) IMPROPER HANDLEDEPEGGEDPOOL IMPLEMENTATION - LOW (3.4)

Description:

The updateDepegThreshold() function in the Omnipool contract is supposed to allow the Opal team to configure the depeg threshold. However, it was identified that the depegThreshold‘ state variable is unused in the contract, and therefore, this feature is not working.

It was also identified that the lpTokenPerPool mapping used in the handleDepeggedPool() function of the Omnipool contract is never initialized. This results in always passing the following check related to the status of the LP token.

Code Location:

The depegThreshold‘ state variable is unused in the contract:

{language="solidity" caption="src/pools/Omnipool.sol" firstnumber="631" function updateDepegThreshold(uint256 newDepegThreshold_)external onlyOpalTeam { if (newDepegThreshold_ > _MAX_DEPEG_THRESHOLD){ revert InvalidThreshold(); } if (newDepegThreshold_ < _MIN_DEPEG_THRESHOLD){ revert InvalidThreshold(); } depegThreshold = newDepegThreshold_; emit DepegThresholdUpdated(newDepegThreshold_); }

The getStatus() function returns 0 because the lpTokenPerPool mapping is never initialized:

Listing 17: src/pools/Omnipool.sol (Lines 928-933)

```solidity
    function handleDepeggedPool(address pool_) external {
        // Validation
        if (!_validatePool(pool_)) {
            revert PoolNotFound();
        }
        UnderlyingPool memory pool = getPoolByAddress(pool_);
        if (pool.targetWeight == 0) {
            return;
        }
        // != oracle.OracleStatus.oracleWorking
        if (oracle.getStatus(address(underlyingToken)) != 0) {
            return;
        }
        address lpToken_ = lpTokenPerPool[pool.poolAddress];

        // != oracle.OracleStatus.oracleWorking
        if (oracle.getStatus(lpToken_) != 0) {
            return;
        }

        // Set target pool weight to 0
        // Scale up other weights to compensate
        _setWeightToZero(pool_);
        rebalancingRewardActive = true;

        emit HandledDepeggedPool(pool_);
    }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (3.4)

Recommendation:

It is recommended that the correctness of the depeg handle functions be reviewed.

Remediation Plan:

SOLVED: The Opal Finance team made a business decision to accept the risk of this finding. The depeg handle functionality was removed from the contract as deemed unnecessary.

4.14 (HAL-14) PRICE FEED ORACLE ADDRESS CANNOT BE UPDATED - LOW (2.5)

Description:

The getUSDPrice function of the BPTOracle contract returns the USD price of the parameter token. The function is used in various places in the protocol. However, it was identified that the address of the price feed cannot be changed. If the price feed stops working, all the related functions will revert.

Code Location:

The priceFeed cannot be updated:

Listing 18: src/pools/BPTOracle.sol (Line 216)

```
211     function getUSDPrice(address token) public view returns (
 ↳ uint256 priceInUSD) {
212         uint256 decimals = ERC20(token).decimals();
213
214         AggregatorV3Interface priceFeed = IPriceFeed(
 ↳ priceFeedAddress).getPriceFeedFromAsset(token);
215         if (address(priceFeed) == address(0)) revert
 ↳ PriceFeedNotFound();
216         (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
217         if (decimals < 18) {
218             return uint256(priceInUSDInt) * 10 ** (18 - decimals -
 ↳  2);
219         }
220         return uint256(priceInUSDInt) * 1e10;
221     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add functionality to enable the Opal Team to update the price feed of the BPTOracle contract.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue by allowing edit price feed.

4.15 (HAL-15) MINUNDERLYINGRECEIVED INCLUDES THE FEES IN OMNIPOOL - LOW (2.5)

Description:

It was identified that the withdraw() function in the Omnipool contract checks the _minUnderlyingReceived value against the transferred amount before deducting the 5% fee. This may result in the user receiving fewer tokens than specified in the parameter.

Code Location:

Listing 19: src/pools/Omnipool.sol (Lines 369-371,378-381)

```
368         uint256 underlyingWithdrawn_ = _min(
 ↳ underlyingBalanceAfter_, underlyingToReceive_);
369         if (underlyingWithdrawn_ < _minUnderlyingReceived) {
370             revert TooMuchSlippage();
371         }
372         lastTransactionBlock[msg.sender] = block.number;
373         lpToken.burn(msg.sender, _amountOut);
374         totalDeposited -= underlyingWithdrawn_;
375
376         // Transfer 5% of the withdrew amount to the treasury
377         uint256 underlyingFees = underlyingWithdrawn_ * 5 / 100;
378         underlyingWithdrawn_ -= underlyingFees;
379         underlyingToken.transfer(OPAL_TREASURY, underlyingFees);
380         underlyingToken.transfer(msg.sender, underlyingWithdrawn_)
 ↳ ;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to deduct the fee from the transferred amount before comparing it to the _minUnderlyingReceived value.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit ea01c2d.

4.16 (HAL-16) DOMAINSEPARATOR CANNOT BE REGENERATED - INFORMATIONAL (1.7)

Description:

It was identified that the domainSeparator is generated and cached in the initialize() function of the LiquidityGauge contract, and it is not possible to change it later if the chain is forked. This allows an attacker to reuse the valid signatures of the permit function on both chains to transfer tokens.

Code Location:

The domainSeparator is initialized in the initialize() function and cannot be changed later:

Listing 20: src/tokenomics/LiquidityGauge.sol (Line 163)

```
152     function initialize(address _lpToken) external {
153         if (lpToken != address(0)) revert CannotInitialize();
154 
155         lpToken = _lpToken;
156         factory = msg.sender;
157 
158         string memory _symbol = IERC20Metadata(_lpToken).symbol();
159         string memory _name = string(abi.encodePacked("Opal ",
  ↳ _symbol, " Gauge Deposit"));
160         name = _name;
161         symbol = string(abi.encodePacked(_symbol, "-Gauge"));
162 
163         domainSeparator =
164             keccak256(abi.encode(EIP712_TYPEHASH, name, VERSION,
  ↳ block.chainid, address(this)));
165 
166         integrateInvSupply.push(0);
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to check the chain ID and regenerate the domain separator if it has changed.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 7a52757.

4.17 (HAL-17) CHECKS-EFFECTS-INTERACTIONS PATTERN IS NOT FOLLOWED IN DEPOSITFOR AND WITHDRAW - INFORMATIONAL (1.7)

Description:

It was identified that the depositFor and withdraw functions in the Omnipool contract do not follow the checks-effects-interactions pattern to prevent any reentrancy vulnerabilities in the functions.

Code Location:

For example, in the depositFor() function, the check is performed in the beginning, but the lastTransactionBlock state variable is only updated at the end:

Listing 21: src/pools/Omnipool.sol (Lines 227-229)

```
226     function depositFor(uint256 _amountIn, address _depositFor,
    ↳ uint256 _minLpReceived) public {
227         if (lastTransactionBlock[msg.sender] == block.number) {
228             revert CantDepositAndWithdrawSameBlock();
229         }
230
231         uint256 underlyingPrice = bptOracle.getUSDPrice(address(
    ↳ underlyingToken));
232
233         underlyingToken.approve(address(
    ↳ auraRewardPoolDepositWrapper), _amountIn);
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to update the user’s lastTransactionBlock just after the check.

Remediation Plan:

ACKNOWLEDGED: The Opal Finance team made a business decision to acknowledge this finding and not alter the contracts.

4.18 (HAL-18) LACK OF EMERGENCY STOP PATTERN IMPLEMENTATION - INFORMATIONAL (1.7)

Description:

It was identified that the OpalLpToken, LiquidityGauge, EscrowedToken contracts do not implement any kind of emergency stop pattern. Such a pattern allows the project team to pause crucial functionalities, while being in the state of emergency, e.g., being under adversary attack. The most prevalent application of the emergency stop pattern is the Pausable contract from the OpenZeppelin's library that.

In the case the emergency stop pattern is not used, critical functions cannot be temporarily disabled.

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to use the emergency stop pattern in the contracts.

Remediation Plan:

ACKNOWLEDGED: The Opal Finance team made a business decision to acknowledge this finding and not alter the contracts.

4.19 (HAL-19) LACK OF ZERO ADDRESS CHECKS - INFORMATIONAL (1.7)

Description:

It was identified that several parameters in the contracts lack zero address validation.

Code Location:

For example, the LiquidityGauge contract lacks zero address validation in its constructor:

Listing 22: src/tokenomics/LiquidityGauge.sol

```
    constructor(
        address _minter,
        address _minterEscrow,
        address _vlToken,
        address _registryContract
    ) {
        MINTER = _minter;
        MINTER_ESCROW = _minterEscrow;
        VL_TOKEN = _vlToken;
        registryContract = IRegistryContract(_registryContract);
        registryAccess = IRegistryAccess(registryContract.
↳ getContract(CONTRACT_REGISTRY_ACCESS));

        GAUGE_CONTROLLER = registryContract.getContract(
↳ CONTRACT_GAUGE_CONTROLLER);

        lpToken = address(0);
    }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to add zero address validation for the address parameters in constructors, initializers and setter functions.

Remediation Plan:

ACKNOWLEDGED: The Opal Finance team made a business decision to acknowledge this finding and not alter the contracts.

4.20 (HAL-20) REDUNDANT LOCK CHECK IN DEPOSIT - INFORMATIONAL (0.0)

Description:

It was identified that the lock check in the deposit() function of the Omnipool contract is redundant, as the check is executed again in the depositFor() function.

Code Location:

Redundant lock check in the deposit() function:

Listing 23: src/pools/Omnipool.sol

```
    function deposit(uint256 _amountIn, uint256 _minLpReceived)
 ↳ external {
        if (lastTransactionBlock[msg.sender] == block.number) {
            revert CantDepositAndWithdrawSameBlock();
        }
        depositFor(_amountIn, msg.sender, _minLpReceived);
    }
```

Listing 24: src/pools/Omnipool.sol

```
    function depositFor(uint256 _amountIn, address _depositFor,
 ↳ uint256 _minLpReceived) public {
        if (lastTransactionBlock[msg.sender] == block.number) {
            revert CantDepositAndWithdrawSameBlock();
        }

        uint256 underlyingPrice = bptOracle.getUSDPrice(address(
 ↳ underlyingToken));
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider removing the redundant lock check from the deposit() function.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commit 289ccaa.

4.21 (HAL-21) HARDCODED CONFIGURATION AND ADDRESSES - INFORMATIONAL (0.0)

Description:

It was identified that the contracts contain hardcoded configurations and addresses. Because the contracts are not upgradable, the Opal team will not be able to change them in the future. For example, if the address of the Opal treasury is compromised or changed, the team cannot update its address.

Code Location:

For example, the following hardcoded constants are used in different places in the protocol:

Listing 25: src/utils/constants.sol

```
    address constant EMERGENCY_MINTER = 0
↳ x1234567890123456789012345678901234567890;
    address constant WETH_ARBITRUM = 0
↳ x82aF49447D8a07e3bd95BD0d56f35241523fBab1;
    address constant ADMIN_ADDRESS = 0
↳ x1234567890123456789012345678901234567890;
    address constant INCENTIVES_MS = 0
↳ x1234567890123456789012345678901234561234;
    address constant BALANCER_VAULT = 0
↳ xBA12222222228d8Ba445958a75a0704d566BF2C8;
    address constant AURA_DEPOSIT_VAULT = 0
↳ x49e998899FF11598182918098588E8b90d7f60D3;
    address constant OPAL_TREASURY = 0
↳ x1234567890123456789012345678901234561234;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended that the smart contracts be reviewed and functions added to enable modifying settings that may need to be changed in the future.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue by implementing a registry.

Commit ID : a22b3e0205afb38438d0dcc203fda3eea71adf98

4.22 (HAL-22) UNUSED CODE - INFORMATIONAL (0.0)

Description:

Several unused state variables and functions were identified in the protocol:
- It was identified that the registryAccess and REWARD_TOKENS_LENGTH state variables and the onlyOpalTeam() modifier are not used in the RewardManager contract, as they have no function requiring authorization.
- It was identified that the usdcAddress state variable is not used in the BPTOracle contract.
- It was identified that the lastWeightUpdate state variable is never initialized, and therefore the getLastWeightUpdate() funciton cannat be used in the OmnipoolController contract.

The unutilized state variables and functions increase the gas cost and complexity of the contracts.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider reviewing the contracts and removing any unused state variables, functions, and libraries.

Remediation Plan:

SOLVED: The Opal Finance team solved the issue in commits b821c18, f9d59f7 and 51dbc2a.

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

src/pools/BPTOracle.sol

Slither results for BPTOracle.sol	
Finding	**Impact**
BPTOracle.BptPriceComposablePool(bytes32).i (src/pools/BPTOracle.sol#161) is a local variable never initialized	Medium
BPTOracle.BptPriceStablePool(bytes32).i (src/pools/BPTOracle.sol#75) is a local variable never initialized	Medium
BPTOracle.getUSDPrice(address) (src/pools/BPTOracle.sol#211-221) has external calls inside a loop: priceFeed = IPriceFeed(priceFeedAddress).getPriceFeedFromAsset(token) (src/pools/BPTOracle.sol#214)	Low
BPTOracle.getUSDPrice(address) (src/pools/BPTOracle.sol#211-221) has external calls inside a loop: (priceInUSDInt) = priceFeed.latestRoundData() (src/pools/BPTOracle.sol#216)	Low
BPTOracle.BptPriceComposablePool(bytes32) (src/pools/BPTOracle.sol#151-187) has external calls inside a loop: poolRate = IRateProvider(pool).getRate() (src/pools/BPTOracle.sol#173)	Low

Finding	Impact
BPTOracle.getUSDPrice(address) (src/pools/BPTOracle.sol#211-221) has external calls inside a loop: decimals = ERC20(token).decimals() (src/pools/BPTOracle.sol#212)	Low
End of table for BPTOracle.sol	

src/pools/Omnipool.sol

Slither results for Omnipool.sol	
Finding	**Impact**
Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381) ignores return value by underlyingToken.transfer(OPAL_TREASURY,underlyingFees) (src/pools/Omnipool.sol#379)	High
Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) ignores return value by underlying Token.transferFrom(msg.sender,address(this),_amountIn) (src/pools/Omnipool.sol#244)	High
Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381) ignores return value by underlyingToken.transfer(msg.sender,underlyingWithdrawn_) (src/pools/Omnipool.sol#380)	High
Omnipool._MIN_DEPEG_THRESHOLD (src/pools/Omnipool.sol#94) is never initialized. It is used in: - Omnipool.updateDepegThreshold(uint256) (src/pools/Omnipool.sol#631-640)	High
Omnipool._MAX_DEPEG_THRESHOLD (src/pools/Omnipool.sol#95) is never initialized. It is used in: - Omnipool.updateDepegThreshold(uint256) (src/pools/Omnipool.sol#631-640)	High
Omnipool.lpTokenPerPool (src/pools/Omnipool.sol#102) is never initialized. It is used in: - Omnipool.handleDepeggedPool(address) (src/pools/Omnipool.sol#915-941)	High
Omnipool._exchangeRate(uint256) (src/pools/Omnipool.sol#688-693) uses a dangerous strict equality: - lpSupply == 0 || totalUnderlying_ == 0 (src/pools/Omnipool.sol#690)	Medium

Finding	Impact
Omnipool._isBalanced(uint256[],uint256) (src/pools/Omnipool.sol#1110-1131) uses a dangerous strict equality: - totalAllocated_ == 0 (src/pools/Omnipool.sol#1115)	Medium
Omnipool._getUnderlyingCurrentWeight(uint256) (src/pools/Omnipool.sol#701-705) uses a dangerous strict equality: - poolTvl == 0 || totalTvl == 0 (src/pools/Omnipool.sol#704)	Medium
Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) uses a dangerous strict equality: - lastTransactionBlock[msg.sender] == block.number (src/pools/Omnipool.sol#227)	Medium
Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381) uses a dangerous strict equality: - lastTransactionBlock[msg.sender] == block.number (src/pools/Omnipool.sol#347)	Medium
Omnipool.deposit(uint256,uint256) (src/pools/Omnipool.sol#334-339) uses a dangerous strict equality: - lastTransactionBlock[msg.sender] == block.number (src/pools/Omnipool.sol#335)	Medium
Contract locking ether found: Contract Omnipool (src/pools/Omnipool.sol#49-1225) has payable functions: - Omnipool.constructor(address,address,address,address,string,string) (src/pools/Omnipool.sol#146-169) But does not have a function to withdraw the ether	Medium

Finding	Impact
Reentrancy in Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268): External calls: - underlyingToken.approve(address(auraRewardPoolDepositWrapper),_amountIn) (src/pools/Omnipool.sol#233) - underlyingToken.transferFrom(msg.sender,address(this),_amountIn) (src/pools/Omnipool.sol#244) - _depositToAura(beforeAllocatedBalance,beforeAllocatedPerPool,_amountIn) (src/pools/Omnipool.sol#246) - auraRewardPoolDepositWrapper.depositSingle(address(_pool.poolAddress),underlyingToken,_underlyingAmountIn,_pool.poolId,joinRequest) (src/pools/Omnipool.sol#451-457) - lpToken.mint(_depositFor,lpReceived) (src/pools/Omnipool.sol#256) - _handleRebalancingRewards(msg.sender,beforeTotalUnderlying,afterTotalUnderlying,beforeAllocatedPerPool,afterAllocatedPerPool) (src/pools/Omnipool.sol#260-266) - controller.handleRebalancingRewards(account,deviationBefore,deviationAfter) (src/pools/Omnipool.sol#1096) State variables written after the call(s): - lastTransactionBlock[msg.sender] = block.number (src/pools/Omnipool.sol#267) Omnipool.lastTransactionBlock (src/pools/Omnipool.sol#103) can be used in cross function reentrancies: - Omnipool.deposit(uint256,uint256) (src/pools/Omnipool.sol#334-339) - Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) - Omnipool.lastTransactionBlock (src/pools/Omnipool.sol#103) - Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381) - _handleRebalancingRewards(msg.sender,beforeTotalUnderlying,afterTotalUnderlying,beforeAllocatedPerPool,afterAllocatedPerPool) (src/pools/Omnipool.sol#260-266) - rebalancingRewardActive = false (src/pools/Omnipool.sol#1099)Omnipool.rebalancingRewardActive (src/pools/Omnipool.sol#92) can be used in cross function reentrancies: - Omnipool._getMaxDeviation() (src/pools/Omnipool.sol#1152-1154) - Omnipool._handleRebalancingRewards(address,uint256,uint256,uint256[],uint256[]) (src/pools/Omnipool.sol#1081-1101) - Omnipool.handleDepeggedPool(address) (src/pools/Omnipool.sol#915-941) - Omnipool.rebalancingRewardActive (src/pools/Omnipool.sol#92) - Omnipool.updateWeight(address,uint256) (src/pools/Omnipool.sol#1061-1079) - Omnipool.updateWeights(IOmnipoolController.WeightUpdate[])	Medium

Finding	Impact
Reentrancy in Omnipool._handleRebalancingRewards(address,uint256,uint256,uint256[],uint256[]) (src/pools/Omnipool.sol#1081-1101): External calls: - controller.handleRebalancingRewards(account,deviationBefore,deviationAfter) (src/pools/Omnipool.sol#1096) State variables written after the call(s): - rebalancingRewardActive = false (src/pools/Omnipool.sol#1099)Omnipool.rebalancingRewardActive (src/pools/Omnipool.sol#92) can be used in cross function reentrancies: - Omnipool._getMaxDeviation() (src/pools/Omnipool.sol#1152-1154) - Omnipool._handleRebalancingRewards(address,uint256,uint256,uint256[],uint256[]) (src/pools/Omnipool.sol#1081-1101) - Omnipool.handleDepeggedPool(address) (src/pools/Omnipool.sol#915-941) - Omnipool.rebalancingRewardActive (src/pools/Omnipool.sol#92) - Omnipool.updateWeight(address,uint256) (src/pools/Omnipool.sol#1061-1079) - Omnipool.updateWeights(IOmnipoolController.WeightUpdate[]) (src/pools/Omnipool.sol#1020-1053)	Medium
Reentrancy in Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381): External calls: - _withdrawFromAura(allocatedUnderlying_,allocatedPerPool,underlyingToWithdraw_) (src/pools/Omnipool.sol#364) - auraPool.withdrawAndUnwrap(_bptAmountOut,true) (src/pools/Omnipool.sol#540) - balancerVault.exitPool(_pool.poolId,address(this),address(address(this)),exitRequest) (src/pools/Omnipool.sol#557) State variables written after the call(s): - lastTransactionBlock[msg.sender] = block.number (src/pools/Omnipool.sol#372) Omnipool.lastTransactionBlock (src/pools/Omnipool.sol#103) can be used in cross function reentrancies: - Omnipool.deposit(uint256,uint256) (src/pools/Omnipool.sol#334-339) - Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) - Omnipool.lastTransactionBlock (src/pools/Omnipool.sol#103) - Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381)	Medium

Finding	Impact
Omnipool.updateWeights(IOmnipoolController.WeightUpdate[]).i (src/pools/Omnipool.sol#1027) is a local variable never initialized	Medium
Omnipool._getDepositPool(uint256,uint256[]).i (src/pools/Omnipool.sol#504) is a local variable never initialized	Medium
Omnipool._computeTotalDeviation(uint256,uint256[]).i (src/pools/Omnipool.sol#991) is a local variable never initialized	Medium
Omnipool._setWeightToZero(address).i (src/pools/Omnipool.sol#953) is a local variable never initialized	Medium
Omnipool._isBalanced(uint256[],uint256).i (src/pools/Omnipool.sol#1117) is a local variable never initialized	Medium
Omnipool._getWithdrawPool(uint256,uint256[]).i (src/pools/Omnipool.sol#592) is a local variable never initialized	Medium
Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) ignores return value by lpToken.mint(_depositFor,lpReceived) (src/pools/Omnipool.sol#256)	Medium
Omnipool.setExtraRewardPool(address,bytes32) (src/pools/Omnipool.sol#772-779) ignores return value by IERC20(_token).approve(address(balancerVault),0) (src/pools/Omnipool.sol#774)	Medium
Omnipool.approve(address,address,uint256) (src/pools/Omnipool.sol#763-767) ignores return value by erc20.approve(addr,amount) (src/pools/Omnipool.sol#766)	Medium
Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) ignores return value by underlying Token.approve(address(auraRewardPoolDepositWrapper),_amountIn) (src/pools/Omnipool.sol#233)	Medium
Omnipool._withdrawFromAuraPool(IOmnipool.UnderlyingPool,uint256) (src/pools/Omnipool.sol#526-558) ignores return value by auraPool.withdrawAndUnwrap(_bptAmountOut,true) (src/pools/Omnipool.sol#540)	Medium
Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381) ignores return value by lpToken.burn(msg.sender,_amountOut) (src/pools/Omnipool.sol#373)	Medium
Omnipool.setExtraRewardPool(address,bytes32) (src/pools/Omnipool.sol#772-779) ignores return value by IERC20(_to ken).approve(address(balancerVault),type()(uint256).max) (src/pools/Omnipool.sol#775)	Medium

Finding	Impact
Omnipool.swapForGem(address,uint256) (src/pools/Omnipool.sol#793-853) ignores return value by balancerVault.batchSwap(IBalancerVault.SwapKind.GIVEN_IN,batchSwapSteps,assets,fundManagement,limits,deadline) (src/pools/Omnipool.sol#843-850)	Medium
Omnipool.swapForGem(address,uint256) (src/pools/Omnipool.sol#793-853) ignores return value by erc20Token.approve(address(balancerVault),_amountIn) (src/pools/Omnipool.sol#800)	Medium
Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268) should emit an event for: - totalDeposited += _amountIn (src/pools/Omnipool.sol#258)	Low
Omnipool.setRewardManager(address)._rewardManager (src/pools/Omnipool.sol#206) lacks a zero-check on : - rewardManager = _rewardManager (src/pools/Omnipool.sol#207)	Low
Omnipool.computeBptValution(uint256) (src/pools/Omnipool.sol#215-218) has external calls inside a loop: bptOracle.getPoolValuation(pool.poolId,pool.poolType) (src/pools/Omnipool.sol#217)	Low
Omnipool.getPoolTvl(uint256) (src/pools/Omnipool.sol#178-184) has external calls inside a loop: bptBalance = IBalancerPool(pool.poolAddress).balanceOf(address(this)) (src/pools/Omnipool.sol#181)	Low
Omnipool.getUserDeposit(address,uint256) (src/pools/Omnipool.sol#408-413) has external calls inside a loop: bptBalance = IBalancerPool(pool.poolAddress).balanceOf(user) (src/pools/Omnipool.sol#410)	Low

Finding	Impact
Reentrancy in Omnipool.depositFor(uint256,address,uint256) (src/pools/Omnipool.sol#226-268): External calls: - underlyingToken.approve(address(auraRewardPoolDepositWrapper),_amountIn) (src/pools/Omnipool.sol#233) - underlyingToken.transferFrom(msg.sender,address(this),_amountIn) (src/pools/Omnipool.sol#244) - _depositToAura(beforeAllocatedBalance,beforeAllocatedPerPool,_amountIn) (src/pools/Omnipool.sol#246) - auraRewardPoolDepositWrapper.depositSingle(address(_pool.poolAddress),underlyingToken,_underlyingAmountIn,_pool.poolId,joinRequest) (src/pools/Omnipool.sol#451-457) - lpToken.mint(_depositFor,lpReceived) (src/pools/Omnipool.sol#256) State variables written after the call(s): - totalDeposited += _amountIn (src/pools/Omnipool.sol#258)	Low
Reentrancy in Omnipool.setExtraRewardPool(address,bytes32) (src/pools/Omnipool.sol#772-779): External calls: - IERC20(_token).approve(address(balancerVault),0) (src/pools/Omnipool.sol#774) - IERC20(_token).approve(address(balancerVault),type()(uint256).max) (src/pools/Omnipool.sol#775) State variables written after the call(s): - extraRewardPools[_token] = _poolId (src/pools/Omnipool.sol#777)	Low
Reentrancy in Omnipool.withdraw(uint256,uint256) (src/pools/Omnipool.sol#346-381): External calls: - _withdrawFromAura(allocatedUnderlying_,allocatedPerPool,underlyingToWithdraw_) (src/pools/Omnipool.sol#364) - auraPool.withdrawAndUnwrap(_bptAmountOut,true) (src/pools/Omnipool.sol#540) - balancerVault.exitPool(_pool.poolId,address(this),address(address(this)),exitRequest) (src/pools/Omnipool.sol#557) - lpToken.burn(msg.sender,_amountOut) (src/pools/Omnipool.sol#373) State variables written after the call(s): - totalDeposited -= underlyingWithdrawn_ (src/pools/Omnipool.sol#374)	Low

Finding	Impact
Reentrancy in Omnipool.setExtraRewardPool(address,bytes32) (src/pools/Omnipool.sol#772-779): External calls: - IERC20(_token).approve(address(balancerVault),0) (src/pools/Omnipool.sol#774) - IERC20(_token).approve(address(balancerVault),type()(uint256).max) (src/pools/Omnipool.sol#775) Event emitted after the call(s): - ExtraRewardPoolIdUpdated(_token,_poolId) (src/pools/Omnipool.sol#778)	Low
End of table for Omnipool.sol	

src/pools/OmnipoolController.sol

Slither results for OmnipoolController.sol	
Finding	**Impact**
OmnipoolController.lastWeightUpdate (src/pools/OmnipoolController.sol#47) is never initialized. It is used in: - OmnipoolController.getLastWeightUpdate(address) (src/pools/OmnipoolController.sol#334-336)	High
OmnipoolController.computePoolWeight(address).poolUSDValue (src/pools/OmnipoolController.sol#313) is a local variable never initialized	Medium
OmnipoolController.updateWeights(address,IOmnipoolController.WeightUpdate[]).i (src/pools/OmnipoolController.sol#206) is a local variable never initialized	Medium
OmnipoolController.computePoolWeights().i_scope_1 (src/pools/OmnipoolController.sol#296) is a local variable never initialized	Medium
OmnipoolController.handleRebalancingRewards(address,uint256,uint256).i (src/pools/OmnipoolController.sol#257) is a local variable never initialized	Medium
OmnipoolController.computePoolWeights().i_scope_0 (src/pools/OmnipoolController.sol#292) is a local variable never initialized	Medium
OmnipoolController.computePoolWeight(address).i (src/pools/OmnipoolController.sol#314) is a local variable never initialized	Medium

Finding	Impact
OmnipoolController.updateAllWeights(IOmnipoolController.WeightUpdate[]).i (src/pools/OmnipoolController.sol#223) is a local variable never initialized	Medium
OmnipoolController.computePoolWeights().i (src/pools/OmnipoolController.sol#277) is a local variable never initialized	Medium
OmnipoolController.handleRebalancingRewards(address,uint256,uint256) (src/pools/OmnipoolController.sol#248-266) ignores return value by IRebalancingRewardsHandler(handler).handleRebalancingRewards(IOmnipool(msg.sender),account,deviationBefore,deviationAfter) (src/pools/OmnipoolController.sol#259-261)	Medium
OmnipoolController.computePoolWeights() (src/pools/OmnipoolController.sol#270-300) has external calls inside a loop: price = oracle.getUSDPrice(address(underlying)) (src/pools/OmnipoolController.sol#281)	Low
OmnipoolController.handleRebalancingRewards(address,uint256,uint256) (src/pools/OmnipoolController.sol#248-266) has external calls inside a loop: IRebalancingRewardsHandler(handler).handleRebalancingRewards(IOmnipool(msg.sender),account,deviationBefore,deviationAfter) (src/pools/OmnipoolController.sol#259-261)	Low
OmnipoolController.computePoolWeight(address) (src/pools/OmnipoolController.sol#304-332) has external calls inside a loop: underlying = currentPool.getUnderlyingToken() (src/pools/OmnipoolController.sol#317)	Low
OmnipoolController.computePoolWeight(address) (src/pools/OmnipoolController.sol#304-332) has external calls inside a loop: usdValue = currentPool.getTotalUnderlying().convertScale(underlying.decimals(),18).mulDown(price) (src/pools/OmnipoolController.sol#319-321)	Low
OmnipoolController.updateWeights(address,IOmnipoolController.WeightUpdate[]) (src/pools/OmnipoolController.sol#201-212) has external calls inside a loop: IOmnipool(omniPool).updateWeights(weights) (src/pools/OmnipoolController.sol#207)	Low
OmnipoolController.onlyOpalTeam() (src/pools/OmnipoolController.sol#73-78) has external calls inside a loop: ! registryAccess.checkRole(ROLE_OPAL_TEAM,msg.sender) (src/pools/OmnipoolController.sol#74)	Low

Finding	Impact
OmnipoolController.computePoolWeights() (src/pools/OmnipoolController.sol#270-300) has external calls inside a loop: poolUSDValue = pool.getTotalUnderlying().convertScale(underlying.decimals(),18).mulDown(price) (src/pools/OmnipoolController.sol#282-283)	Low
OmnipoolController.computePoolWeight(address) (src/pools/OmnipoolController.sol#304-332) has external calls inside a loop: price = oracle.getUSDPrice(address(underlying)) (src/pools/OmnipoolController.sol#318)	Low
OmnipoolController.computePoolWeights() (src/pools/OmnipoolController.sol#270-300) has external calls inside a loop: underlying = pool.getUnderlyingToken() (src/pools/OmnipoolController.sol#280)	Low
End of table for OmnipoolController.sol	

src/pools/OpalLpToken.sol

Slither results for OpalLpToken.sol	
Finding	Impact
End of table for OpalLpToken.sol	

src/tokenomics/EscrowedToken.sol

Slither results for EscrowedToken.sol	
Finding	Impact
EscrowedToken.getVestingClaimValue(address,uint256) (src/tokenomics/EscrowedToken.sol#176-199) performs a multiplication on the result of a division: - claimAmount = (userVesting.amount * (SCALED_ONE + (ratePerToken - userVesting.ratePerToken))) / SCALED_ONE (src/tokenomics/EscrowedToken.sol#190-192) - removedAmount = (claimAmount * remainingTime) / vestingDuration (src/tokenomics/EscrowedToken.sol#195)	Medium

Finding	Impact
EscrowedToken._claim(address,uint256) (src/tokenomics/EscrowedToken.sol#283-308) performs a multiplication on the result of a division: - removedAmount = (claimAmount * remainingTime) / vestingDuration (src/tokenomics/EscrowedToken.sol#295) - ratePerToken += (SCALED_ONE * removedAmount) / totalVesting (src/tokenomics/EscrowedToken.sol#304)	Medium
EscrowedToken._claim(address,uint256) (src/tokenomics/EscrowedToken.sol#283-308) performs a multiplication on the result of a division: - claimAmount = (userVesting.amount * (SCALED_ONE + (ratePerToken - userVesting.ratePerToken))) / SCALED_ONE (src/tokenomics/EscrowedToken.sol#292-294) - removedAmount = (claimAmount * remainingTime) / vestingDuration (src/tokenomics/EscrowedToken.sol#295)	Medium
Reentrancy in EscrowedToken._claim(address,uint256) (src/tokenomics/EscrowedToken.sol#283-308): External calls: - token.safeTransfer(account,claimAmount) (src/tokenomics/EscrowedToken.sol#302) State variables written after the call(s): - ratePerToken += (SCALED_ONE * removedAmount) / totalVesting (src/tokenomics/EscrowedToken.sol#304)EscrowedToken.ratePerToken (src/tokenomics/EscrowedToken.sol#47) can be used in cross function reentrancies: - EscrowedToken.getVestingClaimValue(address,uint256) (src/tokenomics/EscrowedToken.sol#176-199) - EscrowedToken.ratePerToken (src/tokenomics/EscrowedToken.sol#47)	Medium
EscrowedToken.claimMultiple(uint256[]).i (src/tokenomics/EscrowedToken.sol#254) is a local variable never initialized	Medium
EscrowedToken.getUserActiveVestings(address).activeCount (src/tokenomics/EscrowedToken.sol#141) is a local variable never initialized	Medium
EscrowedToken.claimAll().i (src/tokenomics/EscrowedToken.sol#267) is a local variable never initialized	Medium
EscrowedToken.getUserActiveVestings(address).i (src/tokenomics/EscrowedToken.sol#143) is a local variable never initialized	Medium

Finding	Impact
EscrowedToken.getUserActiveVestings(address).index (src/tokenomics/EscrowedToken.sol#152) is a local variable never initialized	Medium
EscrowedToken._claim(address,uint256).remainingTime (src/tokenomics/EscrowedToken.sol#288) is a local variable never initialized	Medium
EscrowedToken.getVestingClaimValue(address,uint256).remainingTime (src/tokenomics/EscrowedToken.sol#185) is a local variable never initialized	Medium
EscrowedToken.getUserActiveVestings(address).i_scope_0 (src/tokenomics/EscrowedToken.sol#154) is a local variable never initialized	Medium
EscrowedToken._claim(address,uint256) (src/tokenomics/EscrowedToken.sol#283-308) uses timestamp for comparisons Dangerous comparisons: - block.timestamp < userVesting.end (src/tokenomics/EscrowedToken.sol#289)	Low
EscrowedToken.getVestingClaimValue(address,uint256) (src/tokenomics/EscrowedToken.sol#176-199) uses timestamp for comparisons Dangerous comparisons: - block.timestamp < userVesting.end (src/tokenomics/EscrowedToken.sol#186)	Low
EscrowedToken.mint(uint256,address,uint256) (src/tokenomics/EscrowedToken.sol#209-237) uses timestamp for comparisons Dangerous comparisons: - startTimestamp < block.timestamp (src/tokenomics/EscrowedToken.sol#216)	Low
End of table for EscrowedToken.sol	

src/tokenomics/GaugeController.sol

Slither results for GaugeController.sol	
Finding	**Impact**

Finding	Impact
GaugeController.addGauge(address,int128,uint256) (src/tokenomics/GaugeController.sol#577-611) performs a multiplication on the result of a division: - nextTimestamp = ((block.timestamp + WEEK) / WEEK) * WEEK (src/tokenomics/GaugeController.sol#590)	Medium
GaugeController._gaugeRelativeWeight(address,uint256) (src/tokenomics/GaugeController.sol#394-408) performs a multiplication on the result of a division: - timestamp = (timestamp / WEEK) * WEEK (src/tokenomics/GaugeController.sol#399)	Medium
GaugeController._changeTypeWeight(int128,uint256) (src/tokenomics/GaugeController.sol#415-428) performs a multiplication on the result of a division: - nextTimestamp = ((block.timestamp + WEEK) / WEEK) * WEEK (src/tokenomics/GaugeController.sol#419)	Medium
GaugeController._voteForGaugeweight(address,address,uint256) (src/tokenomics/GaugeController.sol#476-567) performs a multiplication on the result of a division: - vars.nextTimestamp = ((block.timestamp + WEEK) / WEEK) * WEEK (src/tokenomics/GaugeController.sol#482)	Medium
GaugeController._changeGaugeWeight(address,uint256) (src/tokenomics/GaugeController.sol#435-457) performs a multiplication on the result of a division: - nextTimestamp = ((block.timestamp + WEEK) / WEEK) * WEEK (src/tokenomics/GaugeController.sol#443)	Medium
GaugeController.constructor(address,address,address) (src/tokenomics/GaugeController.sol#101-110) performs a multiplication on the result of a division: - lastUpdate = (block.timestamp / WEEK) * WEEK (src/tokenomics/GaugeController.sol#109)	Medium
GaugeController.addGauge(address,int128,uint256) (src/tokenomics/GaugeController.sol#577-611) uses a dangerous strict equality: - lastTypeUpdate[gaugeType] == 0 (src/tokenomics/GaugeController.sol#605)	Medium

Finding	Impact
GaugeController._getSum(int128) (src/tokenomics/GaugeController.sol#271-300) uses a dangerous strict equality: - timestamp == 0 (src/tokenomics/GaugeController.sol#273)	Medium
GaugeController._getWeight(address) (src/tokenomics/GaugeController.sol#357-386) uses a dangerous strict equality: - timestamp == 0 (src/tokenomics/GaugeController.sol#359)	Medium
GaugeController._getTotal() (src/tokenomics/GaugeController.sol#306-350) uses a dangerous strict equality: - timestamp == 0 (src/tokenomics/GaugeController.sol#308)	Medium
GaugeController._getTypeWeight(int128) (src/tokenomics/GaugeController.sol#243-264) uses a dangerous strict equality: - timestamp == 0 (src/tokenomics/GaugeController.sol#245)	Medium
GaugeController._voteForGaugeweight(address,address,uint256) (src/tokenomics/GaugeController.sol#476-567) contains a tautology or contradiction: - vars.powerUsed > 10_000 || vars.powerUsed < 0 (src/tokenomics/GaugeController.sol#518)	Medium
GaugeController._voteForGaugeweight(address,address,uint256).j (src/tokenomics/GaugeController.sol#529) is a local variable never initialized	Medium
GaugeController.voteForManyGaugeWeights(address[],uint256[]).i (src/tokenomics/GaugeController.sol#227) is a local variable never initialized	Medium
GaugeController._getTotal().j (src/tokenomics/GaugeController.sol#322) is a local variable never initialized	Medium
GaugeController._getWeight(address).i (src/tokenomics/GaugeController.sol#362) is a local variable never initialized	Medium
GaugeController._voteForGaugeweight(address,address,uint256).l (src/tokenomics/GaugeController.sol#559) is a local variable never initialized	Medium

Finding	Impact
GaugeController._getTotal().i (src/tokenomics/GaugeController.sol#311) is a local variable never initialized	Medium
GaugeController._getTypeWeight(int128).i (src/tokenomics/GaugeController.sol#248) is a local variable never initialized	Medium
GaugeController._voteForGaugeweight(address,address,uint256).k (src/tokenomics/GaugeController.sol#545) is a local variable never initialized	Medium
GaugeController._getTotal().k (src/tokenomics/GaugeController.sol#327) is a local variable never initialized	Medium
GaugeController._voteForGaugeweight(address,address,uint256).vars (src/tokenomics/GaugeController.sol#477) is a local variable never initialized	Medium
GaugeController._getSum(int128).i (src/tokenomics/GaugeController.sol#276) is a local variable never initialized	Medium
GaugeController.constructor(address,address,address)._voteLocker (src/tokenomics/GaugeController.sol#101) lacks a zero-check on : - voteLocker = _voteLocker (src/tokenomics/GaugeController.sol#105)	Low
GaugeController.constructor(address,address,address)._token (src/tokenomics/GaugeController.sol#101) lacks a zero-check on : - token = _token (src/tokenomics/GaugeController.sol#104)	Low
GaugeController._voteForGaugeweight(address,address,uint256) (src/tokenomics/GaugeController.sol#476-567) has external calls inside a loop: (locks) = IVoteLocker(voteLocker).lockedBalances(msg.sender) (src/tokenomics/GaugeController.sol#478-479)	Low
GaugeController._getTotal() (src/tokenomics/GaugeController.sol#306-350) uses timestamp for comparisons Dangerous comparisons: - timestamp == 0 (src/tokenomics/GaugeController.sol#308) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#323) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#340)	Low

Finding	Impact
GaugeController._getWeight(address) (src/tokenomics/GaugeController.sol#357-386) uses timestamp for comparisons Dangerous comparisons: - timestamp == 0 (src/tokenomics/GaugeController.sol#359) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#363) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#376)	Low
GaugeController.addGauge(address,int128,uint256) (src/tokenomics/GaugeController.sol#577-611) uses timestamp for comparisons Dangerous comparisons: - lastTypeUpdate[gaugeType] == 0 (src/tokenomics/GaugeController.sol#605)	Low
GaugeController._voteForGaugeweight(address,address,uint256) (src/tokenomics/GaugeController.sol#476-567) uses timestamp for comparisons Dangerous comparisons: - locks[vars.len - 1].unlockTime < vars.nextTimestamp (src/tokenomics/GaugeController.sol#483) - block.timestamp < lastUserVote[user][gauge] + WEIGHT_VOTE_DELAY (src/tokenomics/GaugeController.sol#485) - currentLock.unlockTime > vars.nextTimestamp (src/tokenomics/GaugeController.sol#494) - i > 0 (src/tokenomics/GaugeController.sol#503) - vars.gaugeType < 0 (src/tokenomics/GaugeController.sol#512) - vars.powerUsed > 10_000 || vars.powerUsed < 0 (src/tokenomics/GaugeController.sol#518) - j < vars.oldUnlocksLen (src/tokenomics/GaugeController.sol#529) - oldUnlocks[j].unlockTime <= block.timestamp (src/tokenomics/GaugeController.sol#531) - k < vars.len (src/tokenomics/GaugeController.sol#545) - unlocks[k].unlockTime <= block.timestamp (src/tokenomics/GaugeController.sol#547) - l < vars.len (src/tokenomics/GaugeController.sol#559) - unlocks[l].unlockTime <= block.timestamp (src/tokenomics/GaugeController.sol#561)	Low

Finding	Impact
GaugeController._getTypeWeight(int128) (src/tokenomics/GaugeController.sol#243-264) uses timestamp for comparisons Dangerous comparisons: - timestamp == 0 (src/tokenomics/GaugeController.sol#245) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#249) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#254)	Low
GaugeController._getSum(int128) (src/tokenomics/GaugeController.sol#271-300) uses timestamp for comparisons Dangerous comparisons: - timestamp == 0 (src/tokenomics/GaugeController.sol#273) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#277) - timestamp > block.timestamp (src/tokenomics/GaugeController.sol#290)	Low
End of table for GaugeController.sol	

src/tokenomics/MinterEscrow.sol

Slither results for MinterEscrow.sol	
Finding	**Impact**
MinterEscrow._mintMultipleFor(address[],address).totalMintAmount (src/tokenomics/MinterEscrow.sol#224) is a local variable never initialized	Medium
MinterEscrow._mintMultipleFor(address[],address).i (src/tokenomics/MinterEscrow.sol#227) is a local variable never initialized	Medium
MinterEscrow.updateApprove(uint256) (src/tokenomics/MinterEscrow.sol#111-113) ignores return value by IERC20(token).approve(escrow,_approve) (src/tokenomics/MinterEscrow.sol#112)	Medium
MinterEscrow._prepareGaugeMint(address,address) (src/tokenomics/MinterEscrow.sol#203-216) ignores return value by ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/MinterEscrow.sol#206)	Medium

Finding	Impact
MinterEscrow.constructor(address,address,address,address) (src/tokenomics/MinterEscrow.sol#56-65) ignores return value by IERC20(token).approve(escrow,type()(uint256).max) (src/tokenomics/MinterEscrow.sol#62)	Medium
MinterEscrow.constructor(address,address,address,address).controller_ (src/tokenomics/MinterEscrow.sol#56) lacks a zero-check on : - controller = controller_ (src/tokenomics/MinterEscrow.sol#59)	Low
MinterEscrow.constructor(address,address,address,address)._escrow (src/tokenomics/MinterEscrow.sol#56) lacks a zero-check on : - escrow = _escrow (src/tokenomics/MinterEscrow.sol#58)	Low
MinterEscrow.constructor(address,address,address,address).token_ (src/tokenomics/MinterEscrow.sol#56) lacks a zero-check on : - token = token_ (src/tokenomics/MinterEscrow.sol#57)	Low
Reentrancy in MinterEscrow._prepareGaugeMint(address,address) (src/tokenomics/MinterEscrow.sol#203-216): External calls: - ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/MinterEscrow.sol#206) State variables written after the call(s): - minted[account][gauge] = totalMint (src/tokenomics/MinterEscrow.sol#212) - mintedSupply = _newMintedSupply (src/tokenomics/MinterEscrow.sol#213)	Low
Reentrancy in MinterEscrow._mintMultipleFor(address[],address) (src/tokenomics/MinterEscrow.sol#223-241): External calls: - toMintAmount = _prepareGaugeMint(gauges[i],account) (src/tokenomics/MinterEscrow.sol#228) - ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/MinterEscrow.sol#206) Event emitted after the call(s): - Minted(account,gauges[i],toMintAmount) (src/tokenomics/MinterEscrow.sol#232)	Low

Finding	Impact
Reentrancy in MinterEscrow._mintFor(address,address) (src/tokenomics/MinterEscrow.sol#189-196): External calls: - toMintAmount = _prepareGaugeMint(gauge,account) (src/tokenomics/MinterEscrow.sol#190) - ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/MinterEscrow.sol#206) - EscrowedToken(escrow).mint(toMintAmount,account,block.timestamp) (src/tokenomics/MinterEscrow.sol#193) Event emitted after the call(s): - Minted(account,gauge,toMintAmount) (src/tokenomics/MinterEscrow.sol#194)	Low
MinterEscrow._mintableInTimeframe(uint256,uint256) (src/tokenomics/MinterEscrow.sol#176-182) uses timestamp for comparisons Dangerous comparisons: - start > end (src/tokenomics/MinterEscrow.sol#177) - start < startDistribution (src/tokenomics/MinterEscrow.sol#179)	Low
MinterEscrow.availableSupply() (src/tokenomics/MinterEscrow.sol#73-76) uses timestamp for comparisons Dangerous comparisons: - block.timestamp < startDistribution (src/tokenomics/MinterEscrow.sol#74)	Low
MinterEscrow.rate() (src/tokenomics/MinterEscrow.sol#92-95) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= startDistribution + RATE_END_TIMESTAMP (src/tokenomics/MinterEscrow.sol#93)	Low
MinterEscrow._prepareGaugeMint(address,address) (src/tokenomics/MinterEscrow.sol#203-216) uses timestamp for comparisons Dangerous comparisons: - _newMintedSupply > _availableSupply() (src/tokenomics/MinterEscrow.sol#210)	Low
End of table for MinterEscrow.sol	

src/tokenomics/VoteLocker.sol

Slither results for VoteLocker.sol	
Finding	**Impact**

Finding	Impact
VoteLocker._lock(address,uint256) (src/tokenomics/VoteLocker.sol#362-405) performs a multiplication on the result of a division: - currentEpoch = block.timestamp.div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#382)	Medium
VoteLocker._processExpiredLocks(address,bool,address,uint256) (src/tokenomics/VoteLocker.sol#544-639) performs a multiplication on the result of a division: - currentEpoch_scope_0 = block.timestamp.sub(_checkDelay).div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#595-596)	Medium
VoteLocker._checkpointEpoch() (src/tokenomics/VoteLocker.sol#481-493) performs a multiplication on the result of a division: - currentEpoch = block.timestamp.div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#482)	Medium
VoteLocker.constructor(string,string,address,address) (src/tokenomics/VoteLocker.sol#184-201) performs a multiplication on the result of a division: - currentEpoch = block.timestamp.div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#199)	Medium
VoteLocker._processExpiredLocks(address,bool,address,uint256) (src/tokenomics/VoteLocker.sol#544-639) performs a multiplication on the result of a division: - currentEpoch = block.timestamp.sub(_checkDelay).div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#574-575)	Medium
VoteLocker.getPastVotes(address,uint256) (src/tokenomics/VoteLocker.sol#789-801) performs a multiplication on the result of a division: - epoch = timestamp.div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#791)	Medium

Finding	Impact
VoteLocker.delegate(address) (src/tokenomics/VoteLocker.sol#650-693) performs a multiplication on the result of a division: - upcomingEpoch = block.timestamp.add(rewardsDuration).div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#666-667)	Medium
VoteLocker._checkpointDelegate(address,uint256,uint256) (src/tokenomics/VoteLocker.sol#695-751) performs a multiplication on the result of a division: - upcomingEpoch = block.timestamp.add(rewardsDuration).div(rewardsDuration).mul(rewardsDuration) (src/tokenomics/VoteLocker.sol#702-703)	Medium
VoteLocker.totalSupplyAtEpoch(uint256) (src/tokenomics/VoteLocker.sol#941-963) uses a dangerous strict equality: - e.date == epochStart (src/tokenomics/VoteLocker.sol#952)	Medium
VoteLocker._checkpointDelegate(address,uint256,uint256) (src/tokenomics/VoteLocker.sol#695-751) uses a dangerous strict equality: - prevCkpt.epochStart == upcomingEpoch (src/tokenomics/VoteLocker.sol#708)	Medium
Reentrancy in VoteLocker.getReward(address,bool[]) (src/tokenomics/VoteLocker.sol#448-468): External calls: - IERC20(_rewardsToken).safeTransfer(_account,reward) (src/tokenomics/VoteLocker.sol#461) State variables written after the call(s): - userData[_account][_rewardsToken].rewards = 0 (src/tokenomics/VoteLocker.sol#460) VoteLocker.userData (src/tokenomics/VoteLocker.sol#82) can be used in cross function reentrancies: - VoteLocker._earned(address,address,uint256) (src/tokenomics/VoteLocker.sol#1048-1056) - VoteLocker.getReward(address,bool) (src/tokenomics/VoteLocker.sol#422-441) - VoteLocker.updateReward(address) (src/tokenomics/VoteLocker.sol#209-231) - VoteLocker.userData (src/tokenomics/VoteLocker.sol#82)	Medium

Finding	Impact
Reentrancy in VoteLocker._processExpiredLocks(address,bool,address, uint256) (src/tokenomics/VoteLocker.sol#544-639): External calls: - _checkpointDelegate(delegates(_account),0,0) (src/tokenomics/VoteLocker.sol#619) - ckpts[ckpts.length - 1] = DelegateeCheckpoint((prevCkpt.votes + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48()) (src/tokenomics/VoteLocker.sol#709-712) - ckpts.push(DelegateeCheckpoint((prevCkpt.votes - unlocksSinceLatestCkpt + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48())) (src/tokenomics/VoteLocker.sol#731-739) - stakingToken.safeTransfer(_rewardAddress,reward) (src/tokenomics/VoteLocker.sol#629) - _lock(_account,locked) (src/tokenomics/VoteLocker.sol#635) - ckpts[ckpts.length - 1] = DelegateeCheckpoint((prevCkpt.votes + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48()) (src/tokenomics/VoteLocker.sol#709-712) - ckpts.push(DelegateeCheckpoint((prevCkpt.votes - unlocksSinceLatestCkpt + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48())) (src/tokenomics/VoteLocker.sol#731-739) State variables written after the call(s): - _lock(_account,locked) (src/tokenomics/VoteLocker.sol#635) - bal.locked = bal.locked.add(lockAmount) (src/tokenomics/VoteLocker.sol#376) VoteLocker.balances (src/tokenomics/VoteLocker.sol#94) can be used in cross function reentrancies: - VoteLocker.balances (src/tokenomics/VoteLocker.sol#94) - VoteLocker.claimableRewards(address) (src/tokenomics/VoteLocker.sol#1009-1026) - VoteLocker.lockedBalances(address) (src/tokenomics/VoteLocker.sol#898-924) - VoteLocker.updateReward(address) (src/tokenomics/VoteLocker.sol#209-231) - _lock(_account,locked) (src/tokenomics/VoteLocker.sol#635) - delegateeUnlocks[delegatee][unlockTime] += lockAmount (src/tokenomics/VoteLocker.sol#396)VoteLocker.delegateeUnlocks (src/tokenomics/VoteLocker.sol#103) can be used in cross function reentrancies: - VoteLocker.delegateeUnlocks (src/tokenomics/VoteLocker.sol#103) - VoteLocker.getPastVotes(address,uint256) (src/tokenomics/VoteLocker.sol#789-801)	Medium

Finding	Impact
Reentrancy in VoteLocker.getReward(address,bool) (src/tokenomics/VoteLocker.sol#422-441): External calls: - IERC20(_rewardsToken).safeTransfer(_account,reward) (src/tokenomics/VoteLocker.sol#434) State variables written after the call(s): - userData[_account][_rewardsToken].rewards = 0 (src/tokenomics/VoteLocker.sol#433) VoteLocker.userData (src/tokenomics/VoteLocker.sol#82) can be used in cross function reentrancies: - VoteLocker._earned(address,address,uint256) (src/tokenomics/VoteLocker.sol#1048-1056) - VoteLocker.getReward(address,bool) (src/tokenomics/VoteLocker.sol#422-441) - VoteLocker.updateReward(address) (src/tokenomics/VoteLocker.sol#209-231) - VoteLocker.userData (src/tokenomics/VoteLocker.sol#82)	Medium

Finding	Impact
Reentrancy in VoteLocker.queueNewRewards(address,uint256) (src/tokenomics/VoteLocker.sol#1094-1122): External calls: - IERC20(_rewardsToken).safeTransferFrom(msg.sender,address(this), _rewards) (src/tokenomics/VoteLocker.sol#1100) State variables written after the call(s): - _notifyReward(_rewardsToken,_rewards) (src/tokenomics/VoteLocker.sol#1106) - rdata.rewardRate = _reward.div(rewardsDuration).to96() (src/tokenomics/VoteLocker.sol#1136) - rdata.rewardRate = _reward.add(leftover).div(rewardsDuration).to96() (src/tokenomics/VoteLocker.sol#1140) - rewardData[token].rewardPerTokenStored = newRewardPerToken.to96() (src/tokenomics/VoteLocker.sol#216) - rewardData[token].lastUpdateTime = _lastTimeRewardApplicable(rewardData[token].periodFinish).to32() (src/tokenomics/VoteLocker.sol#217-218) - rdata.lastUpdateTime = block.timestamp.to32() (src/tokenomics/VoteLocker.sol#1147) - rdata.periodFinish = block.timestamp.add(rewardsDuration).to32() (src/tokenomics/VoteLocker.sol#1148) VoteLocker.rewardData (src/tokenomics/VoteLocker.sol#78) can be used in cross function reentrancies: - VoteLocker._rewardPerToken(address) (src/tokenomics/VoteLocker.sol#1072-1081) - VoteLocker.addReward(address,address) (src/tokenomics/VoteLocker.sol#272-281) - VoteLocker.approveRewardDistributor(address,address,bool) (src/tokenomics/VoteLocker.sol#290-296) - VoteLocker.lastTimeRewardApplicable(address) (src/tokenomics/VoteLocker.sol#1033-1035) - VoteLocker.recoverERC20(address,uint256) (src/tokenomics/VoteLocker.sol#329-335) - VoteLocker.rewardData (src/tokenomics/VoteLocker.sol#78) - VoteLocker.updateReward(address) (src/tokenomics/VoteLocker.sol#209-231) - _notifyReward(_rewardsToken,_rewards) (src/tokenomics/VoteLocker.sol#1117) - rdata.rewardRate = _reward.div(rewardsDuration).to96() (src/tokenomics/VoteLocker.sol#1136) - rdata.rewardRate = _reward.add(leftover).div(rewardsDuration).to96() (src/tokenomics/VoteLocker.sol#1140) - rewardData[token].rewardPerTokenStored = newRewardPerToken.to96()	Medium

Finding	Impact
Reentrancy in VoteLocker._lock(address,uint256) (src/tokenomics/VoteLocker.sol#362-405): External calls: - _checkpointDelegate(delegatee,lockAmount,0) (src/tokenomics/VoteLocker.sol#397) - ckpts[ckpts.length - 1] = DelegateeCheckpoint((prevCkpt.votes + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48()) (src/tokenomics/VoteLocker.sol#709-712) - ckpts.push(DelegateeCheckpoint((prevCkpt.votes - unlocksSinceLatestCkpt + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48())) (src/tokenomics/VoteLocker.sol#731-739) State variables written after the call(s): - e.supply = e.supply.add(lockAmount) (src/tokenomics/VoteLocker.sol#402) VoteLocker._epochs (src/tokenomics/VoteLocker.sol#92) can be used in cross function reentrancies: - VoteLocker._checkpointEpoch() (src/tokenomics/VoteLocker.sol#481-493) - VoteLocker._epochs (src/tokenomics/VoteLocker.sol#92) - VoteLocker.balanceAtEpochOf(uint256,address) (src/tokenomics/VoteLocker.sol#858-887) - VoteLocker.constructor(string,string,address,address) (src/tokenomics/VoteLocker.sol#184-201) - VoteLocker.epochCount() (src/tokenomics/VoteLocker.sol#977-979) - VoteLocker.epochs(uint256) (src/tokenomics/VoteLocker.sol#981-983) - VoteLocker.findEpochId(uint256) (src/tokenomics/VoteLocker.sol#966-968) - VoteLocker.totalSupplyAtEpoch(uint256) (src/tokenomics/VoteLocker.sol#941-963)	Medium

Finding	Impact
Reentrancy in VoteLocker.lock(address,uint256) (src/tokenomics/VoteLocker.sol#349-355): External calls: - stakingToken.safeTransferFrom(msg.sender,address(this),_amount) (src/tokenomics/VoteLocker.sol#351) - _lock(_account,_amount) (src/tokenomics/VoteLocker.sol#354) - ckpts[ckpts.length - 1] = DelegateeCheckpoint((prevCkpt.votes + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48()) (src/tokenomics/VoteLocker.sol#709-712) - ckpts.push(DelegateeCheckpoint((prevCkpt.votes - unlocksSinceLatestCkpt + _upcomingAddition - _upcomingDeduction).to208(),upcomingEpoch.to48())) (src/tokenomics/VoteLocker.sol#731-739) State variables written after the call(s): - _lock(_account,_amount) (src/tokenomics/VoteLocker.sol#354) - bal.locked = bal.locked.add(lockAmount) (src/tokenomics/VoteLocker.sol#376) VoteLocker.balances (src/tokenomics/VoteLocker.sol#94) can be used in cross function reentrancies: - VoteLocker.balances (src/tokenomics/VoteLocker.sol#94) - VoteLocker.claimableRewards(address) (src/tokenomics/VoteLocker.sol#1009-1026) - VoteLocker.lockedBalances(address) (src/tokenomics/VoteLocker.sol#898-924) - VoteLocker.updateReward(address) (src/tokenomics/VoteLocker.sol#209-231) - _lock(_account,_amount) (src/tokenomics/VoteLocker.sol#354) - lockedSupply = lockedSupply.add(_amount) (src/tokenomics/VoteLocker.sol#379)VoteLocker.lockedSupply (src/tokenomics/VoteLocker.sol#90) can be used in cross function reentrancies: - VoteLocker._rewardPerToken(address) (src/tokenomics/VoteLocker.sol#1072-1081) - VoteLocker.lockedSupply (src/tokenomics/VoteLocker.sol#90)	Medium
VoteLocker.lockedBalances(address).idx (src/tokenomics/VoteLocker.sol#906) is a local variable never initialized	Medium

Finding	Impact
VoteLocker.getReward(address,bool[]).i (src/tokenomics/VoteLocker.sol#455) is a local variable never initialized	Medium
VoteLocker.getReward(address,bool).i (src/tokenomics/VoteLocker.sol#429) is a local variable never initialized	Medium
Reentrancy in VoteLocker.queueNewRewards(address,uint256) (src/tokenomics/VoteLocker.sol#1094-1122): External calls: - IERC20(_rewardsToken).safeTransferFrom(msg.sender,address(this),_rewards) (src/tokenomics/VoteLocker.sol#1100) State variables written after the call(s): - queuedRewards[_rewardsToken] = 0 (src/tokenomics/VoteLocker.sol#1107) - queuedRewards[_rewardsToken] = 0 (src/tokenomics/VoteLocker.sol#1118) - queuedRewards[_rewardsToken] = _rewards (src/tokenomics/VoteLocker.sol#1120) - _notifyReward(_rewardsToken,_rewards) (src/tokenomics/VoteLocker.sol#1106) - userData[_account][token] = UserData(newRewardPerToken.to128(),_earned(_account,token,userBalance.locked).to128()) (src/tokenomics/VoteLocker.sol#220-223) - _notifyReward(_rewardsToken,_rewards) (src/tokenomics/VoteLocker.sol#1117) - userData[_account][token] = UserData(newRewardPerToken.to128(),_earned(_account,token,userBalance.locked).to128()) (src/tokenomics/VoteLocker.sol#220-223)	Low
Reentrancy in VoteLocker.recoverERC20(address,uint256) (src/tokenomics/VoteLocker.sol#329-335): External calls: - IERC20(_tokenAddress).safeTransfer(ADMIN_ADDRESS,_tokenAmount) (src/tokenomics/VoteLocker.sol#333) Event emitted after the call(s): - Recovered(_tokenAddress,_tokenAmount) (src/tokenomics/VoteLocker.sol#334)	Low
VoteLocker._notifyReward(address,uint256) (src/tokenomics/VoteLocker.sol#1129-1151) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= rdata.periodFinish (src/tokenomics/VoteLocker.sol#1135)	Low

Finding	Impact
VoteLocker._checkpointDelegate(address,uint256,uint256) (src/tokenomics/VoteLocker.sol#695-751) uses timestamp for comparisons Dangerous comparisons: - prevCkpt.epochStart == upcomingEpoch (src/tokenomics/VoteLocker.sol#708) - prevCkpt.epochStart + lockDuration <= upcomingEpoch (src/tokenomics/VoteLocker.sol#716) - nextEpoch > prevCkpt.epochStart (src/tokenomics/VoteLocker.sol#727)	Low
VoteLocker._processExpiredLocks(address,bool,address,uint256) (src/tokenomics/VoteLocker.sol#544-639) uses timestamp for comparisons Dangerous comparisons: - isShutdown || locks[length - 1].unlockTime <= expiryTime (src/tokenomics/VoteLocker.sol#562) - locks[i].unlockTime > expiryTime (src/tokenomics/VoteLocker.sol#587) - reward > 0 (src/tokenomics/VoteLocker.sol#624)	Low
VoteLocker.getPastVotes(address,uint256) (src/tokenomics/VoteLocker.sol#789-801) uses timestamp for comparisons Dangerous comparisons: - timestamp > block.timestamp (src/tokenomics/VoteLocker.sol#790) - votes == 0 || ckpt.epochStart + lockDuration <= epoch (src/tokenomics/VoteLocker.sol#794) - epoch > ckpt.epochStart (src/tokenomics/VoteLocker.sol#797)	Low
VoteLocker._checkpointsLookup(VoteLocker.DelegateeCheckpoint[],uint256) (src/tokenomics/VoteLocker.sol#816-833) uses timestamp for comparisons Dangerous comparisons: - ckpts[mid].epochStart > epochStart (src/tokenomics/VoteLocker.sol#825)	Low
VoteLocker.lockedBalances(address) (src/tokenomics/VoteLocker.sol#898-924) uses timestamp for comparisons Dangerous comparisons: - locks[i].unlockTime > block.timestamp (src/tokenomics/VoteLocker.sol#909)	Low

Finding	Impact
VoteLocker.totalSupplyAtEpoch(uint256) (src/tokenomics/VoteLocker.sol#941-963) uses timestamp for comparisons Dangerous comparisons: - epochStart >= block.timestamp (src/tokenomics/VoteLocker.sol#943) - i > 0 (src/tokenomics/VoteLocker.sol#950) - e.date == epochStart (src/tokenomics/VoteLocker.sol#952) - e.date <= cutoffEpoch (src/tokenomics/VoteLocker.sol#954) - _epoch > lastIndex (src/tokenomics/VoteLocker.sol#948)	Low
VoteLocker.delegate(address) (src/tokenomics/VoteLocker.sol#650-693) uses timestamp for comparisons Dangerous comparisons: - currentLock.unlockTime > upcomingEpoch (src/tokenomics/VoteLocker.sol#672)	Low
VoteLocker.balanceAtEpochOf(uint256,address) (src/tokenomics/VoteLocker.sol#858-887) uses timestamp for comparisons Dangerous comparisons: - epochStart >= block.timestamp (src/tokenomics/VoteLocker.sol#860) - lockEpoch < epochStart (src/tokenomics/VoteLocker.sol#873) - lockEpoch > cutoffEpoch (src/tokenomics/VoteLocker.sol#874)	Low
VoteLocker._checkpointEpoch() (src/tokenomics/VoteLocker.sol#481-493) uses timestamp for comparisons Dangerous comparisons: - nextEpochDate < currentEpoch (src/tokenomics/VoteLocker.sol#487) - nextEpochDate != currentEpoch (src/tokenomics/VoteLocker.sol#488)	Low
VoteLocker.getPastTotalSupply(uint256) (src/tokenomics/VoteLocker.sol#807-810) uses timestamp for comparisons Dangerous comparisons: - timestamp >= block.timestamp (src/tokenomics/VoteLocker.sol#808)	Low
VoteLocker.queueNewRewards(address,uint256) (src/tokenomics/VoteLocker.sol#1094-1122) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= rdata.periodFinish (src/tokenomics/VoteLocker.sol#1105) - queuedRatio < newRewardRatio (src/tokenomics/VoteLocker.sol#1116)	Low

Finding	Impact
VoteLocker._lock(address,uint256) (src/tokenomics/VoteLocker.sol#362-405) uses timestamp for comparisons Dangerous comparisons: - idx == 0 || userLocks[_account][idx - 1].unlockTime < unlockTime (src/tokenomics/VoteLocker.sol#385)	Low
End of table for VoteLocker.sol	

src/tokenomics/GaugeFactory.sol

Slither results for GaugeFactory.sol	
Finding	**Impact**
GaugeFactory.setImplementation(address)._implementation (src/tokenomics/GaugeFactory.sol#79) lacks a zero-check on : - implementation = _implementation (src/tokenomics/GaugeFactory.sol#80)	Low
GaugeFactory.constructor(address,address)._implementation (src/tokenomics/GaugeFactory.sol#42) lacks a zero-check on : - implementation = _implementation (src/tokenomics/GaugeFactory.sol#43)	Low
Reentrancy in GaugeFactory.deployGauge(address) (src/tokenomics/GaugeFactory.sol#57-71): External calls: - ILiquidityGauge(gauge).initialize(lpToken) (src/tokenomics/GaugeFactory.sol#61) State variables written after the call(s): - gaugeToLpToken[gauge] = lpToken (src/tokenomics/GaugeFactory.sol#65) - isFactoryGauge[gauge] = true (src/tokenomics/GaugeFactory.sol#63) - lpTokenToGauge[lpToken] = gauge (src/tokenomics/GaugeFactory.sol#66)	Low
Reentrancy in GaugeFactory.deployGauge(address) (src/tokenomics/GaugeFactory.sol#57-71): External calls: - ILiquidityGauge(gauge).initialize(lpToken) (src/tokenomics/GaugeFactory.sol#61) Event emitted after the call(s): - NewGauge(lpToken,gauge) (src/tokenomics/GaugeFactory.sol#68)	Low
End of table for GaugeFactory.sol	

src/tokenomics/Minter.sol

Slither results for Minter.sol	
Finding	**Impact**
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) ignores return value by IERC20(token).transfer(account,toMintAmount) (src/tokenomics/Minter.sol#271)	High
Minter._mintableInTimeframe(uint256,uint256) (src/tokenomics/Minter.sol#202-246) performs a multiplication on the result of a division: - currentRate = currentRate * RATE_REDUCTION_COEFFICIENT / SCALED_ONE (src/tokenomics/Minter.sol#212) - currentRate = currentRate * RATE_REDUCTION_COEFFICIENT / SCALED_ONE (src/tokenomics/Minter.sol#238)	Medium
Minter._mintableInTimeframe(uint256,uint256) (src/tokenomics/Minter.sol#202-246) performs a multiplication on the result of a division: - toMint += currentRate * (currentEnd - currentStart) (src/tokenomics/Minter.sol#232) - currentRate = currentRate * RATE_REDUCTION_COEFFICIENT / SCALED_ONE (src/tokenomics/Minter.sol#238)	Medium
Minter._mintableInTimeframe(uint256,uint256).toMint (src/tokenomics/Minter.sol#205) is a local variable never initialized	Medium
Minter._mintableInTimeframe(uint256,uint256).i (src/tokenomics/Minter.sol#217) is a local variable never initialized	Medium
Minter.mintMultiple(address[]).i (src/tokenomics/Minter.sol#99) is a local variable never initialized	Medium
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) ignores return value by ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/Minter.sol#260)	Medium
Minter.constructor(address,address).token_- (src/tokenomics/Minter.sol#54) lacks a zero-check on : - token = token_ (src/tokenomics/Minter.sol#55)	Low

Finding	Impact
Minter.constructor(address,address).controller_- (src/tokenomics/Minter.sol#54) lacks a zero-check on : - controller = controller_ (src/tokenomics/Minter.sol#56)	Low
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) has external calls inside a loop: ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/Minter.sol#260)	Low
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) has external calls inside a loop: totalMint = ILiquidityGauge(gauge).integrateFraction(account) (src/tokenomics/Minter.sol#261)	Low
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) has external calls inside a loop: IERC20(token).transfer(account,toMintAmount) (src/tokenomics/Minter.sol#271)	Low
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) has external calls inside a loop: IGaugeController(controller).getGaugeType(gauge) == 0 (src/tokenomics/Minter.sol#254)	Low
Reentrancy in Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275): External calls: - ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/Minter.sol#260) State variables written after the call(s): - minted[account][gauge] = totalMint (src/tokenomics/Minter.sol#268) - mintedSupply = _newMintedSupply (src/tokenomics/Minter.sol#269)	Low
Reentrancy in Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275): External calls: - ILiquidityGauge(gauge).userCheckpoint(account) (src/tokenomics/Minter.sol#260) - IERC20(token).transfer(account,toMintAmount) (src/tokenomics/Minter.sol#271) Event emitted after the call(s): - Minted(account,gauge,toMintAmount) (src/tokenomics/Minter.sol#273)	Low

Finding	Impact
Minter._mintableInTimeframe(uint256,uint256) (src/tokenomics/Minter.sol#202-246) uses timestamp for comparisons Dangerous comparisons: - end > currentEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#210) - end > currentEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#215) - end >= currentEpochTime (src/tokenomics/Minter.sol#218) - currentEnd > currentEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#221) - currentStart >= currentEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#225) - currentStart < currentEpochTime (src/tokenomics/Minter.sol#228) - start >= currentEpochTime (src/tokenomics/Minter.sol#234)	Low
Minter.startEpochTimeWrite() (src/tokenomics/Minter.sol#133-139) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= startEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#135)	Low
Minter.futureEpochTimeWrite() (src/tokenomics/Minter.sol#145-152) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= startEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#147)	Low
Minter._mintFor(address,address) (src/tokenomics/Minter.sol#253-275) uses timestamp for comparisons Dangerous comparisons: - block.timestamp >= startEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#256) - _newMintedSupply > _availableSupply() (src/tokenomics/Minter.sol#265)	Low
Minter.updateMiningParameters() (src/tokenomics/Minter.sol#124-127) uses timestamp for comparisons Dangerous comparisons: - block.timestamp < startEpochTime + RATE_REDUCTION_TIME (src/tokenomics/Minter.sol#125)	Low
End of table for Minter.sol	

src/tokenomics/LiquidityGauge.sol

Slither results for LiquidityGauge.sol	
Finding	**Impact**
LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) performs a multiplication on the result of a division: - lim += L * userBalance / totalLockedSupply * (100 - TOKENLESS_PRODUCTION) / 100 (src/tokenomics/LiquidityGauge.sol#418)	Medium
LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) performs a multiplication on the result of a division: - w = IGaugeController(GAUGE_CONTROLLER).gaugeRelativeWeight(address(this),prevWeekTime / WEEK * WEEK) (src/tokenomics/LiquidityGauge.sol#473-475)	Medium
LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) performs a multiplication on the result of a division: - weekTime = (periodTime + WEEK) / WEEK * WEEK (src/tokenomics/LiquidityGauge.sol#468)	Medium
LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) uses a dangerous strict equality: - weekTime == block.timestamp (src/tokenomics/LiquidityGauge.sol#500)	Medium

Finding	Impact
Reentrancy in LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524): External calls: - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - integrateInvSupply.push(_integrateInvSupply) (src/tokenomics/LiquidityGauge.sol#510)LiquidityGauge.integrateInvSupply (src/tokenomics/LiquidityGauge.sol#77) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.initialize(address) (src/tokenomics/LiquidityGauge.sol#152-171) - LiquidityGauge.integrateInvSupply (src/tokenomics/LiquidityGauge.sol#77) - integrateInvSupplyBoosted = _integrateInvSupplyBoosted (src/tokenomics/LiquidityGauge.sol#511)LiquidityGauge.integrateInvSupplyBoosted (src/tokenomics/LiquidityGauge.sol#78) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.integrateInvSupplyBoosted (src/tokenomics/LiquidityGauge.sol#78) - period = _period (src/tokenomics/LiquidityGauge.sol#508)LiquidityGauge.period (src/tokenomics/LiquidityGauge.sol#73) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.integrateCheckpoint() (src/tokenomics/LiquidityGauge.sol#187-189) - LiquidityGauge.period (src/tokenomics/LiquidityGauge.sol#73) - periodTimestamp.push(block.timestamp) (src/tokenomics/LiquidityGauge.sol#509)LiquidityGauge.periodTimestamp (src/tokenomics/LiquidityGauge.sol#74) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.initialize(address) (src/tokenomics/LiquidityGauge.sol#152-171) - LiquidityGauge.integrateCheckpoint() (src/tokenomics/LiquidityGauge.sol#187-189) - LiquidityGauge.periodTimestamp	Medium

Finding	Impact
Reentrancy in LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524): External calls: - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) State variables written after the call(s): - inflationRate = newRate (src/tokenomics/LiquidityGauge.sol#447)LiquidityGauge.inflationRate (src/tokenomics/LiquidityGauge.sol#91) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.inflationRate (src/tokenomics/LiquidityGauge.sol#91) - LiquidityGauge.initialize(address) (src/tokenomics/LiquidityGauge.sol#152-171)	Medium

Finding	Impact
Reentrancy in LiquidityGauge._deposit(uint256,address) (src/tokenomics/LiquidityGauge.sol#340-357): External calls: - _checkpoint(user) (src/tokenomics/LiquidityGauge.sol#341) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - balanceOf[user] = newUserBalance (src/tokenomics/LiquidityGauge.sol#347)LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - totalSupply = _totalSupply (src/tokenomics/LiquidityGauge.sol#348)LiquidityGauge.totalSupply (src/tokenomics/LiquidityGauge.sol#62) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.totalSupply (src/tokenomics/LiquidityGauge.sol#62) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - _updateLiquidityLimit(user,newUserBalance,_totalSupply) (src/tokenomics/LiquidityGauge.sol#350) - workingBalances[user] = lim (src/tokenomics/LiquidityGauge.sol#423)LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) - _updateLiquidityLimit(user,newUserBalance,_totalSupply)	Medium

Finding	Impact
Reentrancy in LiquidityGauge._withdraw(uint256) (src/tokenomics/LiquidityGauge.sol#363-380): External calls: - _checkpoint(msg.sender) (src/tokenomics/LiquidityGauge.sol#364) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - balanceOf[msg.sender] = newUserBalance (src/tokenomics/LiquidityGauge.sol#370)LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - totalSupply = _totalSupply (src/tokenomics/LiquidityGauge.sol#371)LiquidityGauge.totalSupply (src/tokenomics/LiquidityGauge.sol#62) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.totalSupply (src/tokenomics/LiquidityGauge.sol#62) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - _updateLiquidityLimit(msg.sender,newUserBalance,_totalSupply) (src/tokenomics/LiquidityGauge.sol#373) - workingBalances[user] = lim (src/tokenomics/LiquidityGauge.sol#423)LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) - _updateLiquidityLimit(msg.sender,newUserBalance,_totalSupply)	Medium

Finding	Impact
Reentrancy in LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205): External calls: - _checkpoint(user) (src/tokenomics/LiquidityGauge.sol#202) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#203) - workingBalances[user] = lim (src/tokenomics/LiquidityGauge.sol#423)LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#203) - workingSupply = _workingSupply (src/tokenomics/LiquidityGauge.sol#425)LiquidityGauge.workingSupply (src/tokenomics/LiquidityGauge.sol#69) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.workingSupply (src/tokenomics/LiquidityGauge.sol#69)	Medium

Finding	Impact
Reentrancy in LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233): External calls: - _checkpoint(user) (src/tokenomics/LiquidityGauge.sol#231) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#232) - workingBalances[user] = lim (src/tokenomics/LiquidityGauge.sol#423)LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.workingBalances (src/tokenomics/LiquidityGauge.sol#68) - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#232) - workingSupply = _workingSupply (src/tokenomics/LiquidityGauge.sol#425)LiquidityGauge.workingSupply (src/tokenomics/LiquidityGauge.sol#69) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge._updateLiquidityLimit(address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#413-428) - LiquidityGauge.workingSupply (src/tokenomics/LiquidityGauge.sol#69)	Medium

Finding	Impact
Reentrancy in LiquidityGauge._transfer(address,address,uint256) (src/tokenomics/LiquidityGauge.sol#388-405): External calls: - _checkpoint(from) (src/tokenomics/LiquidityGauge.sol#389) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) - _checkpoint(to) (src/tokenomics/LiquidityGauge.sol#390) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - balanceOf[from] = newFromBalance (src/tokenomics/LiquidityGauge.sol#396)LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - balanceOf[to] = newToBalance (src/tokenomics/LiquidityGauge.sol#400)LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.balanceOf (src/tokenomics/LiquidityGauge.sol#61) - LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) - LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205) - _checkpoint(to) (src/tokenomics/LiquidityGauge.sol#390) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445)LiquidityGauge.futureEpochTime (src/tokenomics/LiquidityGauge.sol#59) can be used in cross function reentrancies: - LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) - LiquidityGauge.futureEpochTime (src/tokenomics/LiquidityGauge.sol#59) - LiquidityGauge.initialize(address)	Medium

Finding	Impact
LiquidityGauge._checkpoint(address).i (src/tokenomics/LiquidityGauge.sol#471) is a local variable never initialized	Medium
LiquidityGauge._checkpoint(address).endTimestamp (src/tokenomics/LiquidityGauge.sol#451) is a local variable never initialized	Medium
LiquidityGauge.constructor(address,address,address,address)._minter (src/tokenomics/LiquidityGauge.sol#132) lacks a zero-check on : - MINTER = _minter (src/tokenomics/LiquidityGauge.sol#137)	Low
LiquidityGauge.constructor(address,address,address,address)._vlToken (src/tokenomics/LiquidityGauge.sol#134) lacks a zero-check on : - VL_TOKEN = _vlToken (src/tokenomics/LiquidityGauge.sol#139)	Low
LiquidityGauge.initialize(address)._lpToken (src/tokenomics/LiquidityGauge.sol#152) lacks a zero-check on : - lpToken = _lpToken (src/tokenomics/LiquidityGauge.sol#155)	Low
LiquidityGauge.constructor(address,address,address,address)._minterEscrow (src/tokenomics/LiquidityGauge.sol#133) lacks a zero-check on : - MINTER_ESCROW = _minterEscrow (src/tokenomics/LiquidityGauge.sol#138)	Low

Finding	Impact
Reentrancy in LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524): External calls: - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - integrateBoostedCheckpointOf[user] = block.timestamp (src/tokenomics/LiquidityGauge.sol#523) - integrateBoostedInvSupplyOf[user] = _integrateInvSupplyBoosted (src/tokenomics/LiquidityGauge.sol#522) - integrateCheckpointOf[user] = block.timestamp (src/tokenomics/LiquidityGauge.sol#517) - integrateFraction[user] += _userBalance * (_integrateInvSupply - integrateInvSupplyOf[user]) / 10 ** 18 (src/tokenomics/LiquidityGauge.sol#514-515) - integrateFractionBoosted[user] += _workingBalance * (_integrateInvSupplyBoosted - integrateBoostedInvSupplyOf[user]) / 10 ** 18 (src/tokenomics/LiquidityGauge.sol#520-521) - integrateInvSupplyOf[user] = _integrateInvSupply (src/tokenomics/LiquidityGauge.sol#516)	Low
Reentrancy in LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524): External calls: - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) State variables written after the call(s): - inflationRateBoosted = newRateBoosted (src/tokenomics/LiquidityGauge.sol#454)	Low

Finding	Impact
Reentrancy in LiquidityGauge._transfer(address,address,uint256) (src/tokenomics/LiquidityGauge.sol#388-405): External calls: - _checkpoint(from) (src/tokenomics/LiquidityGauge.sol#389) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) - _checkpoint(to) (src/tokenomics/LiquidityGauge.sol#390) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) State variables written after the call(s): - _checkpoint(to) (src/tokenomics/LiquidityGauge.sol#390) - integrateBoostedCheckpointOf[user] = block.timestamp (src/tokenomics/LiquidityGauge.sol#523) - _checkpoint(to) (src/tokenomics/LiquidityGauge.sol#390) - integrateCheckpointOf[user] = block.timestamp (src/tokenomics/LiquidityGauge.sol#517)	Low
Reentrancy in LiquidityGauge.userCheckpoint(address) (src/tokenomics/LiquidityGauge.sol#198-205): External calls: - _checkpoint(user) (src/tokenomics/LiquidityGauge.sol#202) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) Event emitted after the call(s): - UpdateLiquidityLimit(user,l,L,lim,_workingSupply) (src/tokenomics/LiquidityGauge.sol#427) - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#203)	Low

Finding	Impact
Reentrancy in LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233): External calls: - _checkpoint(user) (src/tokenomics/LiquidityGauge.sol#231) - futureEpochTime = IMinter(MINTER).futureEpochTimeWrite() (src/tokenomics/LiquidityGauge.sol#445) - IGaugeController(GAUGE_CONTROLLER).checkpointGauge(address(this)) (src/tokenomics/LiquidityGauge.sol#465) Event emitted after the call(s): - UpdateLiquidityLimit(user,l,L,lim,_workingSupply) (src/tokenomics/LiquidityGauge.sol#427) - _updateLiquidityLimit(user,balanceOf[user],totalSupply) (src/tokenomics/LiquidityGauge.sol#232)	Low
LiquidityGauge.kick(address) (src/tokenomics/LiquidityGauge.sol#221-233) uses timestamp for comparisons Dangerous comparisons: - IVoteLocker(VL_TOKEN).balanceOf(user) > 0 && vlTime < lastTime (src/tokenomics/LiquidityGauge.sol#228)	Low
LiquidityGauge.permit(address,address,uint256,uint256) (src/tokenomics/LiquidityGauge.sol#311-331) uses timestamp for comparisons Dangerous comparisons: - block.timestamp > deadline (src/tokenomics/LiquidityGauge.sol#316)	Low
LiquidityGauge._checkpoint(address) (src/tokenomics/LiquidityGauge.sol#434-524) uses timestamp for comparisons Dangerous comparisons: - prevFutureEpoch >= periodTime (src/tokenomics/LiquidityGauge.sol#444) - block.timestamp > periodTime (src/tokenomics/LiquidityGauge.sol#461) - weekTime > block.timestamp (src/tokenomics/LiquidityGauge.sol#469) - prevFutureEpoch >= prevWeekTime && prevFutureEpoch < weekTime (src/tokenomics/LiquidityGauge.sol#477) - endTimestamp >= prevWeekTime && endTimestamp < weekTime && endTimestamp != 0 (src/tokenomics/LiquidityGauge.sol#490) - weekTime == block.timestamp (src/tokenomics/LiquidityGauge.sol#500) - weekTime + WEEK > block.timestamp (src/tokenomics/LiquidityGauge.sol#503)	Low

Finding	Impact
End of table for LiquidityGauge.sol	

src/tokenomics/GemMinterRebalancingReward.sol

Slither results for GemMinterRebalancingReward.sol	
Finding	**Impact**
GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115) uses arbitrary from in transferFrom: IERC20(gem).transferFrom(INCENTIVES_MS,account,amount) (src/tokenomics/GemMinterRebalancingReward.sol#111)	High
GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115) ignores return value by IERC20(gem).transferFrom(INCENTIVES_MS,account,amount) (src/tokenomics/GemMinterRebalancingReward.sol#111)	High
GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115) uses a dangerous strict equality: - amount == 0 (src/tokenomics/GemMinterRebalancingReward.sol#110)	Medium

Finding	Impact
Reentrancy in GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115):External calls: - IERC20(gem).transferFrom(INCENTIVES_MS,account,amount) (src/tokenomics/GemMinterRebalancingReward.sol#111) State variables written after the call(s): - totalGemMinted += amount (src/tokenomics/GemMinterRebalancingReward.sol#112)GemMinterRebalancingReward.totalGemMinted (src/tokenomics/GemMinterRebalancingReward.sol#49) can be used in cross function reentrancies: - GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115) - GemMinterRebalancingReward.totalGemMinted (src/tokenomics/GemMinterRebalancingReward.sol#49)	Medium
Reentrancy in GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115):External calls: - IERC20(gem).transferFrom(INCENTIVES_MS,account,amount) (src/tokenomics/GemMinterRebalancingReward.sol#111) Event emitted after the call(s): - RebalancingRewardDistributed(pool,account,address(gem),amount) (src/tokenomics/GemMinterRebalancingReward.sol#113)	Low
GemMinterRebalancingReward._distributeRebalancingRewards(address,address,uint256) (src/tokenomics/GemMinterRebalancingReward.sol#103-115) uses timestamp for comparisons Dangerous comparisons: - totalGemMinted + amount > _MAX_REBALANCING_REWARDS (src/tokenomics/GemMinterRebalancingReward.sol#107) - amount == 0 (src/tokenomics/GemMinterRebalancingReward.sol#110)	Low
End of table for GemMinterRebalancingReward.sol	

src/RewardManager.sol

Slither results for RewardManager.sol	
Finding	**Impact**

Finding	Impact
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) uses arbitrary from in transferFrom: AURAToken.transferFrom(address(omnipool),msg.sender,auraAmount) (src/RewardManager.sol#169)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) uses arbitrary from in transferFrom: AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) uses arbitrary from in transferFrom: BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) uses arbitrary from in transferFrom: BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) uses arbitrary from in transferFrom: AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360)	High
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) uses arbitrary from in transferFrom: BALToken.transferFrom(address(omnipool),msg.sender,balAmount) (src/RewardManager.sol#165)	High
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) uses arbitrary from in transferFrom: GEMToken.transferFrom(address(omnipool),msg.sender,gemAmount) (src/RewardManager.sol#173)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) ignores return value by BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359)	High
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) ignores return value by BALToken.transferFrom(address(omnipool),msg.sender,balAmount) (src/RewardManager.sol#165)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) ignores return value by BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356)	High
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) ignores return value by GEMToken.transferFrom(address(omnipool),msg.sender,gemAmount) (src/RewardManager.sol#173)	High

Finding	Impact
RewardManager.claimEarnings() (src/RewardManager.sol#142-180) ignores return value by AURAToken.transferFrom(address(omnipool),msg.sender,auraAmount) (src/RewardManager.sol#169)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) ignores return value by AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357)	High
RewardManager._claimProtocolFees() (src/RewardManager.sol#338-361) ignores return value by AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360)	High
Reentrancy in RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233): External calls: - extraRewardsLength = auraPool.extraRewardsLength() (src/RewardManager.sol#206) - extraReward = auraPool.extraRewards(j) (src/RewardManager.sol#208) State variables written after the call(s): - _extraRewardTokens.push(extraRewardToken) (src/RewardManager.sol#218)RewardManager._extraRewardTokens (src/RewardManager.sol#68) can be used in cross function reentrancies: - RewardManager._swapExtraReward() (src/RewardManager.sol#411-431) - RewardManager.getExtraRewardToken(uint256) (src/RewardManager.sol#120-123) - RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233) - ++ _extraRewardTokensLength (src/RewardManager.sol#220)RewardManager._extraRewardTokensLength (src/RewardManager.sol#67) can be used in cross function reentrancies: - RewardManager._swapExtraReward() (src/RewardManager.sol#411-431) - RewardManager.getExtraRewardToken(uint256) (src/RewardManager.sol#120-123) - RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233) - _extraRewardTokensMap[extraRewardToken] = true (src/RewardManager.sol#217)RewardManager._extraRewardTokensMap (src/RewardManager.sol#69) can be used in cross function reentrancies: - RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233)	Medium

Finding	Impact
Reentrancy in RewardManager.claimEarnings() (src/RewardManager.sol#142-180): External calls: - _updateUserState(msg.sender) (src/RewardManager.sol#144) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - omnipool.approve(address(this),BAL,balToClaim) (src/RewardManager.sol#347) - omnipool.approve(address(this),AURA,auraToClaim) (src/RewardManager.sol#348) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) - BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356) - AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357) - BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359) - AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360) State variables written after the call(s): - AURAMeta.accountShare[msg.sender] = 0 (src/RewardManager.sol#158) RewardManager.AURAMeta (src/RewardManager.sol#61) can be used in cross function reentrancies: - RewardManager.AURAMeta (src/RewardManager.sol#61) - RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333) - RewardManager._updateRewards(address,uint256) (src/RewardManager.sol#282-297) - RewardManager.claimEarnings() (src/RewardManager.sol#142-180) - BALMeta.accountShare[msg.sender] = 0 (src/RewardManager.sol#157) RewardManager.BALMeta (src/RewardManager.sol#60) can be used in cross function reentrancies: - RewardManager.BALMeta (src/RewardManager.sol#60) - RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333) - RewardManager._updateRewards(address,uint256) (src/RewardManager.sol#282-297) - RewardManager.claimEarnings() (src/RewardManager.sol#142-180) - GEMMeta.accountShare[msg.sender] = 0 (src/RewardManager.sol#159) RewardManager.GEMMeta (src/RewardManager.sol#62) can be used in cross function reentrancies: - RewardManager.GEMMeta (src/RewardManager.sol#62) - RewardManager._updateOmnipoolState()	Medium

Finding	Impact
Reentrancy in RewardManager._updateUserState(address) (src/RewardManager.sol#267-276): External calls: - _updateOmnipoolState() (src/RewardManager.sol#272) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - omnipool.approve(address(this),BAL,balToClaim) (src/RewardManager.sol#347) - omnipool.approve(address(this),AURA,auraToClaim) (src/RewardManager.sol#348) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) - BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356) - AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357) - BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359) - AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360) State variables written after the call(s): - _updateRewards(_account,deposited) (src/RewardManager.sol#275) - AURAMeta.accountShare[account] += auraShare (src/RewardManager.sol#290) - AURAMeta.accountIntegral[account] = AURAMeta.earnedIntegral (src/RewardManager.sol#291) RewardManager.AURAMeta (src/RewardManager.sol#61) can be used in cross function reentrancies: - RewardManager.AURAMeta (src/RewardManager.sol#61) - RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333) - RewardManager._updateRewards(address,uint256) (src/RewardManager.sol#282-297) - RewardManager.claimEarnings() (src/RewardManager.sol#142-180) - _updateRewards(_account,deposited) (src/RewardManager.sol#275) - BALMeta.accountShare[account] += balShare (src/RewardManager.sol#285) - BALMeta.accountIntegral[account] = BALMeta.earnedIntegral (src/RewardManager.sol#286) RewardManager.BALMeta (src/RewardManager.sol#60) can be used in cross function reentrancies: - RewardManager.BALMeta (src/RewardManager.sol#60) - RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333)	Medium

Finding	Impact
RewardManager._virtualBalanceRewardAddrToTokenAddr(address) (src/RewardManager.sol#255-261) has external calls inside a loop: IBaseToken(IRewardToken(rewardAddr).rewardToken()).baseToken() (src/RewardManager.sol#260)	Low
RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233) has external calls inside a loop: extraReward = auraPool.extraRewards(j) (src/RewardManager.sol#208)	Low
RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233) has external calls inside a loop: underlyingPool = omnipool.getUnderlyingPool(i_scope_0) (src/RewardManager.sol#203)	Low
RewardManager.setExtraRewardTokens() (src/RewardManager.sol#187-233) has external calls inside a loop: extraRewardsLength = auraPool.extraRewardsLength() (src/RewardManager.sol#206)	Low
Reentrancy in RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333): External calls: - (earnedBAL,earnedAURA,earnedGEM) = _claimOmnipoolRewards() (src/RewardManager.sol#304) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) State variables written after the call(s): - AURAMeta.earnedIntegral += (earnedAURA * SCALED_ONE) / totalDeposited (src/RewardManager.sol#322) - AURAMeta.lastEarned += earnedAURA (src/RewardManager.sol#323) - BALMeta.earnedIntegral += (earnedBAL * SCALED_ONE) / totalDeposited (src/RewardManager.sol#319) - BALMeta.lastEarned += earnedBAL (src/RewardManager.sol#320) - GEMMeta.earnedIntegral += (earnedGEM * SCALED_ONE) / totalDeposited (src/RewardManager.sol#325) - GEMMeta.lastEarned += earnedGEM (src/RewardManager.sol#326) - protocolFeesAURABalance += protocolFeesAURA (src/RewardManager.sol#311) - protocolFeesBALBalance += protocolFeesBAL (src/RewardManager.sol#310)	Low

Finding	Impact
Reentrancy in RewardManager._claimUnderlyingPoolRewards(uint8) (src/RewardManager.sol#394-405): External calls: - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) Event emitted after the call(s): - UnderlyingPoolRewardClaimed(omnipool.getUnderlyingPool(_poolIndex),BAL,IERC20(BAL).balanceOf(address(omnipool))) (src/RewardManager.sol#399-403)	Low

Finding	Impact
Reentrancy in RewardManager.claimEarnings() (src/RewardManager.sol#142-180): External calls: - _updateUserState(msg.sender) (src/RewardManager.sol#144) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - omnipool.approve(address(this),BAL,balToClaim) (src/RewardManager.sol#347) - omnipool.approve(address(this),AURA,auraToClaim) (src/RewardManager.sol#348) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) - BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356) - AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357) - BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359) - AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360) - omnipool.approve(address(this),BAL,balAmount) (src/RewardManager.sol#164) - BALToken.transferFrom(address(omnipool),msg.sender,balAmount) (src/RewardManager.sol#165) - omnipool.approve(address(this),AURA,auraAmount) (src/RewardManager.sol#168) - AURAToken.transferFrom(address(omnipool),msg.sender,auraAmount) (src/RewardManager.sol#169) - omnipool.approve(address(this),GEM,gemAmount) (src/RewardManager.sol#172) - GEMToken.transferFrom(address(omnipool),msg.sender,gemAmount) (src/RewardManager.sol#173) Event emitted after the call(s): - RewardClaimed(msg.sender,balAmount,auraAmount,gemAmount) (src/RewardManager.sol#177)	Low

Finding	Impact
Reentrancy in RewardManager._swapExtraReward() (src/RewardManager.sol#411-431): External calls: - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) Event emitted after the call(s): - RewardSwapped(extraRewardToken,extraRewardBalance,GEM,IERC20(GEM).balanceOf(address(omnipool))) (src/RewardManager.sol#419-424)	Low
Reentrancy in RewardManager._updateOmnipoolState() (src/RewardManager.sol#303-333): External calls: - (earnedBAL,earnedAURA,earnedGEM) = _claimOmnipoolRewards() (src/RewardManager.sol#304) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) - _claimProtocolFees() (src/RewardManager.sol#329) - omnipool.approve(address(this),BAL,balToClaim) (src/RewardManager.sol#347) - omnipool.approve(address(this),AURA,auraToClaim) (src/RewardManager.sol#348) - BALToken.transferFrom(omnipoolAddr,opalTreasury,balTreasuryPart) (src/RewardManager.sol#356) - AURAToken.transferFrom(omnipoolAddr,opalTreasury,auraTreasuryPart) (src/RewardManager.sol#357) - BALToken.transferFrom(omnipoolAddr,voteLocker,balToClaim - balTreasuryPart) (src/RewardManager.sol#359) - AURAToken.transferFrom(omnipoolAddr,voteLocker,auraToClaim - auraTreasuryPart) (src/RewardManager.sol#360) Event emitted after the call(s): - RewardUpdated(earnedBAL,earnedAURA,earnedGEM) (src/RewardManager.sol#332)	Low

Finding	Impact
Reentrancy in RewardManager._claimOmnipoolRewards() (src/RewardManager.sol#368-392): External calls: - _claimUnderlyingPoolRewards(i) (src/RewardManager.sol#376) - success = IAuraPool(omnipool.getUnderlyingPool(_poolIndex)).getReward(address(omnipool),true) (src/RewardManager.sol#396-397) - _swapExtraReward() (src/RewardManager.sol#383) - success = omnipool.swapForGem(extraRewardToken,extraRewardBalance) (src/RewardManager.sol#417) Event emitted after the call(s): - RewardSwapped(extraRewardToken,extraRewardBalance,GEM,IERC20(GEM).balanceOf(address(omnipool))) (src/RewardManager.sol#419-424) - _swapExtraReward() (src/RewardManager.sol#383)	Low
End of table for RewardManager.sol	

Results Summary:

The findings obtained as a result of the Slither scan were reviewed:
- The lack of zero-check on findings were added to the report.
- The uses timestamp for comparisons and has external calls inside loop informational findings were manually reviewed and determined false-positives.
- The uses arbitrary from in transferFrom, variable never initialized, reentrancy, sends eth to arbitrary user, uses a dangerous strict equality and ignores return value vulnerabilities were manually reviewed and determined false-positives.