Alpha Contracts v2.1
Security Review

Cantina Managed review by:

Cergyk, Lead Security Researcher
Kaden, Security Researcher

June 27, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Charm's mission is to create liquid markets for every asset, and Alpha Vaults is Charm's flagship product. Alpha Vaults is a binary smart contract system for decentralized liquidity management - it creates liquidity vaults for any Uniswap V3 pool, and contains everything that's needed to manage the liquidity within a pool.

Alpha Vaults' core contracts are alpha-contracts-v2.1 - it defines the logic for vault creation, the infrastructure to manage deposits and withdrawals, and the strategy to manage liquidity. The contracts are the next version of alpha-vaults-v2-contracts, and contain additional features to enhance the yield, liquidity, and composability of the vaults created using Alpha Vaults. From Jun 9th to Jun 12th the Cantina team conducted a review of alpha-contracts-v2.1 on commit hash e5dad18a. The team identified a total of 15 issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	6	4	2
Gas Optimizations	2	1	1
Informational	6	2	4
Total	**15**	**8**	**7**

The Cantina Managed team reviewed Charm's alpha-contracts-v2.1 holistically on commit hash 420c22b0 (which also includes a fix related to EVM deployment limits, addressing an issue in the error code presentation without impacting the contract logic) and concluded that all issues were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Medium Risk

3.1.1 Insufficient factory check during vault creation

Severity: Medium Risk

Context: AlphaProVaultFactory.sol#L46-L47

Description: When creating a vault in `AlphaProVaultFactory`, it is checked that the contract has a field factory which is whitelisted:

- AlphaProVaultFactory.sol#L46-L47:

```
/**
* @notice Create a new Alpha Pro Vault
* @param params InitializeParams Underlying Uniswap V3 pool address
*/
function createVault(VaultParams calldata params) external returns (address vaultAddress) {
    address poolFactory = IUniswapV3Pool(params.pool).factory();
    require(allowedFactories[poolFactory], "allowedFactories");
```

Unfortunately, anybody can deploy a contract which has a field factory which is whitelisted by governance; So this means this check is not sufficient to prove that the pool has been created by the factory and is not malicious;

Recommendation: One should also check that the factory "knows" the pool by checking that the user provided address is indeed the address for the pool with (`token0`, `token1`, `fee`);

- AlphaProVaultFactory.sol#L46-L47:

```
  /**
   * @notice Create a new Alpha Pro Vault
   * @param params InitializeParams Underlying Uniswap V3 pool address
   */
  function createVault(VaultParams calldata params) external returns (address vaultAddress) {
      IUniswapV3Pool pool = IUniswapV3Pool(params.pool);
      address poolFactory = pool.factory();
      require(allowedFactories[poolFactory], "allowedFactories");
+     require(params.pool == IUniswapV3PoolFactory(poolFactory).getPool(pool.token0(), pool.token1(),
↪  pool.fee());
```

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Fee rounding consistency in `getPositionAmounts`

Severity: Low Risk

Context: AlphaProVault.sol#L589-L590

Description: To evaluate shares to mint for given deposit amounts, `AlphaProVault::deposit` uses a `getPositionAmounts(bool roundUp)` function adapted from UniswapV3 library. It enables to compute these amounts with rounding up if necessary. In the case of a roundup however, division related to the evaluation of fees is not rounded up:

- AlphaProVault.sol#L589-L590:

```
// Subtract protocol and manager fees
uint256 oneMinusFee = uint256(1e6) - protocolFee - managerFee;

//@audit Amount left after fee is rounded down here
amount0 = amount0 + ((uint256(tokensOwed0) * oneMinusFee) / 1e6);
amount1 = amount1 + ((uint256(tokensOwed1) * oneMinusFee) / 1e6);
```

As a result the amount returned here due to the vault can be rounded down overall.

Since the total amount is later used to evaluate the amount of shares minted to the user, it is crucial to have it rounded up overall:

- AlphaProVault.sol#L272-L283:

```
} else {
    uint256 cross0 = amount0Desired * total1;
    uint256 cross1 = amount1Desired * total0;
    uint256 cross = cross0 > cross1 ? cross1 : cross0;
    require(cross > 0, "cross");

    // Round up amounts
    amount0 = ((cross - 1) / (total1)) + 1;
    amount1 = ((cross - 1) / (total0)) + 1;
    //@audit shares evaluated here need to have a rounded up total0 and total1, to be rounded down
    ↪  overall
    shares = ((cross * totalSupply) / total0) / total1;
}
```

Recommendation: Use the same formula to compute pending fees as the one used in `_burnAndCollect` for `getPositionAmounts`:

```
  (amount0, amount1) = _amountsForLiquidity(tickLower, tickUpper, liquidity, roundUp);

  // Subtract protocol and manager fees
- uint256 oneMinusFee = uint256(1e6) - protocolFee - managerFee;
+ uint256 _managerFee = managerFee;
+ uint256 _protocolFee = protocolFee;
+ uint256 protocolFee0 = (_protocolFee * tokensOwed0) / 1e6;
+ uint256 protocolFee1 = (_protocolFee * tokensOwed1) / 1e6;
+ uint256 managerFee0 = (_managerFee * tokensOwed0) / 1e6;
+ uint256 managerFee1 = (_managerFee * tokensOwed1) / 1e6;

- amount0 = amount0 + ((uint256(tokensOwed0) * oneMinusFee) / 1e6);
- amount1 = amount1 + ((uint256(tokensOwed1) * oneMinusFee) / 1e6);
+ amount0 = amount0 + (uint256(tokensOwed0) - protocolFee0 - managerFee0);
+ amount1 = amount1 + (uint256(tokensOwed1) - protocolFee1 - managerFee1);
```

Charm: Fixed in commit b398a433.

Cantina Managed: Fix verified.

3.2.2 Protocol fee could be fetched from factory after rebalance

Severity: Low Risk

Context: AlphaProVault.sol#L688-L693

Description: Currently in `AlphaProVault`, there is a dedicated setter to modify protocol fee, whereas one is also present in `AlphaProVaultFactory` to set protocol fee for vaults created in the future:

- AlphaProVault.sol#L684-L693:

```
/**
 * @notice Change the protocol fee charged on pool fees earned from
 * Uniswap, expressed as multiple of 1e-6. Fee is hard capped at 25%.
 */
function setProtocolFee(uint24 _pendingProtocolFee) external {
    require(msg.sender == factory.governance(), "governance");
    require(_pendingProtocolFee <= 25e4, "protocolFee must be <= 250000");
    pendingProtocolFee = _pendingProtocolFee;
    emit UpdateProtocolFee(_pendingProtocolFee);
}
```

- AlphaProVaultFactory.sol#L67-L72:

```
/**
 * @notice Change the protocol fee charged on pool fees earned from
 * Uniswap, expressed as multiple of 1e-6. Fee is hard capped at 25%.
 */
function setProtocolFee(uint24 _protocolFee) external onlyGovernance {
    require(_protocolFee <= 25e4, "protocolFee must be <= 250000");
    protocolFee = _protocolFee;
    emit UpdateProtocolFee(_protocolFee);
}
```

A more practical approach would be to pull the protocol fee from factory right after rebalance, as this would avoid the need to call on every vault created historically to update protocol fee.

Recommendation: Fetch protocol fee from factory at the end of rebalance, and remove dedicated setter and `pendingProtocolFee`:

- AlphaProVault.sol#L433-L441:

```
  // Update fee only at each rebalance, so that if fee is increased
  // it won't be applied retroactively to current open positions
- uint24 _protocolFee = protocolFee = pendingProtocolFee;
+ uint24 _protocolFee = protocolFee = factory.protocolFee();
  // Manager + protocol fee must be <= 100%
  if (pendingManagerFee + _protocolFee <= HUNDRED_PERCENT) {
      managerFee = pendingManagerFee;
  } else {
      managerFee = HUNDRED_PERCENT - _protocolFee;
  }
```

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.2.3 Manager can allow himself to rebalance unrestricted

Severity: Low Risk

Context: AlphaProVault.sol#L740-L743

Description: There are checks implemented in `checkCanRebalance` to rate limit the number of rebalances that are made, to avoid slow leakage of funds through successive rebalancings;

- AlphaProVault.sol#L444-L463:

```
function checkCanRebalance() public view override {
    checkPriceNearTwap();
    uint256 _lastTimestamp = lastTimestamp;

    // check enough time has passed
    require(block.timestamp >= (_lastTimestamp + period), "PE");

    // check price has moved enough
    (, int24 tick,,,,,) = pool.slot0();
    int24 tickMove = tick > lastTick ? tick - lastTick : lastTick - tick;
    require(_lastTimestamp == 0 || tickMove >= minTickMove, "TM");

    // check price not too close to boundary
    int24 maxThreshold = baseThreshold > limitThreshold ? baseThreshold : limitThreshold;
    require(
        tick >= TickMath.MIN_TICK + maxThreshold + tickSpacing
            && tick <= TickMath.MAX_TICK - maxThreshold - tickSpacing,
        "PB"
    );
}
```

Unfortunately all of these checks can be lifted by the manager by resetting the values:

- AlphaProVault.sol#L740-L761:

```
function setPeriod(uint32 _period) external onlyManager {
    period = _period;
    emit UpdatePeriod(_period);
}

function setMinTickMove(int24 _minTickMove) external onlyManager {
    require(_minTickMove >= 0, "minTickMove must be >= 0");
    minTickMove = _minTickMove;
    emit UpdateMinTickMove(_minTickMove);
}

function setMaxTwapDeviation(int24 _maxTwapDeviation) external onlyManager {
    require(_maxTwapDeviation >= 0, "maxTwapDeviation must be >= 0");
    maxTwapDeviation = _maxTwapDeviation;
    emit UpdateMaxTwapDeviation(_maxTwapDeviation);
}

function setTwapDuration(uint32 _twapDuration) external onlyManager {
    require(_twapDuration > 0, "twapDuration must be > 0");
    twapDuration = _twapDuration;
    emit UpdateTwapDuration(_twapDuration);
}
```

Recommendation: Multiple mitigations are possible:

- Implement a delay for setting these important values.
- Implement a minimum non-zero delay between rebalances.

Charm: Acknowledged. Mismanagement such as the one described in this finding is an example of manager risks - this is highlighted in Charm's docs, along with other risks for liquidity providers.

Cantina Managed: Acknowledged.

3.2.4 Vault can be drained if `baseThreshold` is equal to `wideThreshold`

Severity: Low Risk

Context: AlphaProVault.sol#L312-L314

Description: An implicit assumption in `AlphaProVault` is that `wide` and `base` position are independent, however these can be set to be equal by the manager, causing miscalculations during vault calls. Due to some implementation specifics, the impacts on the methods `deposit`, `rebalance` and `withdraw` are different. Let's analyse these impacts when adding minting 100% of current supply, and then withdrawing all of minted shares (50% of `totalSupply`). For simplicity we assume zero liquidity for the `limit` range and zero balances for tokens directly in vault, which are not impacted by this:

1. Deposits: Since liquidities are prefetched, and if we assume L of current liquidity in `wide/base` range, we will mint for 2*L:
   - Deposits are overestimated by a factor of 2x:
     - AlphaProVault.sol#L231-L239:

```
    for (uint256 i = 0; i < 3; i++) {
        int24 tickLower = positions[i][0];
        int24 tickUpper = positions[i][1];

        if (liquidities[i] > 0) {
            uint128 liquidityShare = uint128(Math.mulDiv(uint256(liquidities[i]), shares,
            ↪  totalSupply()));
            _mintLiquidity(tickLower, tickUpper, liquidityShare);
        }
    }
```

2. Rebalance: Since liquidities are prefetched, and if we assume L of current liquidity in wide/base range, we will attempt to burn for 2*L:
   - Rebalances will fail when L != 0.
     - AlphaProVault.sol#L361-L371:

```
        // Withdraw all current liquidity from Uniswap pool
        {
            int24 _wideLower = wideLower;
            int24 _wideUpper = wideUpper;
            (uint128 wideLiquidity,,,,) = _position(_wideLower, _wideUpper);
            (uint128 baseLiquidity,,,,) = _position(baseLower, baseUpper);
            (uint128 limitLiquidity,,,,) = _position(limitLower, limitUpper);
            _burnAndCollect(_wideLower, _wideUpper, wideLiquidity);
            _burnAndCollect(baseLower, baseUpper, baseLiquidity);
            _burnAndCollect(limitLower, limitUpper, limitLiquidity);
        }
```

During mint however, the vault will only attempt to deposit all of the available tokens, so deposited amounts would be unchanged here.

3. Withdraw: Liquidities are not prefetched. This means that when trying to withdraw 50% of `totalSupply()`:
    - We will first withdraw 50% of the sum `wide/base` when burn/collecting from wide range.
    - Then again 50% of the remainder when burn/collecting from base range.

    This means we are withdrawing 75% of the liquidity in the `wide/base` range instead of 50%.

    - AlphaProVault.sol#L311-L314:

```
        // Withdraw proportion of liquidity from Uniswap pool
        //@audit if shares/totalSupply = 50%, we withdraw 50% of position liq here
        (uint256 wideAmount0, uint256 wideAmount1) = _burnLiquidityShare(wideLower, wideUpper, shares,
        ↪  totalSupply);
        //@audit if shares/totalSupply = 50%, we withdraw 50% of remainder of position liq here, so 25%
        ↪  of initial position liq
        (uint256 baseAmount0, uint256 baseAmount1) = _burnLiquidityShare(baseLower, baseUpper, shares,
        ↪  totalSupply);
        (uint256 limitAmount0, uint256 limitAmount1) = _burnLiquidityShare(limitLower, limitUpper,
        ↪  shares, totalSupply);
```

Impact Analysis: Overall we notice that in this situation where `baseThreshold == wideThreshold`, making a deposit for 100% of current shares and then withdrawing it all is profitable, enabling the drainage of the vault.

Recommendation: Setting `baseThreshold == wideThreshold` should be protected against in the setters and during vault creation:

- AlphaProVault.sol#L716-L738:

```
  function setBaseThreshold(int24 _baseThreshold) external onlyManager {
      _checkThreshold(_baseThreshold, tickSpacing);
+     require(wideThreshold != _baseThreshold);
      baseThreshold = _baseThreshold;
      emit UpdateBaseThreshold(_baseThreshold);
  }

  function setLimitThreshold(int24 _limitThreshold) external onlyManager {
      _checkThreshold(_limitThreshold, tickSpacing);
      limitThreshold = _limitThreshold;
      emit UpdateLimitThreshold(_limitThreshold);
  }

  function setWideRangeWeight(uint24 _wideRangeWeight) external onlyManager {
      require(_wideRangeWeight <= 1e6, "wideRangeWeight must be <= 1e6");
      wideRangeWeight = _wideRangeWeight;
      emit UpdateWideRangeWeight(_wideRangeWeight);
  }

  function setWideThreshold(int24 _wideThreshold) external onlyManager {
      _checkThreshold(_wideThreshold, tickSpacing);
+     require(baseThreshold != _wideThreshold);
      wideThreshold = _wideThreshold;
      emit UpdateWideThreshold(_wideThreshold);
  }
```

- AlphaProVault.sol#L165-L175:

```
    factory = AlphaProVaultFactory(_factory);
    pendingProtocolFee = factory.protocolFee();

    _checkThreshold(_params.baseThreshold, _tickSpacing);
    _checkThreshold(_params.limitThreshold, _tickSpacing);
    _checkThreshold(_params.wideThreshold, _tickSpacing);
    require(_params.wideRangeWeight <= 1e6, "wideRangeWeight must be <= 1e6");
    require(_params.minTickMove >= 0, "minTickMove must be >= 0");
    require(_params.maxTwapDeviation >= 0, "maxTwapDeviation must be >= 0");
    require(_params.twapDuration > 0, "twapDuration must be > 0");
    require(_params.managerFee <= HUNDRED_PERCENT, "managerFee must be <= 1000000");
+   require(_params.baseThreshold != _params.wideThreshold, "baseThreshold cannot be equal to
↪   wideThreshold");
```

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.2.5 Rounding error during direct deposit may Dos deposits

Severity: Low Risk

Context: AlphaProVault.sol#L228-L239

Description: This issue was self identified by the Charm Finance team and discussed with the Cantina Managed team. To deposit on `AlphaProVault`, users provide `amount0Desired` and `amount1Desired` of tokens which is the max amount of either token which will be pulled from the user. One of these amounts is guaranteed to be pulled in the general case.

`amount0` and `amount1` pulled amount calculation in `_calcAmountsAndShares`:

- AlphaProVault.sol#L273-L282:

```
uint256 cross0 = amount0Desired * total1;
uint256 cross1 = amount1Desired * total0;
uint256 cross = cross0 > cross1 ? cross1 : cross0;
require(cross > 0, "cross");

// Round up amounts
//@audit either amount0 == amount0Desired or amount1 == amount1Desired
amount0 = ((cross - 1) / (total1)) + 1;
amount1 = ((cross - 1) / (total0)) + 1;
shares = ((cross * totalSupply) / total0) / total1;
```

However since the vault ends up minting in 3 different ranges after the deposit made by the user, the pulled amount from the vault is rounded up 3 times and may exceed the amount pulled from the user and would revert as a result:

- AlphaProVault.sol#L231-L240:

```
for (uint256 i = 0; i < 3; i++) {
    int24 tickLower = positions[i][0];
    int24 tickUpper = positions[i][1];

    if (liquidities[i] > 0) {
        uint128 liquidityShare = uint128(Math.mulDiv(uint256(liquidities[i]), shares, totalSupply()));
        //@audit shares has been computed accordingly to amount0Desired or amount1Desired
        //@audit since the amounts pulled below are rounded up 3 times, they end up exceeding
        ↪  amount0Desired or amount1Desired by a few wei
        _mintLiquidity(tickLower, tickUpper, liquidityShare);
    }
}
```

Recommendation: The following solution was proposed by the Charm Finance team and reviewed as adequate by Cantina:

- AlphaProVault.sol#L231-L240:

```
// Pull in tokens from sender
if (amount0 > 0) token0.safeTransferFrom(msg.sender, address(this), amount0);
if (amount1 > 0) token1.safeTransferFrom(msg.sender, address(this), amount1);

uint256 _totalSupply = totalSupply();
(uint160 sqrtRatioX96,,,,,) = pool.slot0();
for (uint256 i = 0; i < 3; i++) {
    int24 tickLower = positions[i][0];
    int24 tickUpper = positions[i][1];
    if (liquidities[i] > 0) {
        uint128 liquidityToMint = uint128(Math.mulDiv(uint256(liquidities[i]), shares, _totalSupply));
        uint128 liquidityFromAmounts =
            _liquidityForAmounts(tickLower, tickUpper, amount0, amount1, sqrtRatioX96);
        liquidityToMint = liquidityToMint > liquidityFromAmounts ? liquidityFromAmounts :
        ↪  liquidityToMint;

        (uint256 mintAmount0, uint256 mintAmount1) = _mintLiquidity(tickLower, tickUpper,
        ↪  liquidityToMint);
        amount0 -= mintAmount0;
        amount1 -= mintAmount1;
    }
}
```

The `getTotalAmounts(bool roundUp)` function forked from UniswapV3 library is needed to evaluate shares to be minted for the user, as the shares minted to the user should be rounded down.

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.2.6 ERC777 tokens will enable caller to self sandwich mint to extract part of deposit

Severity: Low Risk

Context: AlphaProVault.sol#L227-L229

Description: ERC777 is a token standard derived from ERC20 and generally compatible with UniswapV3. However the integration of `AlphaProVault` with UniswapV3 may enable value extraction from vault if one of the tokens of the pool is an ERC777 token. Indeed the `_callTokensToSend` hook can be used by caller to swap in the underlying pool of the vault and modify the price. This causes slippage on the mint operation conducted later by the vault.

- AlphaProVault.sol#L227-L240:

```
// Pull in tokens from sender
//@audit _callTokensToSend is called on msg.sender, which will manipulate price of underlying pool
if (amount0 > 0) token0.safeTransferFrom(msg.sender, address(this), amount0);
if (amount1 > 0) token1.safeTransferFrom(msg.sender, address(this), amount1);

for (uint256 i = 0; i < 3; i++) {
    int24 tickLower = positions[i][0];
    int24 tickUpper = positions[i][1];

    if (liquidities[i] > 0) {
        uint128 liquidityShare = uint128(Math.mulDiv(uint256(liquidities[i]), shares, totalSupply()));
        //@audit mint of liquidity to the vault is done at the manipulated price
        _mintLiquidity(tickLower, tickUpper, liquidityShare);
    }
}
```

Recommendation: Either document the incompatibility with ERC777 standard, or add the twap check again after token transfer:

- AlphaProVault.sol#L227-L240:

```
    // Pull in tokens from sender
    if (amount0 > 0) token0.safeTransferFrom(msg.sender, address(this), amount0);
    if (amount1 > 0) token1.safeTransferFrom(msg.sender, address(this), amount1);

+   checkPriceNearTwap();

    for (uint256 i = 0; i < 3; i++) {
        int24 tickLower = positions[i][0];
        int24 tickUpper = positions[i][1];

        if (liquidities[i] > 0) {
            uint128 liquidityShare = uint128(Math.mulDiv(uint256(liquidities[i]), shares, totalSupply()));
            _mintLiquidity(tickLower, tickUpper, liquidityShare);
        }
    }
```

Charm: Acknowledged. Charm's contracts are incompatible with the ERC777 standard, as described by the docs.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Storage variables can be immutable

Severity: Gas Optimization

Context: AlphaProVault.sol#L99-L102

Description: In `AlphaProVault`, we have a few variables defined during initialization which are never reassigned, and are thus effectively immutable:

```
IUniswapV3Pool public override pool;
IERC20 public token0;
IERC20 public token1;
AlphaProVaultFactory public factory;
```

Since these are effectively immutable, we can mark them as `immutable` to save gas on each read.

Note: This does require switching from use of regular ERC1167 `clone` to deploy this contract from the `AlphaProVaultFactory`. It can alternatively be deployed with `create2` directly or `ClonesWithImmutableArgs` (see `Clones.cloneDeterministicWithImmutableArgs`) to allow for the use of immutables.

Recommendation: Consider making these storage variables immutable, either via:

1. Use `create2` to deploy the vaults in `AlphaProVaultFactory`, then mark the aforementioned storage variables as `immutable`, initializing them in the constructor instead of `initialize`.
2. Use `cloneDeterministicWithImmutableArgs` to deploy the vaults in `AlphaProVaultFactory`, passing the aforementioned storage variables as arguments and updating logic to access them as necessary.

Charm: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Storage variables can be packed

Severity: Gas Optimization

Context: AlphaProVault.sol#L112-L116

Description: We include the following storage variables in `AlphaProVault`:

```
uint256 public override accruedProtocolFees0;
uint256 public override accruedProtocolFees1;
uint256 public override accruedManagerFees0;
uint256 public override accruedManagerFees1;
uint256 public override lastTimestamp;
```

We can reduce the size of these uints to pack them into shared storage slots:

```
// accruedProtocolFees0 and accruedProtocolFees1 are in the same storage slot
uint128 public override accruedProtocolFees0;
uint128 public override accruedProtocolFees1;
// accruedManagerFees0 and accruedManagerFees1 are in the same storage slot
uint128 public override accruedManagerFees0;
uint128 public override accruedManagerFees1;
```

Since these storage vars are always used together, this allows us to save gas on expensive SSTORE/SLOAD operations each time we read/write them. We can also include `lastTimestamp` in one of these slots since it's only used in `rebalance`, in which the others are too:

```
// accruedProtocolFees0 and accruedProtocolFees1 are in the same storage slot
uint128 public override accruedProtocolFees0;
uint128 public override accruedProtocolFees1;
// accruedManagerFees0, accruedManagerFees1, and lastTimestamp are in the same storage slot
uint104 public override accruedManagerFees0;
uint104 public override accruedManagerFees1;
uint40 public override lastTimestamp;
```

Recommendation: Reduce the size of the aforementioned storage variables as indicated above.

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Tick boundary check is off by one in `checkCanRebalance`

Severity: Informational

Context: AlphaProVault.sol#L459-L460

Description: In the function `checkCanRebalance` used to approve the call to `rebalance`, we make sure that the new boundaries for the `base` and `limit` ranges are not out of bounds:

- AlphaProVault.sol#L456-L462:

```
// check price not too close to boundary
int24 maxThreshold = baseThreshold > limitThreshold ? baseThreshold : limitThreshold;
require(
    tick >= TickMath.MIN_TICK + maxThreshold + tickSpacing
        && tick <= TickMath.MAX_TICK - maxThreshold - tickSpacing,
    "PB"
);
```

Unfortunately this tick is more restrictive than simply restricting `newBaseLower >= -maxTick` and `newBaseUpper <= maxTick`, so the function will revert if `newBaseLower == -maxTick` or `newBaseUpper == maxTick` which could be allowed.

Charm: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Fee share may not be distributed to user in edge-case

Severity: Informational

Context: AlphaProVault.sol#L337-L344

Description: During the evaluation of the amounts which should be transferred to user during withdrawal, the burning and collecting of fees is conditioned to `liquidity > 0`; But in the edge case where `fees0` becomes bigger than `totalLiquidity`, the due fees may be non-zero even if the `liquidity == 0`.

> This is an extreme edge-case because any action on the vault which reduces the liquidity, also collects fees. So the only case when this would happen is if the fees collected on a position grow bigger than the liquidity itself.

- AlphaProVault.sol#L329-L345:

```
/// @dev Withdraws share of liquidity in a range from Uniswap pool.
function _burnLiquidityShare(int24 tickLower, int24 tickUpper, uint256 shares, uint256 totalSupply)
    internal
    returns (uint256 amount0, uint256 amount1)
{
    (uint128 totalLiquidity,,,,) = _position(tickLower, tickUpper);
    uint256 liquidity = (uint256(totalLiquidity) * shares) / totalSupply;

    if (liquidity > 0) {
        (uint256 burned0, uint256 burned1, uint256 fees0, uint256 fees1) =
            _burnAndCollect(tickLower, tickUpper, _toUint128(liquidity));

        // Add share of fees
        //@audit if fees0 > totalLiquidity, (fees0 * shares) / totalSupply can be non-zero, and
        ↪  liquidity == 0
        //@audit In this edge-case, fee is not distributed to user
        amount0 = burned0 + ((fees0 * shares) / totalSupply);
        amount1 = burned1 + ((fees1 * shares) / totalSupply);
    }
}
```

Recommendation: One could simply remove the `liquidity > 0` condition:

- AlphaProVault.sol#L329-L345:

```
  /// @dev Withdraws share of liquidity in a range from Uniswap pool.
  function _burnLiquidityShare(int24 tickLower, int24 tickUpper, uint256 shares, uint256 totalSupply)
      internal
      returns (uint256 amount0, uint256 amount1)
  {
      (uint128 totalLiquidity,,,,) = _position(tickLower, tickUpper);
      uint256 liquidity = (uint256(totalLiquidity) * shares) / totalSupply;

-      if (liquidity > 0) {
-          (uint256 burned0, uint256 burned1, uint256 fees0, uint256 fees1) =
-              _burnAndCollect(tickLower, tickUpper, _toUint128(liquidity));
-
-          // Add share of fees
-          amount0 = burned0 + ((fees0 * shares) / totalSupply);
-          amount1 = burned1 + ((fees1 * shares) / totalSupply);
-      }
+      (uint256 burned0, uint256 burned1, uint256 fees0, uint256 fees1) =
+          _burnAndCollect(tickLower, tickUpper, _toUint128(liquidity));
+
+      // Add share of fees
+      amount0 = burned0 + ((fees0 * shares) / totalSupply);
+      amount1 = burned1 + ((fees1 * shares) / totalSupply);
  }
```

Charm: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Minor changes and renaming suggestions

Severity: Informational

Context: *(No context files were provided by the reviewer)*

- `period` could be renamed to `rebalancePeriod` to be more specific.

  AlphaProVault.sol#L118:

```
uint32 public override period;
```

- `maxTwapDeviation` could be renamed to `maxTwapDeviationTicks` to make unit clear:

  AlphaProVault.sol#L129:

```
int24 public override maxTwapDeviation;
```

> Additionally `maxTwapDeviation` can be made `uint24` to avoid unecessary sign checks.

Charm: Acknowledged.

Cantina Managed: Acknowledged.

3.4.4 Emergency burn action should have cooldown

Severity: Informational

Context: AlphaProVault.sol#L777-L780

Description: As seen in reported issues "Manager can allow himself to rebalance unrestricted" and "Vault can be drained if `baseThreshold` is equal to `wideThreshold`" the `emergencyBurn` endpoint enables the manager to do manipulations on liquidity which otherwise would not be possible during a rebalancing in normal conditions (most notably sandwich the `mint` operation on the uniswap v3 pool, because during rebalancing the `burn` and `mint` action happen at the same price).

It would make sense as such to restrict the call to this operation to exceptional conditions, and disable a repeated or frequent usage of the endpoint.

Recommendation: Introduce a `emergencyState` which would restrict `deposit` and `rebalance` actions, with a short cooldown after which the `emergencyState` can be deactivated.

Charm: Acknowledged. Mismanagement such as the one described in this finding is an example of manager risks - this is highlighted in Charm's docs, along with other risks for liquidity providers.

Cantina Managed: Acknowledged.

3.4.5 `_verifyTick` is a misnomer

Severity: Informational

Context: AlphaProVault.sol#L483-L492

Description: `AlphaProVault._verifyTick` takes a `tick` and a `_maxTick` and clamps the `tick` to be within the negative and positive `_maxTick`:

```
/// @dev Verifies that tick is within the range boundaries
function _verifyTick(int24 tick, int24 _maxTick) internal pure returns (int24) {
    if (tick < -_maxTick) {
        return -_maxTick;
    }
    if (tick > _maxTick) {
        return _maxTick;
    }
    return tick;
}
```

Given the name of the function alone, this is a bit unexpected. Instead, it might be expected that the function checks if the `tick` is valid and reverts or returns false if it's not.

Recommendation: Consider renaming the function to something that better represents the behavior, e.g.: `_boundTick` or `_clampTick`.

Charm: Fixed in commit 4932841d.

Cantina Managed: Fix verified.

3.4.6 Shadowed function name

Severity: Informational

Context: AlphaProVault.sol#L255, AlphaProVault.sol#L302, AlphaProVault.sol#L330

Description: In each of `_calcSharesAndAmounts`, `withdraw`, and `_burnLiquidityShare`, we use `totalSupply` as a variable or parameter name, e.g.:

```
uint256 totalSupply = totalSupply();
```

This shadows the function by the same name. As a result, in any scope which we have defined a variable or parameter as `totalSupply`, the function will no longer executable. Currently this doesn't pose any issues and is likely to be blocked by the compiler if it does become an issue, but it's best practice to avoid shadowing function names for improved maintainability.

Recommendation: Use a different name for the variables and parameters referencing the `totalSupply`, e.g.:

```
- uint256 totalSupply = totalSupply();
+ uint256 _totalSupply = totalSupply();
```

Charm: Fixed in commit e47b8853.

Cantina Managed: Fix verified.