SALUS SECURITY

JUL 2025

CODE SECURITY ASSESSMENT

UPTOP V2

Overview

Project Summary

- Name: UpTopV2 - Fund
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/dsoftgames/UpTopV2Fund
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	UpTopV2 - Fund
Version	v3
Type	Solidity
Dates	Jul 29 2025
Logs	Jul 19 2025, Jul 25 2025, Jul 29 2025

Vulnerability Summary

Total High-Severity issues	5
Total Medium-Severity issues	10
Total Low-Severity issues	4
Total informational issues	3
Total	22

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Potential inflation attack in FundVault	High	Business Logic	Resolved
2	Funds will be forced to keep in the vault	High	Business Logic	Resolved
3	Missing input validation for orderId	High	Business Logic	Resolved
4	Incorrect `amountToSwap` calculation in the `swapToTokenBase` function	High	Business Logic	Resolved
5	Inappropriate reward distribution design in the GaugeV3 contract	High	Business Logic	Resolved
6	Malicious users can mint frequently to consume swap fees	Medium	Business Logic	Resolved
7	The protocol pause feature is unavailable	Medium	Business Logic	Resolved
8	Overflow risk in the `checkPriceDifference` function	Medium	Business Logic	Resolved
9	The withdrawal order affects user benefits	Medium	Business Logic	Resolved
10	Missing validation of fund validity in the `vote` function	Medium	Business Logic	Resolved
11	Rewards continue to be distributed even when the fund shares are zero	Medium	Business Logic	Resolved
12	Users' cake tokens may be transferred incorrectly	Medium	Business Logic	Resolved
13	Possible MEV attack	Medium	Business Logic	Resolved
14	Improper rebase minimum check	Medium	Business Logic	Resolved
15	Centralization risk	Medium	Centralization	Acknowledged
16	The killGauge() did not clear votes for the next epoch	Low	Business Logic	Resolved
17	Calling `cachePeriodEarned` exactly at the start of a period fail writing to storage	Low	Business Logic	Resolved
18	User vote withdrawals do not clear `votedUsersPerPeriod`	Low	Business Logic	Resolved

19	The `setAccessHub` function affects the upgrade of the beacon contract	Low	Business Logic	Resolved
20	Inconsistency between the comment and the implementation	Informational	Business Logic	Resolved
21	Incorrect event parameters	Informational	Business Logic	Resolved
22	Gas optimization	Informational	Gas optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Potential inflation attack in FundVault	
Severity: High	Category: Business Logic
Target: - contracts/abstract/FundVault.sol	

Description

Users can mint shares by depositing base tokens, earning profit as the value of their shares increases. The share price is determined by the formula:
share price = (baseAmount × totalShares) / previousTotalValue

However, this mechanism has a vulnerability. The first depositor can mint a very small number of shares at a low cost. Later, before another user deposits, the first depositor can artificially inflate the share price by donating additional base tokens. As a result, when the second user attempts to mint shares, the inflated share price — combined with rounding down during calculation — causes them to receive fewer shares than they should, effectively leading to a loss.

contracts/abstract/FundVault.sol: L62-L73

```
function mint(
    uint256 value,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
    if (_fundInfo.shares == 0) {
        mintShares = amount;
    } else {
        uint256 totalValue = getTotalTokenBaseValue();
        require(totalValue > amount, NotEnoughBalance());
        mintShares = FullMath.mulDiv(amount, _fundInfo.shares, totalValue - amount);
    }
}
```

Recommendation

When we deploy the vault, we can mint some dead shares as the first depositor.

Status

This issue has been resolved by the team with commit 489bf05.

2. Funds will be forced to keep in the vault

Severity: High	Category: Business Logic
Target: - contracts/abstract/FundVault.sol	

Description

Users can mint shares by depositing base tokens, aiming to earn profit from swap fees generated by providing liquidity in the Pancake pool. However, liquidity will only be minted if the provided `` `lTick` `` falls within a predefined protected tick range.

The issue lies in the lack of input validation for `` `lTick` ``. A malicious user can exploit this by minting shares with a minimal amount of base tokens while specifying an `` `lTick` `` that falls outside the protected range. As a result, no liquidity is minted, but the tokens remain locked in the vault contract. This effectively drains all liquidity while leaving the vault in a dysfunctional state.

contracts/abstract/FundVault.sol: L31-L98

```
function mint(
    uint256 value,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
    ...
    mintLiquidity(lTick, deadline);
}
```

contracts/abstract/FundLiquidity.sol: L140-L147

```
function mintLiquidity(int24 tick, uint256 deadline) internal override {
    if (tick < _fundInfo.lowerTickProtection || tick > _fundInfo.upperTickProtection) {
        return;
    }
    …
}
```

Recommendation

Add one input security check for function mint.

Status

This issue has been resolved by the team with commit 489bf05.

3. Missing input validation for orderId

Severity: High	Category: Business Logic
Target: - contracts/GaugeV3.sol	

Description

In GaugeV3, the function `` `cachePeriodEarned` `` is used to calculate and cache the period duration (in seconds) for a given order.

However, there is a missing input validation for the `` `orderId` `` parameter. A malicious user can call this function with a non-existent `` `orderId` ``, causing the contract to cache period data for past intervals. Later, when the user creates the actual order with the same `` `orderId` ``, they can retroactively claim rewards for those past periods — effectively exploiting the system to earn unearned rewards.

contracts/GaugeV3: L154-L208

```
   function cachePeriodEarned(
       uint256 period,
       uint256 orderId,
       bool caching
   ) public override returns (uint256 amount) {
       IFund.Order memory order = $.fund.orders(orderId);
       if (!$.periodAmountsWritten[period][orderId]) {
           uint256 lastPeriodEndTimestamp = (period - 1) * UpTopConstant.MINTER_PERIOD;
           uint256 currentPeriodTimestamp = period * UpTopConstant.MINTER_PERIOD;
           uint256 endEarnedTimestamp = Math.max(
               order.inTime,
               lastPeriodEndTimestamp
           );
           if (endEarnedTimestamp >= currentPeriodTimestamp) {
               periodSeconds = 0;
           } else {
               periodSeconds = currentPeriodTimestamp - endEarnedTimestamp;
           }
           if (period < block.timestamp / UpTopConstant.MINTER_PERIOD && caching) {
               $.periodAmountsWritten[period][orderId] = true;
               $.periodOrderSeconds[period][orderId] = periodSeconds;
           }
       } else {
           periodSeconds = $.periodOrderSeconds[period][orderId];
       }
}
```

Recommendation

Add one input security check for orderId.

Status

This issue has been resolved by the team with commit 489bf05.

4. Incorrect `amountToSwap` calculation in the `swapToTokenBase` function

Severity: High	Category: Business Logic

Target:

- contracts/abstract/FundSwap.sol

Description

In the `swapToTokenBase` function, when `token0` is `tokenBase`, the calculation of `amountToSwap` should be `baseAmount * price / (1 << 96)`. However, the contract incorrectly uses `baseAmount * (1 << 96) / price`, which will result in an incorrect calculation of `amountToSwap`.

contracts/abstract/FundSwap.sol: L31-L98

```
function swapToTokenBase(uint256 baseAmount, int24 sTick) internal override {
    ...
    if (_fundInfo.token0 == _fundInfo.tokenBase) {
        uint256 amountToSwap = FullMath.mulDiv(baseAmount, 1 << 96, price);
        if (balance1 < amountToSwap) amountToSwap = balance1;
        ...
    } else if (_fundInfo.token1 == _fundInfo.tokenBase) {
        ...
    }
}
```

Recommendation

Calculate `amountToSwap` using the correct formula.

Status

This issue has been resolved by the team with commit 489bf05.

5. Inappropriate reward distribution design in the GaugeV3 contract

Severity: High	Category: Business Logic
Target: - contracts/GaugeV3.sol	

Description

The reward distribution design in the `GaugeV3` contract is flawed, which may cause the following issues:

1. Calling `notifyRewardAmount` in the current period will settle the order rewards for the previous period. However, the `sharesPerPeriod[period]` variable is set to the current `shares` value of the `funds` contract. This allows a malicious actor to mint a large amount of shares in the `funds` contract before `notifyRewardAmount` is called, resulting in the dilution of legitimate user rewards from the previous period.

2. The reward calculation formula is flawed. The current formula is: `tokenTotalSupplyByPeriod[period] * orderShares / sharesPerPeriod[period] * periodSeconds / UpTopConstant.MINTER_PERIOD`.

With this calculation, if an order is created partway through the previous period, it can result in incomplete reward distribution.

For example, assume each period lasts for 100 time units. User1 had already created an order before the period began, with `shares = 100`, so their `periodSeconds = 100` for this period. User2 creates an order at time 90 within the same period, with `shares = 900`, so their `periodSeconds = 10`.

At the beginning of period 101, calling `vote::distribute` will settle the rewards for period 100. Assume a total of 1000 reward tokens are allocated.

Then the total claimable rewards are: `1000 * 100 / 100 * 100 / 1000 + 1000 * 10 / 100 * 900 / 1000 = 190`.

So only 190 out of 1000 tokens can be claimed, resulting in a large portion of rewards being unallocated.

contracts/GaugeV3.sol: L122-L123

```
function notifyRewardAmount(uint256 amount) external override lock {
    ...
    $.tokenTotalSupplyByPeriod[period] += amount;  // 101
    $.sharesPerPeriod[period] = $.fund.shares();
    emit NotifyReward(msg.sender, token, amount, period);
}
```

contracts/GaugeV3.sol: L190-L194

```
function cachePeriodEarned(
    uint256 period,
    uint256 orderId,
    bool caching
) public override returns (uint256 amount) {
    ...
    amount = FullMath.mulDiv(FullMath.mulDiv(
        $.tokenTotalSupplyByPeriod[period],
        periodSeconds,
        UpTopConstant.MINTER_PERIOD
    ), order.shares, $.sharesPerPeriod[period]);
    ...
}
```

Recommendation

It is recommended to refactor the reward distribution logic.

Status

This issue has been resolved by the team with commit 489bf05.

6. Malicious users can mint frequently to consume swap fees

Severity: Medium	Category: Business Logic
Target: - contracts/abstract/FundVault.sol	

Description

In `` `FundVault` ``, users can mint shares by depositing base tokens. Each time shares are minted, the protocol automatically rebalances the capital pool. During this rebalancing process, a portion of the pool's swap fees is consumed.

A malicious user can exploit this mechanism by repeatedly minting very small amounts, triggering frequent rebalancing and continuously draining swap fees. This behavior gradually reduces the share price. Although a minting fee is in place to deter such actions, it may not be sufficient — attackers can still incur minimal cost while consuming a disproportionate amount of swap fees using tiny deposits.

contracts/abstract/FundVault.sol: L31-L99

```
function mint(
    uint256 value,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
    external nonReentrant whenNotPaused returns (uint256 orderId ,uint256 mintShares) {
       uint256 amount = FullMath.mulDiv(value, 10000 - MINT_FEE_RATE, 10000);

       balanceCapitalPool(lTick, sTick, deadline);
       mintLiquidity(lTick, deadline);
}
```

Recommendation

Consider adding one minimum base token amount for the mint function.

Status

This issue has been resolved by the team with commit 489bf05.

7. The protocol pause feature is unavailable

Severity: Medium	Category: Business Logic
Target: - contracts/abstract/FundVault.sol	

Description

The `UpTopV2Fund` contract inherits from the `PausableUpgradeable` contract, and some of its functions use the `whenNotPaused` modifier. However, since the `PausableUpgradeable` contract does not expose any public functions to pause the protocol, and `UpTopV2Fund` also does not define public `pause` or `unpause` functions, the protocol pause feature is effectively unavailable.

contracts/abstract/FundVault.sol: L33-L34

```
function mint(
    uint256 value,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
    external nonReentrant whenNotPaused returns (uint256 orderId ,uint256 mintShares) {
   ...
}
```

Recommendation

It is recommended to define `pause` and `unpause` functions in the `UpTopV2Fund` contract.

Status

This issue has been resolved by the team with commit 489bf05.

8. Overflow risk in the `checkPriceDifference` function

Severity: Medium	Category: Business Logic

Target:
- contracts/abstract/FundSwap.sol

Description

In the `checkPriceDifference` function, the protocol uses `FullMath.mulDiv` to safely calculate the price and prevent overflow risks. When the product `x * y` exceeds 256 bits, mulDiv performs the calculation using 512-bit precision. However, instead of normalizing the result by dividing by `1 << 96`, the protocol only divides by 1. This omission can cause the output of `FullMath.mulDiv(x, y, z)` to exceed the maximum value of `uint256`, leading to a transaction revert.

contracts/abstract/FundSwap.sol: L62-L63

```
function checkPriceDifference(int24 tickNew, int24 tickLast) internal view override
returns (bool) {
    ...
    uint256 priceX192 = FullMath.mulDiv(sqrtPriceX96, sqrtPriceX96, 1);
    uint256 priceX192Last = FullMath.mulDiv(sqrtPriceX96Last, sqrtPriceX96Last, 1);
    ...
}
```

Recommendation

It is recommended to divide by `1 << 96` instead of 1 when calculating the price to ensure the result does not overflow.

Status

This issue has been resolved by the team with commit 489bf05.

9. The withdrawal order affects user benefits

Severity: Medium	Category: Business Logic
Target: - contracts/abstract/FundVault.sol	

Description

In the current withdrawal design, the protocol first removes liquidity and then checks whether the existing `tokenBase` balance is sufficient to fulfill the user's withdrawal request. If sufficient, the withdrawal proceeds directly; otherwise, the protocol swaps other tokens into `tokenBase` to cover the shortfall.

This approach introduces several issues. Early withdrawers avoid swap fees by using the available `tokenBase` without triggering token swaps. However, this creates an imbalance between `tokenBase` and other tokens in the pool. As a result, the protocol's remaining funds must bear swap fees during subsequent rebalancing. Moreover, users withdrawing later may face additional swap fees if the `tokenBase` balance is insufficient, effectively shifting costs onto them.

contracts/abstract/FundVault.sol: L122-L124

```
function burn(
    uint256 orderId,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
external nonReentrant returns(uint256) {
    ...
    if (tokenBaseBalance < withdrawAmount) {
        swapToTokenBase(withdrawAmount - tokenBaseBalance, sTick);
        withdrawAmount = IERC20(_fundInfo.tokenBase).balanceOf(address(this));
    }
    ...
    if (_fundInfo.shares > 0) {
        balanceCapitalPool(lTick, sTick, deadline);
        mintLiquidity(lTick, deadline);
    }
    return withdrawAmount;
}
```

Recommendation

It is recommended that regardless of whether `tokenBase` is sufficient, the user's shares should be proportionally calculated into corresponding amounts of `tokenBase` and the other token, then the other token is swapped into `tokenBase` and sent to the user.

Status

This issue has been resolved by the team with commit 489bf05.

10. Missing validation of fund validity in the `vote` function

Severity: Medium	Category: Business Logic
Target: - contracts/Voter.sol	

Description

In the `vote` function, the protocol lacks validation to ensure that the fund being voted for is valid, which may lead to several issues:

1. Malicious actors can cast votes for invalid addresses, thereby diluting rewards intended for legitimate pools and hindering the full distribution of reward tokens.
2. Malicious actors can vote for funds that have already been removed. Since removing a fund does not revoke the token approval granted to the Gauge contract, and the `distribute` function does not verify whether the Gauge contract is still active, rewards may continue to be distributed to obsolete or removed contracts.

contracts/Voter.sol: L293-L295

```
function vote(address user, address[] calldata _funds, uint256[] calldata _weights)
external {
    ...
    address[] memory votedPools = new address[](_funds.length);
    for (uint256 i = 0; i < _funds.length; ++i) {
        votedPools[i] = _funds[i];
    }

    _vote(user, votedPools, _weights);
}
```

Recommendation

It is recommended to skip the calculation for any pool that contains invalid addresses.

Status

This issue has been resolved by the team with commit 489bf05.

11. Rewards continue to be distributed even when the fund shares are zero

Severity: Medium	Category: Business Logic

Target:

- contracts/Voter.sol

Description

In extreme cases, when the fund contract has no shares, the `_distribute` function will still send rewards to the Gauge, and these rewards will become unclaimable by anyone. This will result in reward tokens being locked in the `GaugeV3` contract.

contracts/Voter.sol: L401-L404

```
function _distribute(address _gauge, uint256 _claimable, uint256 _period) internal {
    ...
    if (
        _xUpTopClaimable / DURATION == 0
            || (_xUpTopClaimable < IGaugeV3(_gauge).left())
    ) {
        canDistribute = false;
    }
    ...
}
```

Recommendation

It is recommended to skip reward distribution when shares are equal to zero.

Status

This issue has been resolved by the team with commit 489bf05.

12. Users’ cake tokens may be transferred incorrectly

Severity: Medium	Category: Business Logic
Target: - contracts/abstract/FundLiquidity.sol	

Description

When minting or burning shares, the protocol collects all assets and rebalances the pool. If the position NFT is staked in the MasterChef contract, the staking reward token CAKE is transferred to the `feeVault`.

The issue arises when CAKE is also one of the tokens in the pool. In this case, users’ CAKE tokens may remain stored in the Vault, but the protocol transfers all CAKE tokens to the `feeVault` indiscriminately. This results in users’ CAKE balances being incorrectly moved, potentially causing loss or misallocation of funds.

contracts/abstract/FundingLiquidity.sol: L21-L38

```
function collectAssets(uint256 deadline) internal override {
    if (_fundInfo.tokenID == 0) return;

    if (_fundInfo.isStaked) {
        IMasterChefV3(masterChef).withdraw(
            _fundInfo.tokenID,
            address(this)
        );
        _fundInfo.isStaked = false;
        uint256 cakeBalance =
IERC20(IMasterChefV3(masterChef).CAKE()).balanceOf(address(this));
        if (cakeBalance > 0) {
            IERC20(IMasterChefV3(masterChef).CAKE()).transfer(feeVault, cakeBalance);
            emit CakeHarvested(cakeBalance);
        }
    }
}
```

Recommendation

The function `withdraw` in masterChef will return the reward amount. We should check this return value and return the actual reward CAKE amount to the fee vault.

Status

This issue has been resolved by the team with commit 489bf05.

13. Possible MEV attack

Severity: Medium	Category: Business Logic
Target: - contracts/abstract/FundLiquidity.sol	

Description

When minting shares, the protocol aggregates all assets and simulates converting all tokens into the base token based on the current liquidity pool price.

The issue is that malicious users can manipulate the liquidity pool price, thereby artificially influencing the calculated total base token value. This price manipulation poses a risk of inaccurate valuations and potential exploitation during the minting process.

For example:

1. We deposit 1000 USDC to mint liquidities in the USDC/WETH pool between 2660 and 3502.Current price is around 3502 USDC/ETH.

2. Malicious users can manipulate the pool's price to increase WETH price. If our base token is WETH, we need to convert 1000 USDC to WETH via this manipulated WETH price.

contracts/abstract/FundingLiquidity.sol: L21-L38

```
function mint(
    uint256 value,
    int24 lTick,
    int24 sTick,
       uint256 deadline,
       address recipient)
       external nonReentrant whenNotPaused returns (uint256 orderId ,uint256 mintShares)
{
       if (_fundInfo.shares == 0) {
           mintShares = amount;
       } else {
           uint256 totalValue = getTotalTokenBaseValue();
           require(totalValue > amount, NotEnoughBalance());
           mintShares = FullMath.mulDiv(amount, _fundInfo.shares, totalValue - amount);
       }
}
```

Recommendation

Consider adding one Twap price check to avoid the price manipulation.

Status

This issue has been resolved by the team with commit 489bf05.

14. Improper rebase minimum check

Severity: Medium	Category: Business Logic

Target:
- contracts/abstract/FundLiquidity.sol

Description

In a new distribution period, the protocol may rebase and distribute pending penalties as rewards. Rebasing is allowed only if the pending base exceeds the defined threshold, `` `BASIS` ``.

The issue is that the minimum threshold of 10,000 (`` `BASIS` ``) is insufficient. This can result in rewards that are too small to be claimed due to rounding errors, effectively causing some rewards to remain undistributed.

For example:

1. pendingRebase = 10_000
2. rewardsRate = 5
3. Assume totalSupply = 1000 * 1e18. If delta time is less than 200s, all rewards will be rounded down to 0.

contracts/abstract/FundingLiquidity.sol: L21-L38

```
function rebase() external whenNotPaused {
    uint256 period = VOTER.getPeriod();
    if (
        period > lastDistributedPeriod &&
@>        pendingRebase >= BASIS // 10_000
    ) {

        IVoteModule(VOTE_MODULE).notifyRewardAmount(_temp);
    }
}
```

contracts/VoteModule.sol: L170-L205

```
   function notifyRewardAmount(
       uint256 amount
   ) external updateReward(address(0)) nonReentrant {
       if (block.timestamp >= periodFinish) {
@>           rewardRate = amount / duration;
       }
}
```

Recommendation

Consider increasing the minimum value to rebase, e.g 1e18.

Status

This issue has been resolved by the team with commit 489bf05.

15. Centralization risk

Severity: Medium	Category: Centralization

Target:
- contracts/Voter.sol
- contracts/abstract/FundBalancer.sol

Description

In UpTopV2Fund contracts, there exists some privileged roles, e.g. `Governance_role`, `BALANCER_ROLE`, etc. These roles have the authority to execute some key functions such as `setGovernor`, `killGauge` and `performUpKeep`, etc.

If these roles' private keys are compromised, an attacker could trigger these functions to block key functions.

contracts/Voter.sol: L21-L38

```
function setGovernor(address _governor) external onlyGovernance {
    VoterStorage.VoterState storage $ = VoterStorage.getStorage();

    if ($.governor != _governor) {
        $.governor = _governor;
        emit IVoter.NewGovernor(msg.sender, _governor);
    }
}
function killGauge(address _gauge) public onlyGovernance { // 移除 gauge
    VoterStorage.VoterState storage $ = VoterStorage.getStorage();
    address fund = $.fundForGauge[_gauge];
}
```

contracts/abstract/FundBalancer.sol: L45-L48

```
function performUpkeep(bytes calldata data) external override nonReentrant whenNotPaused
onlyRole(BALANCER_ROLE) {
    (int24 lTick, int24 sTick, uint256 deadline) = abi.decode(data, (int24, int24,
uint256));
    autoRebalance(lTick, sTick, deadline);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

16. The killGauge() did not clear votes for the next epoch

Severity: Low	Category: Business Logic
Target: - contracts/Voter.sol	

Description

The `killGauge` function removes the fund and Gauge, but it does not clear users' votes for the fund in the next epoch.

If users do not withdraw their votes, it may result in the removed pool diluting the rewards of the active pools.

contracts/Voter.sol: L469-L475

```
function killGauge(address _gauge) public onlyGovernance {
    ...
    $.gauges.remove(_gauge);
    $.funds.remove(fund);
    delete $.gaugeForFund[fund];
    delete $.fundForGauge[_gauge];
    delete $.lastDistro[_gauge];
    emit IVoter.GaugeKilled(_gauge);
}
```

Recommendation

Consider clearing the votes for the fund to be removed within the `killGauge` function.

Status

This issue has been resolved by the team with commit 489bf05.

17. Calling `cachePeriodEarned` exactly at the start of a period fail writing to storage

Severity: Low	Category: Business Logic

Target:

- contracts/GaugeV3.sol

Description

In the `cachePeriodEarned` function, rewards are calculated for the period between `period - 1` and `period`.

However, when `period` equals `block.timestamp / UpTopConstant.MINTER_PERIOD`, the `periodSeconds` value becomes static and no longer updates. As a result, although the reward is computed, it is not written to storage, potentially causing inconsistencies in reward tracking.

contracts/GaugeV3.sol: L154-L208

```
function cachePeriodEarned(
    uint256 period,
    uint256 orderId,
    bool caching
) public override returns (uint256 amount) {
    ...
    if (period < block.timestamp / UpTopConstant.MINTER_PERIOD && caching) {
        $.periodAmountsWritten[period][orderId] = true;
        $.periodOrderSeconds[period][orderId] = periodSeconds;
    }
    ...
}
```

Recommendation

It is recommended to use `<=` instead of `<`.

Status

This issue has been resolved by the team with commit 489bf05.

18. User vote withdrawals do not clear `votedUsersPerPeriod`

Severity: Low	Category: Business Logic
Target: - contracts/Voter.sol	

Description

When a user votes for the next period, the user is added to `votedUsersPerPeriod[nextPeriod]`.

The user can call the `reset` function to clear their vote for the next period, but the `_reset` function does not remove the user from `votedUsersPerPeriod[nextPeriod]`.

contracts/Voter.sol: L226-L230

```
function _reset(address user) internal {
    ...
    if (votingPower > 0) {
        ...
        /// @dev reduce the overall vote power casted
        $.totalVotesPerPeriod[nextPeriod] -= votingPower;
        /// @dev wipe the mappings
        delete $.userVotingPowerPerPeriod[user][nextPeriod];
        delete $.userVotedFundsPerPeriod[user][nextPeriod];
    }
}
```

Recommendation

It is recommended to remove the user from `votedUsersPerPeriod[nextPeriod]` in the `reset` function.

Status

This issue has been resolved by the team with commit 489bf05.

19. The `setAccessHub` function affects the upgrade of the beacon contract

Severity: Low	Category: Business Logic

Target:
- contracts/UpTopV2FundFactory.sol

Description

In the initial state, the owner of the beacon contract is `UpTopV2FundFactory`, and the `UpTopV2FundFactory` contract has an `upgradeImplementation` function to update the beacon contract's implementation.

However, when calling the `setAccessHub` function to transfer the `accessHub`, it also transfers the ownership of the beacon contract to the new `accessHub`.

This is unnecessary and causes the `upgradeImplementation` function to become ineffective.

Furthermore, if the new `accessHub` does not transfer the beacon contract's ownership back to `upgradeImplementation`, subsequent calls to `setAccessHub` will revert due to failure in transferring ownership of the beacon contract.

contracts/UpTopV2FundFactory.sol: L118-L135

```
function upgradeImplementation(address newImplementation) external {
    require(msg.sender == accessHub, NOT_AUTHORIZED(msg.sender));
    beacon.upgradeTo(newImplementation);
    emit Upgraded(newImplementation);
}

function setAccessHub(address newAccessHub) external {
    require(msg.sender == accessHub, NOT_AUTHORIZED(msg.sender));
    require(newAccessHub != address(0), ZERO_ADDRESS());
    beacon.transferOwnership(newAccessHub);
    accessHub = newAccessHub;
    emit AccessHubChanged(newAccessHub, accessHub);
}
```

Recommendation

It is recommended not to transfer the ownership of the beacon contract when calling the `setAccessHub` function.

Status

This issue has been resolved by the team with commit 489bf05.

2.3 Informational Findings

20. Inconsistency between the comment and the implementation	
Severity: Informational	Category: Business Logic
Target: contracts/abstract/FundVault.sol	

Description

In the `burn` function, the comment mentions that a fee should be charged to users to prevent frequent redemptions that could lead to swap fee losses.

However, in the actual implementation, no redemption fee is charged to the user; only a fee is applied to the profit portion.

contracts/abstract/FundVault.sol: L118-L135

```
function burn(
    uint256 orderId,
    int24 lTick,
    int24 sTick,
    uint256 deadline,
    address recipient)
external nonReentrant returns(uint256) {
    ...
    uint256 withdrawAmount = FullMath.mulDiv(totalValue, order.shares,
_fundInfo.shares);
    ...
}
```

Recommendation

It is recommended to update either the comment or the implementation to keep them consistent.

Status

This issue has been resolved by the team with commit 489bf05.

21. Incorrect event parameters

Severity: Informational	Category: Business Logic
Target: contracts/abstract/FundVault.sol	

Description

In the `exitVest` function, when the user's vesting duration exceeds `MIN_VEST` but has not yet reached the unlock time, `UPTOP` is released based on the vesting duration. The actual amount of `UPTOP` released is `exitedAmount`, but the `ExitVesting` event still uses `_amount`.

The `setExemption` function is used to set the `exempt` mapping, and the `setExemptionTo` function is used to set the `exemptTo` mapping. These two different operations should use different events, but both functions use the `Exemption` event.
contracts/abstract/FundVault.sol: L264-L265

```
function exitVest(uint256 _vestID) external whenNotPaused {
    ...
    else {
        ...
        UPTOP.transfer(msg.sender, exitedAmount);
        emit ExitVesting(msg.sender, _vestID, _amount);
    }
}
```

contracts/abstract/FundVault.sol: L301-L332

```
function setExemptionTo(
    address[] calldata _exemptee,
    bool[] calldata _exempt
) external onlyGovernance {
    ...
    for (uint256 i = 0; i < _exempt.length; ++i) {
        ...
        emit Exemption(_exemptee[i], _exempt[i], success);
    }
}
```

Recommendation

It is recommended to use the correct event parameters in the `ExitVesting` event and to use different events to distinguish between the `setExemption` and `setExemptionTo` operations.

Status

This issue has been resolved by the team with commit 489bf05.

22. Gas optimization

Severity: Informational	Category: Gas Optimization
Target: contracts/abstract/FundLiquidity.sol	

Description

Every time liquidity is provided by interacting with the fund contract, if staking is allowed, `setApprovalForAll` is called to authorize `masterChef`. However, without revoking the authorization, this function only needs to be called once. Currently, it causes unnecessary gas waste.

contracts/abstract/FundLiquidity.sol: L118-L135

```
function mintLiquidity(int24 tick, uint256 deadline) internal override {
    ...
    if (_fundInfo.isStakable) {
    INonfungiblePositionManager(positionManager).setApprovalForAll(address(masterChef),
true);
        INonfungiblePositionManager(positionManager).safeTransferFrom(
            address(this),
            address(masterChef),
            newTokenID
        );
        _fundInfo.isStaked = true;
    }
}
```

Recommendation

It is recommended to call `setApprovalForAll` only once when authorization has not yet been granted.

Status

This issue has been resolved by the team with commit 489bf05.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit e14d32e:

File	SHA-1 hash
UpTopV2FundFactory.sol	c8a6273d40276ae3ef0d14f1de2fb7b420df5fa1
FundLiquidity.sol	71a6e2519e997ae0ae31abed655363376fe6425d
FundSwap.sol	8aaaedb8b3f632f5a91265d4efa21f10f3ab6e69
FundBase.sol	1f8a7e16e6ac1e485da8f823476fc1bad97b4222
FundBalancer.sol	7caef70deed376f5f89ecca272bb9f2580deccbe
FundVault.sol	d53ea4b446fdb610a7577f0c185117ada49ad094
GaugeV3Factory.sol	418f47704432d2fae969978d986bed19bfbcedf1
UpTop.sol	b3d88d0f9bffcd63267c0c7922dc2ff2e7b225a8
UpTopV2Fund.sol	b64dbe3fe13e53c580510fa5477bd4f4bf1c1e30
UpTopConstant.sol	4307987637108ac6baa9ae4b73d0c11dce380e09
FundStorage.sol	08259ce1faa7ddd5fe188366ed7a891e51ffb8af
Errors.sol	e194370b4a1c3ee2f5572af6dbec0f59a81ebe79
AMMLibrary.sol	eb1ab4d3582051a2cdc608bdd7ff386c918e1619
VoterStorage.sol	672017ff79647f9c36f2b05f8694f63cf4016e11
GaugeV3Storage.sol	7f5a11bce812f3aae400c1d6304955f333f44df8
GaugeV3.sol	4057a36228f69e2591fe5228fbf524044571c5f9
Voter.sol	1e715e7c7ece75e10569fe7a5bbdb262f3a7c95d
xUpTop.sol	f745bf5898bfaf813641be6902ff7e8c520025b2
VoteModule.sol	2137a1faf88f8a97075785dbd3a084b29d878417
UpTopQuerier.sol	433277daa383bba9d8acef656c02169f9c374a01
Minter.sol	49c5bda0b80db7fe2c18232dd01848b489d09f5a
IFund.sol	49d5db073b147ef573c16382ec6f6ca8d15f0b31
IERC20Extended.sol	a8c0490bc1a798df0a84ec61f347aaa1b577f9c9

IGaugeV3.sol	69693ede15ac123fd83ad1f55b65b236eabeb895
IGaugeV3Factory.sol	7970cff5cbedfcd5418c17b029f2660497432d24
IUniversalRouter.sol	b37998047d3f8e970589cc8a23310e1054082b9b
IAllowanceTransfer.sol	2cfe719cec6fb5032fdf025ecb406e9ab09d96e3
IVoteModule.sol	44059d6c6bbc4f77703cc9a5aa09633fa4ffc075
IXUpTop.sol	fd085d7234182790cd55a213171b8a8b37822143
IVoter.sol	aecd3ad35f9a981ecddaadce3b18ab9a450fc668
IMinter.sol	d408f9d031c27b8fe19e6255c7e98ed1389ad922
IMasterChefV3.sol	a52a74564e75548b023a7436baa2657feeb1053d
IEIP712.sol	1379988147c72e29aec4b8032401aa8f36ee15d2
IAccessHub.sol	e1c9b8b6dafb998a491bb039303dfbb58b393e10
IFundFactory.sol	d2926a80d0b5df4cc652233d8691e8debc3237d3

This audit also covers some new features in commit 84f5669.