SHELLBOXES

Velvet Capital V2

Smart Contract Security Audit

Prepared by ShellBoxes

July 17th, 2023 - August 7th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Velvet Capital
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
`https://github.com/Velvet-Capital/protocol-v2-public`	a7a968ccd39ffedcd372717cd41ce8e155272d2c

Re-Audit

Repository	Commit Hash
`https://github.com/Velvet-Capital/protocol-v2-public`	32452f2cff4eae008c59a376952b6d9d21ffc202

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Velvet Capital engaged ShellBoxes to conduct a security assessment on the Velvet Capital V2 beginning on July 17th, 2023 and ending August 7th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Velvet Capital

Velvet Capital is a DeFi protocol that helps people & institutions create tokenized index funds, portfolios & other financial products with additional yield. The protocol provides all the necessary infrastructure for financial product development being integrated with AMMs, Lending protocols and other DeFi primitives to give users a diverse asset management toolkit.

Issuer	Velvet Capital
Website	`https://www.velvet.capital/`
Type	Solidity Smart Contract
Documentation	Velvet Capital Docs
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

— Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.

— Impact quantifies the technical and economic costs of a successful attack.

— Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact			
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Velvet Capital V2 implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

Throughout the audit, the Velvet Capital team demonstrated commendable professionalism and commitment. Their responsiveness and comprehensive documentation greatly facilitated the process. Notably, they placed a high emphasis on security, promptly addressing and rectifying the majority of the identified issues. In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 5 critical-severity, 4 high-severity, 7 medium-severity, 7 low-severity, 2 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Flaw in Share Minting Leading to Potential Fund Misappropriation	CRITICAL	Mitigated
SHB.2. Potential Over-Minting of Tokens Due to Unchecked Deposited Amount	CRITICAL	Fixed
SHB.3. StreamingFee Check Can Cause a Denial of Service	CRITICAL	Fixed
SHB.4. Incorrect Token Price Calculation Leading to Denial of Service	CRITICAL	Fixed

SHB.5. Inaccuracy in LP Token Price Calculation Due to Decimal Mismatch	CRITICAL	Fixed
SHB.6. Potential Loss of Index Tokens Due to Lack of Swap Result Update	HIGH	Fixed
SHB.7. Misevaluation of User's Investments in LP Tokens	HIGH	Fixed
SHB.8. Potential Portfolio Imbalance Due to OffChain Swaps	HIGH	Fixed
SHB.9. Bypass of Withdrawal Cooldown Period Restriction	HIGH	Fixed
SHB.10. Unfair Distribution of Rewards Due to Timing of claimTokens Function Calls	MEDIUM	Acknowledged
SHB.11. Griefing Attack in Withdrawal Process	MEDIUM	Fixed
SHB.12. Hard-coded Slippage Leading to Potential Fund Freeze	MEDIUM	Acknowledged
SHB.13. Potential Sandwich Attack Due to Chainlink Oracle Failure	MEDIUM	Fixed
SHB.14. Lack of Freshness Check for Chainlink Price Feed Data	MEDIUM	Fixed
SHB.15. Precision Loss in Price Calculation Function	MEDIUM	Fixed
SHB.16. Mismatch Between _tokenAmount and buyAmounts Array Can Lead to Uninvested Funds	MEDIUM	Fixed
SHB.17. Unchecked Transfer Return Value	LOW	Fixed
SHB.18. Missing Array Length Check	LOW	Fixed
SHB.19. Missing Maximum Amount for User Supplied Slippage	LOW	Fixed

SHB.20. Potential Out of Gas Exception Due to Long _tokens Array	LOW	Fixed
SHB.21. Potential Failure of Off-Chain Investment Due to Disabled Tokens	LOW	Fixed
SHB.22. Potential Unrestricted Withdrawals During Pause State	LOW	Fixed
SHB.23. Precision Loss When Dividing Odd Integers by Two	LOW	Fixed
SHB.24. Lack of Cross-Contract Reentrancy Protection	INFORMATIONAL	Fixed
SHB.25. Off-Chain Investment Failure Due to Non-Zero Protocol Fees	INFORMATIONAL	Fixed

3 Finding Details

SHB.1 Flaw in Share Minting Leading to Potential Fund Misappropriation

- Severity : CRITICAL
- Status : Mitigated
- Likelihood : 3
- Impact : 3

Description:

The protocol swaps the invested funds into the tokens of the portfolio, then calculates the USD value of the swap results, and converts them to BNB to decide how many index tokens will be minted for the user. These price conversions to USD and then to BNB are done using Chainlink price feeds.

However, there can be a delay in the reflection of the actual market value of the tokens in the Chainlink price feeds. This delay can be exploited by a user who withdraws and then re-deposits after the value goes up in the feed, ending up with more index tokens while depositing the same initial amount. This means the balance didn't change, but the user got more index tokens, allowing them to withdraw a part of someone else's funds. The same can be applied if the BNB's value increases in USD.

Exploit Scenario:

An attacker can exploit this flaw by monitoring the market for tokens that are going up in value. They can then withdraw their funds and re-deposit after the value goes up in the Chainlink price feed, effectively getting more index tokens while depositing the same initial amount. This allows them to withdraw a part of someone else's funds, leading to financial losses for other users.

Files Affected:

SHB.1.1: Exchange.sol

```
for (uint256 j = 0; j < swapResult.length; j++) {
    investedAmountAfterSlippage = investedAmountAfterSlippage.add(
      oracle.getPriceTokenUSD18Decimals(underlying[j], swapResult[j])
    );
}
```

SHB.1.2: IndexSwap.sol

```
uint256 investedAmountAfterSlippageBNB = _oracle.getUsdEthPrice(
    ↪ investedAmountAfterSlippage);

if (investedAmountAfterSlippageBNB <= 0) {
  revert ErrorLibrary.ZeroFinalInvestmentValue();
}
uint256 tokenAmount;
uint256 _totalSupply = totalSupply();
tokenAmount = getTokenAmount(_totalSupply,
    ↪ investedAmountAfterSlippageBNB, vaultBalanceInBNB);
if (tokenAmount <= 0) {
  revert ErrorLibrary.ZeroTokenAmount();
}
_mintInvest(_to, tokenAmount);
```

Recommendation:

To mitigate this issue, it is recommended to add a delay between the withdrawal and the next invest call to prevent an attacker from exploiting the delay between the real world price and the Chainlink price feeds, or implement a mechanism that locks the withdrawal and invest functions during periods of significant price volatility to reduce the risk.

Updates

The team mitigated the issue by removing the USD to BNB conversion to calculate the minted amount, this action reduces the likelihood of the attack since it will only be applicable on price changes of the portfolio tokens in USD.

SHB.2 Potential Over-Minting of Tokens Due to Unchecked Deposited Amount

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The contract has an issue in its swapOffChainTokens function where it does not check for the actual deposited amount of underlying tokens in a liquidity pool and the returned amount to the user. In the swapOffChainTokens function, the user can input arbitrary buyAmounts, so they can make it appear as if they are depositing a large amount into the vault, but only a small portion of it will actually go to the vault. The majority will be returned to them as leftovers. This can lead to the user having minted more tokens than they actually deposited.

Exploit Scenario:

Let's set a scenario of a portfolio that has only an LP token for simplicity, where the liquidity pool is balanced with 100 tokens each side (100 tokenA, 100 tokenB). An attacker can set a buyAmounts array that looks like this [100000000,100]. The exchange will swap the invested tokens to the underlying tokens of the pool and deposit this unbalanced allocation of tokens to the liquidity pool, the liquidity pool will only take 100 tokenA and 100 tokenB and return 100000000 - 100 tokenA to the attacker, then the index tokens will be minted based on the swap results which do not take into account the returned dust. This results in the attacker having more index tokens than the actual deposit.

Files Affected:

SHB.2.1: AbstractLPHandler.sol

```
(amountA, amountB, liquidity) = router.addLiquidity(
        address(underlying[0]),
        address(underlying[1]),
        _amount[0],
        _amount[1],
        1,
        1,
        _to,
        block.timestamp
      );

      _returnDust(
        underlying[0],
        user // we need to pass user from exchange
      );
      _returnDust(
        underlying[1],
        user // we need to pass user from exchange
      );
```

SHB.2.2: OffChainIndexSwap.sol

```
    // Perform off-chain investment
    balanceInUSD = _offChainInvestment(_initData, _tokenAmount,
        ↪ _lpSlippage);
```

SHB.2.3: OffChainIndexSwap.sol

```
  function _offChainInvestment(
    ExchangeData.ZeroExData memory inputData,
    uint256 _tokenAmount,
    uint256[] calldata _lpSlippage
  ) internal virtual returns (uint256 balanceInUSD) {
    uint256 underlyingIndex = 0;
    balanceInUSD = 0;
    address[] memory _tokens = index.getTokens();
    uint256[] memory _buyAmount = calculateSwapAmountsOffChain(index,
      ↪ _tokenAmount);
    for (uint256 i = 0; i < _tokens.length; i++) {
      // Get the handler contract for the current token
      // Perform off-chain token swap using the exchange contract
      (balanceInUSD, underlyingIndex) = exchange.swapOffChainTokens(
        ExchangeData.IndexOperationData(
          ExchangeData.InputData(
            inputData.buyAmount,
            inputData.sellTokenAddress,
            inputData._offChainHandler,
            inputData._buySwapData
          ),
          index,
          underlyingIndex,
          inputData.protocolFee[i],
          balanceInUSD,
          _lpSlippage[i],
          _buyAmount[i],
          _tokens[i],
          msg.sender
        )
      );
    }
  }
```

Recommendation:

Consider relying on the fair LP price of the returned liquidity by the AMM pair to calculate the amount of index tokens to be minted.

Updates

The team has resolved the issue by relying on the value of the minted LP tokens calculated using a custom aggregator that uses the fair lp price formula.

SHB.3 StreamingFee Check Can Cause a Denial of Service

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The function calculateStreamingFee checks if _lastCharged is less than block.timestamp. If _lastCharged is not less than block.timestamp, the function reverts with ErrorLibrary.NoTimePassedSinceLastCharge(). This check effectively enforces that fees are taken only once per block. Any subsequent calls within the same block will revert, leading to a denial of service.

Exploit Scenario:

An attacker can exploit this vulnerability by front-running all calls to the protocol that takes fees with an operation that calls the calculateStreamingFee function. This will cause all subsequent calls within the same block to revert, effectively causing a denial of service for all those calls.

Files Affected:

SHB.3.1: FeeLibrary.sol

```solidity
function calculateStreamingFee(
    uint256 _totalSupply,
    uint256 _vaultBalance,
    uint256 _lastCharged,
    uint256 _fee
) public view returns (uint256 tokensToMint) {
    if (_lastCharged >= block.timestamp) {
      revert ErrorLibrary.NoTimePassedSinceLastCharge();
    }

    uint256 feeForIntervall = _vaultBalance.mul(_fee).mul(block.
        ↪ timestamp.sub(_lastCharged)).div(365 days).div(
      TOTAL_WEIGHT
    );

    tokensToMint = feeForIntervall.mul(_totalSupply).div(_vaultBalance.
        ↪ sub(feeForIntervall));
}
```

Recommendation:

Consider returning zero if _lastCharged is equal to the block.timestamp to avoid causing a denial of service when the fee was already taken by the protocol for that interval.

Updates

The team has resolved the issue by removing the revert statement and returning zero when _lastCharged is equal to the block.timestamp to avoid DoS when a transaction was executed in the same block.

SHB.3.2: FeeLibrary.sol

```
  function calculateStreamingFee(
    uint256 _totalSupply,
    uint256 _vaultBalance,
    uint256 _lastCharged,
    uint256 _fee
  ) public view returns (uint256 tokensToMint) {
    if (_lastCharged >= block.timestamp) {
      return tokensToMint;
    }
    uint256 feeForIntervall = (_vaultBalance * (_fee) * (block.timestamp
       ↪  - _lastCharged)) / (365 days) / (TOTAL_WEIGHT);

    tokensToMint = (feeForIntervall * _totalSupply) / (_vaultBalance -
       ↪ feeForIntervall);

    return tokensToMint;
  }
```

SHB.4 Incorrect Token Price Calculation Leading to Denial of Service

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The contract uses Chainlink price feeds to calculate the price of a token in multiple handlers. This price is then used to validate the LP slippage. However, when calculating the price of a token, the contract specifies 1e18 as an input to represent one token. While this is correct for tokens that have a decimal of 18, it will yield an extremely incorrect price for tokens that have a different decimal count. This can lead to a denial of service (DoS), as the slippage protection will always revert the deposit and redeem transactions due to the incorrect price calculation. It is worth mentioning that deposit and redeem are used in investments and withdrawals for each non primary token that uses an LP handler, therefore this will cause a DoS in the main functionalities of the protocol. The same issue exists in the AbstractLPHandler for calculating the liquidity fair value price.

Files Affected:

SHB.4.1: ApeSwapLPHandler.sol

```
function deposit(
    address _lpAsset,
    uint256[] memory _amount,
    uint256 _lpSlippage,
    address _to,
    address user
) public payable override {
    address[] memory t = getUnderlying(_lpAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
        ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
        ↪ 1000000000000000000);
    _deposit(_lpAsset, _amount, _lpSlippage, _to, address(router), user,
        ↪  p1, p2);
    emit Deposit(block.timestamp, msg.sender, _lpAsset, _amount, _to);
}
```

SHB.4.2: ApeSwapLPHandler.sol

```
function redeem(FunctionParameters.RedeemData calldata inputData) public
    ↪  override {
    address[] memory t = getUnderlying(inputData._yieldAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
        ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
        ↪ 1000000000000000000);
    _redeem(inputData, routerAddress, p1, p2);
    emit Redeem(block.timestamp, msg.sender, inputData._yieldAsset,
        ↪ inputData._amount, inputData._to, inputData.isWETH);
}
```

SHB.4.3: BiSwapLPHandler.sol

```
function deposit(
    address _lpAsset,
    uint256[] memory _amount,
    uint256 _lpSlippage,
    address _to,
    address user
) public payable override {
    address[] memory t = getUnderlying(_lpAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
       ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
       ↪ 1000000000000000000);
    _deposit(_lpAsset, _amount, _lpSlippage, _to, address(router), user,
       ↪  p1, p2);
    emit Deposit(block.timestamp, msg.sender, _lpAsset, _amount, _to);
}
```

SHB.4.4: BiSwapLPHandler.sol

```
function redeem(FunctionParameters.RedeemData calldata inputData) public
   ↪  override {
    address[] memory t = getUnderlying(inputData._yieldAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
       ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
       ↪ 1000000000000000000);
    _redeem(inputData, routerAddress, p1, p2);
    emit Redeem(block.timestamp, msg.sender, inputData._yieldAsset,
       ↪ inputData._amount, inputData._to, inputData.isWETH);
}
```

SHB.4.5: PancakeSwapLPHandler.sol

```
function deposit(
    address _lpAsset,
    uint256[] memory _amount,
    uint256 _lpSlippage,
    address _to,
    address user
) public payable override {
    address[] memory t = getUnderlying(_lpAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
       ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
       ↪ 1000000000000000000);
    _deposit(_lpAsset, _amount, _lpSlippage, _to, address(router), user,
       ↪  p1, p2);
    emit Deposit(block.timestamp, msg.sender, _lpAsset, _amount, _to);
}
```

SHB.4.6: PancakeSwapLPHandler.sol

```
function redeem(FunctionParameters.RedeemData calldata inputData) public
   ↪  override {
    address[] memory t = getUnderlying(inputData._yieldAsset);
    uint p1 = _oracle.getPriceTokenUSD18Decimals(t[0],
       ↪ 1000000000000000000);
    uint p2 = _oracle.getPriceTokenUSD18Decimals(t[1],
       ↪ 1000000000000000000);
    _redeem(inputData, routerAddress, p1, p2);
    emit Redeem(block.timestamp, msg.sender, inputData._yieldAsset,
       ↪ inputData._amount, inputData._to, inputData.isWETH);
}
```

SHB.4.7: AbstractLPHandler.sol

```
function _calculatePrice(address t, address priceOracle) internal view
   ↪ returns (uint256) {
    address[] memory underlying = _getUnderlyingTokens(t);
    LPInterface _asset = LPInterface(t);
    (uint reserve0, uint reserve1, ) = _asset.getReserves();
    uint totalSupply = _asset.totalSupply();
    uint price0 = IPriceOracle(priceOracle).getPriceTokenUSD18Decimals(
       ↪ underlying[0], ONE_ETH);
    uint price1 = IPriceOracle(priceOracle).getPriceTokenUSD18Decimals(
       ↪ underlying[1], ONE_ETH);

    uint256 sqrtReserve = Babylonian.sqrt(reserve0.mul(reserve1));
    uint256 sqrtPrice = Babylonian.sqrt(price0.mul(price1));
    uint256 price = sqrtReserve.mul(sqrtPrice).mul(2).div(totalSupply);
    return price;
}
```

Recommendation:

To mitigate this issue, it is recommended to dynamically calculate the token representation based on the token's decimal count. Instead of hard-coding 1e18 as the representation of one token, the contract should call the decimals() function on the token contract to get the correct decimal count. This will ensure that the price calculation is accurate for all tokens, regardless of their decimal count.

Updates

The team has resolved the issue by adding a function in the PriceOracle contract that calculated the price of one token taking into account the decimals.

SHB.4.8: PriceOracle.sol

```
    /**
     * @notice Returns the latest token price for a specific token for 1
        ↪ unit
     * @param _base base asset address
     * @return amountOut The latest USD token price of the base token in
        ↪ 18 decimals
     */
    function getPriceForOneTokenInUSD(address _base) public view returns (
       ↪ uint256 amountOut) {
```

```
    uint256 amountIn = 10 ** IERC20MetadataUpgradeable(_base).decimals()
        ↪ ;
    amountOut = getPriceTokenUSD18Decimals(_base, amountIn);
  }
```

SHB.5 Inaccuracy in LP Token Price Calculation Due to Decimal Mismatch

- Severity : CRITICAL
- Status : Fixed

- Likelihood : 3
- Impact : 3

Description:

The _calculatePrice function gets the price of a full token (1 x decimal) from Chainlink, but the reserves returned by the pair are in units of tokens (already multiplied by the decimal). This mismatch in decimal representation leads to an inaccuracy when calculating the price of an LP token.

Files Affected:

SHB.5.1: AbstractLPHandler.sol

```
function _calculatePrice(address t, address priceOracle) internal view
    ↪ returns (uint256) {
    address[] memory underlying = _getUnderlyingTokens(t);
    LPInterface _asset = LPInterface(t);
    (uint reserve0, uint reserve1, ) = _asset.getReserves();
    uint totalSupply = _asset.totalSupply();
    uint price0 = IPriceOracle(priceOracle).getPriceTokenUSD18Decimals(
        ↪ underlying[0], ONE_ETH);
    uint price1 = IPriceOracle(priceOracle).getPriceTokenUSD18Decimals(
        ↪ underlying[1], ONE_ETH);

    uint256 sqrtReserve = Babylonian.sqrt(reserve0.mul(reserve1));
    uint256 sqrtPrice = Babylonian.sqrt(price0.mul(price1));
    uint256 price = sqrtReserve.mul(sqrtPrice).mul(2).div(totalSupply);
    return price;
}
```

Recommendation:

To mitigate this issue, it is recommended to align the decimal representation when getting the prices from Chainlink and when getting the reserves from the pair. This can be achieved by getting the price of one unit of the token instead of a full token. This would ensure that the calculation is performed with the correct decimal representation, leading to an accurate price calculation for LP tokens.

Updates

The team has resolved the issue by calculating the value of the minted LP tokens using a custom aggregator that uses the fair LP price formula.

SHB.6 Potential Loss of Index Tokens Due to Lack of Swap Result Update

- Severity : HIGH
- Status : Fixed
- Likelihood : 3
- Impact : 2

Description:

The _swapTokenToToken function does not update the swapResult array if both tokenIn and tokenOut are primary tokens. This leads to the function returning zero as a default value, which will not get added to investedAmountAfterSlippage. The investedAmountAfterSlippage is used to calculate the index tokens to be minted. This could potentially lead to a loss of index tokens for the user.

Files Affected:

SHB.6.1: Exchange.sol

```
if (!tokenInfoIn.primary  !tokenInfoOut.primary) {
  if (inputData._isInvesting) {
    swapResult = _swapTokenToTokenInvest(inputData, tokenInfoIn.enabled)
        ↪ ;
  } else {
    swapResult = _swapTokenToTokenWithdraw(inputData);
  }
} else {
  IHandler handler = IHandler(tokenInfoOut.handler);
  swapResult = new uint256[](1);
  if (isWETH(tokenOut, address(handler))) {
    address to = inputData._to;
    if (inputData._isInvesting) {
      to = address(this);
    }
    _swapTokenToETH(
      FunctionParameters.SwapTokenToETHData(
        tokenIn,
        to,
        inputData._swapHandler,
        inputData._swapAmount,
        inputData._slippage,
        inputData._lpSlippage
      )
    );
    if (inputData._isInvesting) {
      uint256 balance = address(this).balance;
      IWETH(tokenOut).deposit{value: balance}();
      if (inputData._to != address(this)) {
        IWETH(tokenOut).transfer(inputData._to, balance);
      }
    }
  } else {
    swapResult[0] = IndexSwapLibrary.transferAndSwapTokenToToken(
      tokenIn,
      swapHandler,
      inputData._swapAmount,
      inputData._slippage,
      tokenOut,
      inputData._to,
      tokenInfoIn.enabled
    );
  }
}
return swapResult;
```

Recommendation:

To mitigate this issue, it is recommended to update the swapResult array with the amount of ETH returned from the _swapTokenToETH function if both tokenIn and tokenOut are primary tokens.

Updates

The team has resolved the issue by assigning the _swapTokenToETH return value to the swapResult.

SHB.6.2: Exchange.sol

```
} else {
  IHandler handler = IHandler(tokenInfoOut.handler);
  swapResult = new uint256[](1);
  if (isWETH(tokenOut, address(handler))) {
    address to = inputData._to;
    if (inputData._isInvesting) {
      to = address(this);
    }
    swapResult = _swapTokenToETH(
      FunctionParameters.SwapTokenToETHData(
        tokenIn,
        to,
        inputData._swapHandler,
        inputData._swapAmount,
        inputData._slippage,
        inputData._lpSlippage
      )
    );
```

SHB.7 Misevaluation of User's Investments in LP Tokens

- Severity : HIGH
- Status : Fixed
- Likelihood : 2
- Impact : 3

Description:

The investInFund mis-evaluates the value of a user's investment in liquidity provider (LP) tokens. The project implements index tokens that represent the investor's portfolio, a part of which can be LP tokens from providing liquidity to a pair. The project calculates the value of these LP tokens based on the underlying tokens' value in USD. However, this may not yield accurate results due to the phenomenon known as impermanent loss, which LP providers typically experience when the price of one of the tokens in the pair shifts in the market.

In a scenario where the price of one of the tokens in the pair shifts significantly, the calculated value of the LP tokens based on the underlying tokens' value in USD may not accurately reflect the user's investment. This can lead to a misrepresentation of the user's portfolio value, potentially causing financial losses to the protocol.

Files Affected:

SHB.7.1: IndexSwap.sol

```
investedAmountAfterSlippage = _exchange._swapTokenToTokens{value: msg.
    ↪ value}(
  FunctionParameters.SwapTokenToTokensData(
    address(this),
    _token,
    investData._swapHandler,
    msg.sender,
    _amount,
    totalSupply(),
    amount,
    slippage,
    investData._lpSlippage
  )
);

uint256 investedAmountAfterSlippageBNB = _oracle.getUsdEthPrice(
    ↪ investedAmountAfterSlippage);

if (investedAmountAfterSlippageBNB <= 0) {
  revert ErrorLibrary.ZeroFinalInvestmentValue();
}
uint256 tokenAmount;
uint256 _totalSupply = totalSupply();
tokenAmount = getTokenAmount(_totalSupply,
    ↪ investedAmountAfterSlippageBNB, vaultBalanceInBNB);
if (tokenAmount <= 0) {
  revert ErrorLibrary.ZeroTokenAmount();
}
_mintInvest(_to, tokenAmount);
```

SHB.7.2: OffChainIndexSwap.sol

```
// Perform off-chain investment
balanceInUSD = _offChainInvestment(_initData, _tokenAmount, _lpSlippage)
    ↪ ;

// Calculate the invested amount in BNB after slippage
uint256 investedAmountAfterSlippageBNB = oracle.getUsdEthPrice(
    ↪ balanceInUSD);

// Ensure the final invested amount is not zero
require(investedAmountAfterSlippageBNB > 0, "final invested amount is
    ↪ zero");

// Calculate the vault balance in BNB
uint256 vaultBalanceBNB = oracle.getUsdEthPrice(vaultBalance);

// Calculate the token amount to be minted
uint256 tokenAmount;
uint256 _totalSupply = index.totalSupply();
if (_totalSupply > 0) {
  tokenAmount = IndexSwapLibrary._mintShareAmount(
      ↪ investedAmountAfterSlippageBNB, vaultBalanceBNB, _totalSupply);
} else {
  tokenAmount = investedAmountAfterSlippageBNB;
}

// Ensure the token amount is not zero
require(tokenAmount > 0, "token amount is 0");

// Mint investment tokens to the specified address
index.mintInvest(_to, tokenAmount);
```

SHB.7.3: Exchange.sol

```
for (uint256 j = 0; j < swapResult.length; j++) {
    investedAmountAfterSlippage = investedAmountAfterSlippage.add(
        oracle.getPriceTokenUSD18Decimals(underlying[j], swapResult[j])
      );
  }
```

Recommendation:

To mitigate this issue, it is recommended to use the getFairLpPrice function, which calculates the fair price of an LP token based on the real reserves.

Updates

The team resolved the issue by relying on the value of the minted LP tokens calculated using a custom aggregator, which uses the fair LP price formula.

SHB.8 Potential Portfolio Imbalance Due to OffChain Swaps

- Severity : HIGH
- Status : Fixed
- Likelihood : 3
- Impact : 2

Description:

The protocol allows for off-chain exchanges, such as the 0x protocol, to generate transactions that will swap a user's tokens into the portfolio tokens. The contract allows the user to input the buyAmounts for how the invested amount will be allocated in the portfolio, then it calculates these amounts using the denorms and verifies them to be close to the inputted values by the user, the actual verification passes if the user supplied amounts are at most 50% smaller than the expected amounts, and will always pass if we pass more than the expected amount.

Therefore, the contract allows for a high difference between the inputted values and the calculated values. This can allow a user to capitalize on this discrepancy to unbalance the portfolio, putting it at a different risk level from the one intended by the portfolio creator.

In addition to that, this can result in triggering multiple rebalancing transactions to get the portfolio back to the rebalanced state. It's worth mentioning that this can result in a significant loss to the investors due to the fees that will be spent in the rebalancing process.

Exploit Scenario:

An attacker can exploit this flaw by inputting buyAmounts that significantly differ from the calculated values. This can allow the attacker to unbalance the portfolio, potentially putting it at a different risk level from the one intended by the portfolio creator. This could lead to financial losses for other users.

Files Affected:

SHB.8.1: Exchange.sol

```
function validateAmount(uint256 expectedAmount, uint256 userAmount,
    ↪ uint256 len) internal pure {
    uint256 PERCENTIn18Decimal = 10 ** 22;
    uint256 diff = expectedAmount.div(len).mul(PERCENTIn18Decimal).div(
        ↪ userAmount);
    uint256 diffPercentage = diff < PERCENTIn18Decimal ?
        ↪ PERCENTIn18Decimal.sub(diff) : diff.sub(PERCENTIn18Decimal);
    if (diffPercentage > PERCENTIn18Decimal) {
      revert ErrorLibrary.InvalidBuyValues();
    }
}
```

Recommendation:

To mitigate this issue, it is recommended to implement a stricter verification mechanism for the userAmount inputted by the user. This could involve reducing the allowed difference between the inputted values and the calculated values. This would reduce the risk of causing an unbalance to the portfolio.

Updates

The team has resolved the issue by adjusting the amount validation process to require a reasonable difference between the expectedAmount and userAmount.

SHB.8.2: Exchange.sol

```
function validateAmount(uint256 expectedAmount, uint256 userAmount,
  ↪ uint256 underlyingLen) internal pure {
  uint256 exceptedRangeDecimal = 10 ** 6;
  uint256[] memory diff = new uint256[](underlyingLen);

  if (underlyingLen > 1) {
    uint amount0 = expectedAmount / underlyingLen;
    uint amount1 = expectedAmount - amount0;

    diff[0] = getdiff(userAmount, amount0, exceptedRangeDecimal);

    diff[1] = getdiff(userAmount, amount1, exceptedRangeDecimal);
  } else {
    diff[0] = getdiff(userAmount, expectedAmount, exceptedRangeDecimal
      ↪ );
  }
  for (uint256 j = 0; j < underlyingLen; j++) {
    if (diff[j] > exceptedRangeDecimal) {
      revert ErrorLibrary.InvalidBuyValues();
    }
  }
}
```

SHB.8.3: Exchange.sol

```
function getdiff(uint _userAmount, uint _calcAmount, uint
  ↪ _exceptedRangeDecimal) internal pure returns (uint) {
  return
    _userAmount > _calcAmount
      ? (_userAmount * _exceptedRangeDecimal) / _calcAmount
```

```
        : (_calcAmount * _exceptedRangeDecimal) / _userAmount;
}
```

SHB.9 Bypass of Withdrawal Cooldown Period Restriction

- Severity : HIGH
- Status : Fixed
- Likelihood : 3
- Impact : 2

Description:

The investInFund function allows an investor to specify a _to address that will receive the minted index tokens and updates this address's timestamp to restrict it from instantly withdrawing, forcing it to wait for the cooldown period to end. However, this restriction can be easily bypassed by transferring the index tokens to another address and withdrawing using that address. This is possible when transferableToPublic is enabled in the config or when two users collaborate in the case when transferable is true and those users are whitelisted.

Exploit Scenario:

Any user can exploit this issue by transferring the index tokens to another address and withdrawing using that address, effectively bypassing the withdrawal restriction. This can allow the attacker to withdraw their funds before the cooldown period ends.

Files Affected:

SHB.9.1: IndexSwap.sol

```
_mintInvest(_to, tokenAmount);
lastInvestmentTime[_to] = block.timestamp;
```

SHB.9.2: IndexSwap.sol

```
function withdrawFund(FunctionParameters.WithdrawFund calldata initData)
    ↪ external nonReentrant notPaused {
```

```
    IndexSwapLibrary.checkCoolDownPeriod(lastInvestmentTime[msg.sender],
      ↪  _tokenRegistry);
```

SHB.9.3: IndexSwap.sol

```
function _beforeTokenTransfer(address from, address to, uint256 amount)
    ↪ internal virtual override {
    super._beforeTokenTransfer(from, to, amount);
    IndexSwapLibrary._beforeTokenTransfer(from, to, _iAssetManagerConfig
      ↪ );
}
```

SHB.9.4: IndexLibrary.sol

```
  function _beforeTokenTransfer(address from, address to,
    ↪ IAssetManagerConfig config) external {
    if (from == address(0)  to == address(0)) {
      return;
    }
    if (!(config.transferableToPublic()  (config.transferable() &&
      ↪ config.whitelistedUsers(to)))) {
      revert ErrorLibrary.Transferprohibited();
    }
  }
```

Recommendation:

To mitigate this issue, it is recommended to implement a mechanism that tracks the original address that received the minted index tokens and applies the withdrawal restriction to any subsequent addresses that receive the tokens. This would prevent users from being able to bypass the withdrawal restriction by transferring the tokens to another address. Alternatively, the contract could disallow transfers of index tokens during the cooldown period, ensuring that the withdrawal restriction cannot be bypassed.

Updates

The team resolved the issue by adding a check in the _beforeTokenTransfer that requires the cooldown period to pass before allowing the token transfer.

SHB.9.5: IndexSwap.sol

```
function _beforeTokenTransfer(address from, address to, uint256 amount
    ↪ ) internal virtual override {
  super._beforeTokenTransfer(from, to, amount);
  if (from == address(0)  to == address(0)) {
    return;
  }
  if (
    !(_iAssetManagerConfig.transferableToPublic()
      (_iAssetManagerConfig.transferable() && _iAssetManagerConfig.
          ↪ whitelistedUsers(to)))
  ) {
    revert ErrorLibrary.Transferprohibited();
  }
  checkCoolDownPeriod(from);
}
```

SHB.9.6: IndexSwap.sol

```
function checkCoolDownPeriod(address _user) public view {
  if (getRemainingCoolDown(_user) > 0) {
    revert ErrorLibrary.CoolDownPeriodNotPassed();
  }
}
```

SHB.10 Unfair Distribution of Rewards Due to Timing of claimTokens Function Calls

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 2
- Impact : 2

Description:

The claimTokens function in the IndexSwap contract, which can be called by anyone, collects rewards from handlers that require a method call to harvest the rewards. These rewards are then added to the vault. If this function is not invoked before any investInFund call, a new depositor could potentially receive a share of the rewards that were generated by other investors. Similarly, if it is not called before withdrawFund calls, the withdrawing investor might not receive their share of the rewards generated by their capital. This can lead to an unfair distribution of rewards.

Files Affected:

SHB.10.1: IndexSwap.sol

```
function claimTokens(address[] calldata tokens) external nonReentrant {
    _exchange.claimTokens(IIndexSwap(address(this)), tokens);
}
```

SHB.10.2: Exchange.sol

```
function claimTokens(IIndexSwap _index, address[] calldata _tokens)
    ↪ external onlyIndexManager {
    for (uint256 i = 0; i < _tokens.length; i++) {
      address _token = _tokens[i];
      IHandler handler = IHandler(getTokenInfo(_token).handler);

      (bytes memory callData, address callAddress) = handler.
        ↪ getClaimTokenCalldata(_token, _index.vault());

      if (callAddress != zeroAddress) {
        safe.executeWallet(callAddress, callData);
      }
    }

    emit TokensClaimed(block.timestamp, address(_index), _tokens);
}
```

Recommendation:

Consider implementing a mechanism that automatically distributes rewards to investors in proportion to their shares at the time of each deposit or withdrawal. This would ensure that rewards are fairly distributed and cannot be manipulated by timing transactions.

Updates

The team acknowledged the issue, stating that the asset manager will be specifying the harvest time and frequency in the strategy (frontend).So, users can consider this information to choose their investment time.

SHB.11 Griefing Attack in Withdrawal Process

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The contract has a vulnerability in its withdrawal function that allows an attacker to grief any investor who wants to withdraw their funds. The contract enforces a duration between the investor's last deposit and their withdrawal. However, when investing, an investor can specify a _to address that will receive the shares and also update its lastInvestmentTime to block.timestamp. This means an attacker can invest the minimum amount of shares for another investor, updating their lastInvestmentTime and effectively preventing them from withdrawing their funds.

Exploit Scenario:

An attacker can exploit this issue by front-running the withdrawal transaction of any investor by investing the minimum amount of shares then, updating their lastInvestmentTime and effectively preventing them from withdrawing their funds. This can be done repeatedly, causing continuous grief to the investors.

Files Affected:

SHB.11.1: IndexSwap.sol

```
_mintInvest(_to, tokenAmount);
lastInvestmentTime[_to] = block.timestamp;
```

SHB.11.2: IndexSwap.sol

```
function withdrawFund(FunctionParameters.WithdrawFund calldata initData)
    ↪ external nonReentrant notPaused {
    IndexSwapLibrary.checkCoolDownPeriod(lastInvestmentTime[msg.sender],
        ↪ _tokenRegistry);
```

Recommendation:

To mitigate this issue, it is recommended to separate the logic for updating the lastInvestmentTime from the investment function. This way, only the investor themselves can update their lastInvestmentTime when they make an investment. Alternatively, a validation could be added to ensure that the _to address in the investment function matches msg.sender, preventing an attacker from updating the lastInvestmentTime of another investor.

Updates

The team has resolved the issue by removing the option for users to invest on behalf of someone else. In addition to that, The cooldown period was adapted to take into account the invested amount.

SHB.12 Hard-coded Slippage Leading to Potential Fund Freeze

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The contract uses a hardcoded slippage of 10% in the OneInch, Paraswap, and ZeroEx handlers. While this is generally a good practice to avoid losing value in MEV scenarios, it can become an issue in volatile market conditions. If the price of a token fluctuates by more than 10% within a short period, transactions may fail due to the slippage limit, effectively leading to a freeze of funds.

In a highly volatile market, the price of a token can fluctuate by more than 10% within a short period. If a user tries to perform a transaction during this period, the transaction may fail due to the hard-coded slippage limit of 10%. This can effectively lead to a freeze of funds, as users may be unable to perform transactions until the market stabilizes.

Files Affected:

SHB.12.1: ExternalSlippageControl.sol

```
function getSlippage(uint256 _amount) internal view returns (uint256
    ↪ minAmount) {
  minAmount = _amount.mul(HUNDRED_PERCENT.sub(maxSlippage)).div(
      ↪ HUNDRED_PERCENT);
}
```

SHB.12.2: ExternalSlippageControl.sol

```
function validateSwap(uint priceSellToken, uint priceBuyToken)
    ↪ internal view {
```

```
    if (maxSlippage != 0) {
      if (priceBuyToken < getSlippage(priceSellToken)) {
        revert ErrorLibrary.InvalidAmount();
      }
    }
  }
```

Recommendation:

To mitigate this issue, it is recommended to implement a dynamic slippage mechanism instead of using a hard-coded value. This mechanism could adjust the slippage limit based on market conditions, allowing for higher slippage in more volatile markets and lower slippage in more stable markets. This would provide a balance between protecting users from MEV and ensuring that transactions can still be performed in volatile market conditions.

Updates

The team acknowledged the issue, stating that they'll be adapting the maxSlippage in volatile market conditions.

SHB.13 Potential Sandwich Attack Due to Chainlink Oracle Failure

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The AMM handlers rely on Chainlink to determine the value of a token and calculate the minAmount depending on the user-supplied slippage. However, if the Chainlink oracle fails to deliver the value, the user will add/remove liquidity with no slippage protection. This could potentially expose the user to a sandwich attack, where an attacker manipulates the token pair to their extract MEV.

Exploit Scenario:

A user calls investInFund from IndexSwap to invest. The tokens invested will be swapped to various other tokens using a swapHandler. The slippage calculation is done in the getSlippage function and it uses Chainlink to get prices. However, if Chainlink fails, the currentAmount will be set to 0, and so the investor will enter the trade with no slippage protection. This could expose investors to a sandwich attack, where an attacker manipulates the price of the token to their advantage, potentially leading to financial losses.

Files Affected:

SHB.13.1: SlippageControl.sol

```
function _validateLPSlippage(
  uint _amountA,
  uint _amountB,
  uint _priceA,
  uint _priceB,
  uint _lpSlippage
) internal view {
  require(maxSlippage >= _lpSlippage, "Invalid LP Slippage!");
  uint amountDivision = _amountA.mul(10 ** 18).div(_amountB);
  uint priceDivision = _priceB.mul(10 ** 18).div(_priceA);
  uint absoluteValue = 0;
  if (amountDivision > priceDivision) {
    absoluteValue = amountDivision.sub(priceDivision);
  } else {
    absoluteValue = priceDivision.sub(amountDivision);
  }
  if (absoluteValue.mul(10 ** 2) > (_lpSlippage.mul(10 ** 18))) {
    revert ErrorLibrary.InvalidAmount();
```

Recommendation:

To mitigate this issue, it is recommended to add a check if the returned value is 0, the function should revert with an appropriate error message or use a default value for slippage. This will ensure that the user always enters the trade with slippage protection, preventing potential sandwich attacks.

Updates

The team resolved the issue by reverting the transaction when the price feed returns zero as a price.

SHB.13.2: PriceOracle.sol

```solidity
function latestRoundData(address base, address quote) internal view
    ↪ returns (int256) {
  (
    ,
    /*uint80 roundID*/
    int256 price /*uint startedAt*/ /*uint timeStamp*/ /*uint80
        ↪ answeredInRound*/,
    ,
    uint256 updatedAt,

  ) = aggregatorAddresses[base].aggregatorInterfaces[quote].
      ↪ latestRoundData();

  if (updatedAt + oracleExpirationThreshold < block.timestamp) {
    revert ErrorLibrary.PriceOracleExpired();
  }

  if (price == 0) {
    revert ErrorLibrary.PriceOracleInvalid();
  }

  return price;
}
```

SHB.14 Lack of Freshness Check for Chainlink Price Feed Data

- Severity : MEDIUM
- Status : Fixed

- Likelihood : 1
- Impact : 3

Description:

The contract uses Chainlink price feeds to get the latest price of tokens. However, it does not check the updatedAt value returned by the latestRoundData function. According to Chainlink's documentation, consumers are encouraged to check the updatedAt value to ensure they are receiving fresh data.

If the updatedAt value is not checked, the contract could potentially use stale or outdated price data, which could lead to incorrect calculations and potential loss of funds.

Files Affected:

SHB.14.1: PriceOracle.sol

```
function latestRoundData(address base, address quote) internal view
    ↪ returns (int256) {
    (
      ,
      /*uint80 roundID*/
      int256 price /*uint startedAt*/ /*uint timeStamp*/ /*uint80
          ↪ answeredInRound*/,
      ,
      ,

    ) = aggregatorAddresses[base].aggregatorInterfaces[quote].
        ↪ latestRoundData();
    return price;
}
```

Recommendation:

The contract should check the updatedAt value returned by the latestRoundData function and revert the transaction if the data is not fresh.

Updates

The team resolved the issue by requiring the updatedAt to not be older than a oracleExpirationThreshold.

SHB.14.2: PriceOracle.sol

```
function latestRoundData(address base, address quote) internal view
    ↪ returns (int256) {
  (
    ,
    /*uint80 roundID*/
    int256 price /*uint startedAt*/ /*uint timeStamp*/ /*uint80
        ↪ answeredInRound*/,
    ,
    uint256 updatedAt,

  ) = aggregatorAddresses[base].aggregatorInterfaces[quote].
      ↪ latestRoundData();

  if (updatedAt + oracleExpirationThreshold < block.timestamp) {
    revert ErrorLibrary.PriceOracleExpired();
  }

  if (price == 0) {
    revert ErrorLibrary.PriceOracleInvalid();
  }

  return price;
}
```

SHB.15 Precision Loss in Price Calculation Function

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

The contract has a precision loss issue in the getPriceTokenUSD18Decimals function. The contract performs division before multiplication (Decimal Normalization), which results in a loss of precision. Specifically, the price will lose all the decimal points received by the price feed.

Files Affected:

SHB.15.1: PriceOracle.sol

```
function getPriceTokenUSD18Decimals(address _base, uint256 amountIn)
    ↪ public view returns (uint256 amountOut) {
    uint256 output = uint256(getPrice(_base, Denominations.USD));
    uint256 decimalChainlink = decimals(_base, Denominations.USD);
    IERC20MetadataUpgradeable token = IERC20MetadataUpgradeable(_base);
    uint8 decimal = token.decimals();

    uint256 diff = uint256(18).sub(decimal);

    amountOut = output.mul(amountIn).div(10 ** decimalChainlink).mul(10
        ↪ ** diff);
}
```

Recommendation:

To mitigate this issue, it is recommended to rearrange the operations to perform multiplication before division. This can help prevent the loss of precision. The corrected line of code would be: amountOut = output.mul(amountIn).mul(10 ** diff).div(10 ** decimalChainlink); This change ensures that the multiplication operation is performed before the division operation, which increases the value of the amountOut, thus preserving precision.

Updates

The team resolved the issue by changing the operation order and performing multiplications before divisions.

SHB.15.2: PriceOracle.sol

```
function getPriceTokenUSD18Decimals(address _base, uint256 amountIn)
    ↪ public view returns (uint256 amountOut) {
  uint256 output = uint256(getPrice(_base, Denominations.USD));
  uint256 decimalChainlink = decimals(_base, Denominations.USD);
  IERC20MetadataUpgradeable token = IERC20MetadataUpgradeable(_base);
  uint8 decimal = token.decimals();

  uint256 diff = uint256(18) - (decimal);

  amountOut = (output * amountIn * (10 ** diff)) / (10 **
      ↪ decimalChainlink);
}
```

SHB.16 Mismatch Between _tokenAmount and buyAmounts Array Can Lead to Uninvested Funds

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

The smart contract has an issue in its investInFundOffChain function where the _tokenAmount parameter, which represents the amount the user wants to invest, does not necessarily match the actual amounts invested as specified in the buyAmounts array. This mismatch can lead to a situation where some funds remain uninvested in the contract.

Exploit Scenario:

An investor may be exposed to this issue by unintentionally providing a _tokenAmount that is larger than the total of the buyAmounts array. This would result in some funds remaining uninvested in the contract. An attacker could then potentially withdraw these uninvested funds from the contract using the same issue, effectively withdrawing funds from the contract.

Files Affected:

SHB.16.1: OffChainIndexSwap.sol

```
if (msg.value > 0) {
  if (!(WETH == _initData.sellTokenAddress)) {
    revert ErrorLibrary.InvalidToken();
  }
  _tokenAmount = msg.value;
  IndexSwapLibrary._checkInvestmentValue(_tokenAmount,
     ↪ iAssetManagerConfig);

  // Deposit ETH into WETH
  IWETH(WETH).deposit{value: msg.value}();

  // Transfer the WETH to index operations contract
  IWETH(WETH).transfer(address(exchange), _tokenAmount);
} else {
  // Check permission and balance for the sell token
  IndexSwapLibrary._checkPermissionAndBalance(
    _initData.sellTokenAddress,
    _tokenAmount,
    iAssetManagerConfig,
    msg.sender
  );

  // Get the token balance in BNB
  uint256 tokenBalanceInBNB = _getTokenBalanceInBNB(_initData.
      ↪ sellTokenAddress, _tokenAmount);
  IndexSwapLibrary._checkInvestmentValue(tokenBalanceInBNB,
      ↪ iAssetManagerConfig);

  // Transfer the sell token from the sender to index operations
      ↪ contract
  TransferHelper.safeTransferFrom(_initData.sellTokenAddress, msg.sender
      ↪ , address(exchange), _tokenAmount);
}
```

Recommendation:

To mitigate this issue, it is recommended to return the uninvested funds if the _tokenAmount was more than the required amount to get the buyAmounts.

Updates

The team resolved the issue by returning the unused funds to the investor using the returnUninvestedFunds function.

SHB.16.2: Exchange.sol

```
  /**
   * @notice This function is used to return any uninvested funds left
       ↪ in the Exchange handler during OffChain/Onchain investment
   * @param _token Address of the deposit token whose undeposited dust
       ↪ is left stuck in the contract
   * @param _to Address where the uninvested funds have to be sent
   */
```

```
function returnUninvestedFunds(address _token, address _to, uint256
    ↪ _balance) internal {
  if (_token != WETH) {
    TransferHelper.safeTransfer(_token, _to, _balance);
  } else {
    (bool success, ) = payable(_to).call{value: _balance}("");
    if (!success) {
      revert ErrorLibrary.ETHTransferFailed();
    }
  }
  emit returnedUninvestedFunds(_to, _token, _balance, block.timestamp)
      ↪ ;
}
```

SHB.17 Unchecked Transfer Return Value

- Severity: LOW
- Status: Fixed
- Likelihood: 1
- Impact: 2

Description:

The contract has an issue in its _safeTokenTransfer function, where it does not check the return value of a token transfer. The function generates transfer calldata to the Gnosis Safe vault to execute, but the Safe only checks the transaction status without verifying if the function returns a boolean and whether it's true or not. This could potentially lead to unnoticed failed transfers.

Files Affected:

SHB.17.1: Exchange.sol

```
function _safeTokenTransfer(address token, uint256 amount, address to)
    ↪ internal {
```

```
    bytes memory inputData = abi.encodeWithSelector(IERC20Upgradeable.
        ↪ transfer.selector, to, amount);

    safe.executeWallet(token, inputData);
}
```

SHB.17.2: VelvetSafeModule.sol

```
function executeWallet(
    address handlerAddresses,
    bytes calldata encodedCalls
) public onlyOwner returns (bool isSuccess) {
    isSuccess = exec(handlerAddresses, 0, encodedCalls, Enum.Operation.
        ↪ Call);
    require(isSuccess, "Call failed");
}
```

Recommendation:

To mitigate this issue, it is recommended to add a check, if there is a return value of the token transfer in the _safeTokenTransfer function, then it should be required to be true to avoid the case where the transfer fails silently.

Updates

The team resolved the issue by adding a check if the transfer function returns a boolean representing the status.

SHB.17.3: Exchange.sol

```
  function _safeTokenTransfer(address token, uint256 amount, address to)
      ↪  internal {
    bytes memory inputData = abi.encodeWithSelector(IERC20Upgradeable.
        ↪ transfer.selector, to, amount);

    (, bytes memory data) = safe.executeWallet(token, inputData);

    // bool returned by executeWallet is already checked
    if (!(data.length == 0  abi.decode(data, (bool)))) revert
      ↪ ErrorLibrary.TransferFailed();
  }
```

SHB.17.4: VelvetSafeModule.sol

```
  function executeWallet(
    address handlerAddresses,
    bytes calldata encodedCalls
  ) public onlyOwner returns (bool isSuccess, bytes memory data) {
    (isSuccess, data) = execAndReturnData(handlerAddresses, 0,
      ↪ encodedCalls, Enum.Operation.Call);
    if (!isSuccess) revert ErrorLibrary.CallFailed();
  }
```

SHB.18 Missing Array Length Check

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

The contract has an issue in its _addFeed function where it does not check if the lengths of the input arrays base, quote, and aggregator are equal. This can result in a revert of the transaction if the aggregator array is shorter than the base or quote arrays, or it can result in skipping elements from the longest array if the base or quote arrays are longer than the aggregator array.

Files Affected:

SHB.18.1: PriceOracle.sol

```
function _addFeed(
  address[] memory base,
  address[] memory quote,
  AggregatorV2V3Interface[] memory aggregator
) public onlyOwner {
  for (uint256 i = 0; i < base.length; i++) {
    if (aggregatorAddresses[base[i]].aggregatorInterfaces[quote[i]] !=
       ↪  AggregatorInterface(address(0))) {
      revert AggregatorAlreadyExists();
    }
    aggregatorAddresses[base[i]].aggregatorInterfaces[quote[i]] =
       ↪ aggregator[i];
  }
  emit addFeed(block.timestamp, base, quote, aggregator);
}
```

Recommendation:

To mitigate this issue, it is recommended to add a check at the beginning of the _addFeed function to ensure that the lengths of the base, quote, and aggregator arrays are equal. If they are not equal, the function should revert with an appropriate error message. This will prevent the function from being called with arrays of unequal lengths, ensuring that all elements are processed correctly.

Updates

The team resolved the issue by adding a check to the _addFeed function to ensure that the lengths of the base, quote, and aggregator arrays are equal.

SHB.18.2: PriceOracle.sol

```
function _addFeed(
  address[] memory base,
  address[] memory quote,
  AggregatorV2V3Interface[] memory aggregator
) public onlyOwner {
  if (!((base.length == quote.length) && (quote.length == aggregator.
      ↪ length)))
      revert ErrorLibrary.IncorrectArrayLength();

  for (uint256 i = 0; i < base.length; i++) {
      if (aggregatorAddresses[base[i]].aggregatorInterfaces[quote[i]] !=
        ↪  AggregatorInterface(address(0))) {
        revert AggregatorAlreadyExists();
      }
    aggregatorAddresses[base[i]].aggregatorInterfaces[quote[i]] =
        ↪ aggregator[i];
  }
  emit addFeed(block.timestamp, base, quote, aggregator);
}
```

SHB.19 Missing Maximum Amount for User Supplied Slippage

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The contract has an issue in its getSlippage function in the PancakeSwapHandler contract where it does not check for a maximum value of slippage other than 100% (DIVISOR_INT). This could potentially lead to users setting an excessively high slippage, which could result in unfavorable swaps.

Files Affected:

SHB.19.1: PancakeSwapHandler.sol

```
function getSlippage(
  uint256 _amount,
  uint256 _slippage,
  address[] memory path
) internal view returns (uint256 minAmount) {
  if (!(_slippage < DIVISOR_INT)) {
    revert ErrorLibrary.SlippageCannotBeGreaterThan100();
  }
  uint256 currentAmount;
  if (path[0] == getETH()) {
    currentAmount = oracle.getPriceForAmount(path[1], _amount, false);
  } else if (path[1] != getETH()) {
    currentAmount = oracle.getPriceForTokenAmount(path[0], path[1],
       ↪ _amount);
  } else {
    currentAmount = oracle.getPriceForAmount(path[0], _amount, true);
  }
  minAmount = currentAmount.mul(DIVISOR_INT.sub(_slippage)).div(
     ↪ DIVISOR_INT);
}
```

Recommendation:

To mitigate this issue, it is recommended to add a check in the getSlippage function to ensure that the user-supplied slippage is less than a maximum amount. This maximum amount should be set to a reasonable value to protect users from setting an excessively high slippage.

Updates

The team resolved the issue by adding a safety maxSlippage to ensure that the user-supplied slippage is reasonable to protect from sandwich attacks.

SHB.19.2: PancakeSwapHandler.sol

```
function getSlippage(
  uint256 _amount,
  uint256 _slippage,
  address[] memory path
) internal view returns (uint256 minAmount) {
  if (!(_slippage < DIVISOR_INT)) {
    revert ErrorLibrary.SlippageCannotBeGreaterThan100();
  }
  if (_slippage > maxSlippage) {
    revert ErrorLibrary.InvalidSlippage();
  }
  uint256 currentAmount;
  if (path[0] == getETH()) {
    currentAmount = oracle.getPriceForAmount(path[1], _amount, false);
  } else if (path[1] != getETH()) {
    currentAmount = oracle.getPriceForTokenAmount(path[0], path[1],
      ↪ _amount);
  } else {
    currentAmount = oracle.getPriceForAmount(path[0], _amount, true);
  }
  minAmount = (currentAmount * (DIVISOR_INT - _slippage)) / (
    ↪ DIVISOR_INT);
}
```

SHB.20 Potential Out of Gas Exception Due to Long _tokens Array

- Severity: LOW
- Status: Fixed

- Likelihood: 1
- Impact: 2

Description:

The contract has an issue in its initToken and updateTokenList functions where it does not limit the length of the _tokens array when it is initialized or updated. This could potentially lead to an Out of Gas (OOG) exception if the _tokens array becomes excessively long. Therefore, a Denial of Service for all the functionalities of the protocol.

Files Affected:

SHB.20.1: IndexSwap.sol

```
function initToken(address[] calldata tokens, uint96[] calldata
  ↪ denorms) external virtual onlySuperAdmin {
  if (tokens.length != denorms.length) {
    revert ErrorLibrary.InvalidInitInput();
  }
  if (_tokens.length != 0) {
    revert ErrorLibrary.AlreadyInitialized();
  }
  uint256 totalWeight = 0;
  for (uint256 i = 0; i < tokens.length; i++) {
    address token = tokens[i];
    uint96 _denorm = denorms[i];
    IndexSwapLibrary._beforeInitCheck(IIndexSwap(address(this)), token
      ↪ , _denorm);
    _records[token] = Record({lastDenormUpdate: uint40(block.timestamp
      ↪ ), denorm: _denorm, index: uint256(i)});
    _tokens.push(token);

    totalWeight = totalWeight.add(_denorm);
  }
  _weightCheck(totalWeight);
  emit LOG_PUBLIC_SWAP_ENABLED();
}
```

SHB.20.2: IndexSwap.sol

```
function updateTokenList(address[] calldata tokens) external virtual
    ↪ onlyRebalancerContract {
  _tokens = tokens;
}
```

Recommendation:

To mitigate this issue, it is recommended to add a check in the initToken and updateTokenList functions to ensure that the length of the _tokens array does not exceed a certain limit. This limit should be set to a reasonable value to prevent the array from becoming excessively long. If the length of the _tokens array exceeds this limit, the function should revert with an appropriate error message. This will prevent potential Out of Gas (OOG) exceptions and ensure that the investInFund function can be executed successfully.

Updates

The team resolved the issue by adding a limitation to the size of the _tokens array.

SHB.20.3: IndexSwap.sol

```
function initToken(address[] calldata tokens, uint96[] calldata
    ↪ denorms) external virtual onlySuperAdmin {
  if (tokens.length > _tokenRegistry.getMaxAssetLimit())
    revert ErrorLibrary.TokenCountOutOfLimit(_tokenRegistry.
        ↪ getMaxAssetLimit());
```

SHB.20.4: IndexSwap.sol

```
function updateTokenList(address[] calldata tokens) external virtual
    ↪ onlyRebalancerContract {
  uint256 _maxAssetLimit = _tokenRegistry.getMaxAssetLimit();
  if (tokens.length > _maxAssetLimit) revert ErrorLibrary.
      ↪ TokenCountOutOfLimit(_maxAssetLimit);
  _tokens = tokens;
}
```

SHB.21 Potential Failure of Off-Chain Investment Due to Disabled Tokens

- Severity: LOW
- Status: Fixed
- Likelihood: 1
- Impact: 2

Description:

The contract has an issue in its swapOffChainTokens function where it checks if the input token is enabled. If not, the function reverts. This check is performed for all the tokens in the _tokens array. Therefore, if any of the tokens are not enabled, the investment operation cannot proceed.

Files Affected:

SHB.21.1: OffChainIndexSwap.sol

```
(balanceInUSD, underlyingIndex) = exchange.swapOffChainTokens(
    ExchangeData.IndexOperationData(
      ExchangeData.InputData(
        inputData.buyAmount,
        inputData.sellTokenAddress,
        inputData._offChainHandler,
        inputData._buySwapData
      ),
      index,
      underlyingIndex,
      inputData.protocolFee[i],
      balanceInUSD,
      _lpSlippage[i],
      _buyAmount[i],
      _tokens[i],
      msg.sender
    )
);
```

SHB.21.2: Exchange.sol

```
function swapOffChainTokens(
  ExchangeData.IndexOperationData memory inputdata
) external virtual onlyIndexManager returns (uint256, uint256) {
  IndexSwapLibrary._whitelistAndHandlerCheck(inputdata._token,
      ↪ inputdata.inputData._offChainHandler, inputdata.index);
```

SHB.21.3: IndexSwapLibrary.sol

```
function _whitelistAndHandlerCheck(address _token, address
    ↪ _offChainHandler, IIndexSwap index) external {
  IAssetManagerConfig config = IAssetManagerConfig(index.
      ↪ iAssetManagerConfig());
  if ((config.whitelistTokens() && !config.whitelistedToken(_token)))
      ↪ {
    revert ErrorLibrary.TokenNotWhitelisted();
  }
  ITokenRegistry registry = ITokenRegistry(index.tokenRegistry());
  if (!(registry.isExternalSwapHandler(_offChainHandler))) {
    revert ErrorLibrary.OffHandlerNotValid();
  }
  if (!(registry.isEnabled(_token))) {
    revert ErrorLibrary.TokenNotEnabled();
  }
}
```

Recommendation:

To mitigate this issue, it is recommended to either remove the check for whether the token is enabled in the swapOffChainTokens function or ensure that all tokens in the _tokens array are enabled. This will prevent the swapOffChainTokens function from reverting due to disabled tokens and ensure that users can invest as intended.

Updates

The team resolved the issue by removing the whitelist check from the _swapOffChainTokens function.

SHB.22 Potential Unrestricted Withdrawals During Pause State

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The contract has an issue in its triggerMultipleTokenWithdrawal function where it allows users to withdraw funds even when the protocol is paused. While the withdrawOffChain function has the notPaused modifier and checks in the tokenRegistry if the protocol is paused, the triggerMultipleTokenWithdrawal function does not perform these checks.

Files Affected:

SHB.22.1: OffChainIndexSwap.sol

```
function withdrawOffChain(ExchangeData.ZeroExWithdraw memory inputData
  ↪ ) external virtual nonReentrant notPaused {
  address user = msg.sender;
  address withdrawToken = userWithdrawData[user].withdrawToken;
```

SHB.22.2: OffChainIndexSwap.sol

```
function triggerMultipleTokenWithdrawal() external nonReentrant {
  // Check if the user has redeemed their tokens
  if (!userWithdrawData[msg.sender].userRedeemedStatus) {
    revert ErrorLibrary.TokensNotRedeemed();
  }
```

Recommendation:

To mitigate this issue, it is recommended to add the notPaused modifier to the triggerMultipleTokenWithdrawal function and include a check to verify if the protocol is paused. If the protocol is paused, the function should revert with an appropriate error message. This will ensure that withdrawals cannot be made during a pause state, maintaining the integrity of the protocol's operations.

Updates

The team has resolved the issue by removing the notPaused modifier from the withdrawOffChain function to ensure a consistent behavior between the withdrawal functions.

SHB.23 Precision Loss When Dividing Odd Integers by Two

- Severity : LOW
- Status : Fixed
- Likelihood : 2
- Impact : 1

Description:

The contract has a flaw where it may lose precision when dividing odd integers by two. This is because in Solidity, integer division is floor division, meaning that the result of the division operation will be the largest integer less than or equal to the exact result. Therefore, when an odd integer is divided by two, the result will be rounded down, leading to a loss of precision.

Files Affected:

SHB.23.1: Exchange.sol

```
uint256 swapValue = underlying.length > 1 ? inputData._swapAmount.div(2)
    ↪ : inputData._swapAmount;
```

SHB.23.2: Exchange.sol

```
function getSwapVaule(uint256 len, uint256 amount) internal pure returns
  ↪ (uint256) {
  return (len > 1 ? amount.div(2) : amount);
}
```

SHB.23.3: Exchange.sol

```
function validateAmount(uint256 expectedAmount, uint256 userAmount,
  ↪ uint256 len) internal pure {
  uint256 PERCENTIn18Decimal = 10 ** 22;
  uint256 diff = expectedAmount.div(len).mul(PERCENTIn18Decimal).div(
    ↪ userAmount);
  uint256 diffPercentage = diff < PERCENTIn18Decimal ?
    ↪ PERCENTIn18Decimal.sub(diff) : diff.sub(PERCENTIn18Decimal);
  if (diffPercentage > PERCENTIn18Decimal) {
    revert ErrorLibrary.InvalidBuyValues();
  }
}
```

Recommendation:

When dividing an amount by two, consider taking the first amount as the division result by two, and the second one to be the total amount minus the first one.

Updates

The team resolved the issue by considering the first amount as the division result and the second one as the rest.

SHB.24 Lack of Cross-Contract Reentrancy Protection

- Severity : INFORMATIONAL
- Status : Fixed
- Likelihood : 1
- Impact : 0

Description:

The contract has an issue in its triggerMultipleTokenWithdrawal function where it lacks protection against cross-contract reentrancy attacks. While the investInFund, withdrawFund, investInFundOffChain, and redeemTokens functions in the IndexSwap and OffChainIndexSwap contracts have individual reentrancy guards, there is no single reentrancy guard spanning the two contracts.

Files Affected:

SHB.24.1: IndexSwap.sol

```
contract IndexSwap is Initializable, ERC20Upgradeable,
    ↪ ReentrancyGuardUpgradeable, UUPSUpgradeable, OwnableUpgradeable {
```

SHB.24.2: OffChainIndexSwap.sol

```
contract OffChainIndexSwap is Initializable, OwnableUpgradeable,
    ↪ UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

Recommendation:

To mitigate this issue, it is recommended to implement a single reentrancy guard that spans both the IndexSwap and OffChainIndexSwap contracts. This will ensure that reentrancy attacks cannot be made across the two contracts if the logic ever gets updated to be vulnerable to reentrancy attacks.

Updates

The team has resolved the issue by implementing a cross contract reentrancy guard using the CommonReentrancyGuard contract.

SHB.25 Off-Chain Investment Failure Due to Non-Zero Protocol Fees

- Severity : INFORMATIONAL
- Status : Fixed
- Likelihood : 1
- Impact : 0

Description:

In the OffChainIndexSwap contract, the investInFundOffChain function allows a user to pass a protocolFee array that signifies the fees to be paid to the protocol. However, if the user passes any value greater than 0, the investment operation will fail. This is because the function checks if the balance of the contract in Ether is less than the protocolFee and the swap call to the ZeroExHandler does not deposit any Ether, therefore the function reverts with an InsufficientFeeFunds error.

Files Affected:

SHB.25.1: ZeroExHandler.sol

```
function swap(
  address sellTokenAddress,
  address buyTokenAddress,
  uint256 sellAmount,
  uint256 protocolFee,
  bytes memory callData,
  address _to
) public payable {
  uint256 tokenBalance = IERC20Upgradeable(sellTokenAddress).balanceOf
      ↪ (address(this));
  if (tokenBalance < sellAmount) {
    revert ErrorLibrary.InsufficientFunds(tokenBalance, sellAmount);
  }
  uint256 ethBalance = address(this).balance;
  if (ethBalance < protocolFee) {
    revert ErrorLibrary.InsufficientFeeFunds(ethBalance, protocolFee);
  }

  setAllowance(sellTokenAddress, swapTarget, sellAmount);

  uint256 tokensBefore = IERC20Upgradeable(buyTokenAddress).balanceOf(
      ↪ address(this));
  (bool success, ) = swapTarget.call{value: protocolFee}(callData);
  if (!success) {
    revert ErrorLibrary.SwapFailed();
  }
```

Recommendation:

Consider requiring the protocolFee to be equal to zero.

Updates

The team has resolved the issue by removing the unused protocolFee parameter.

4 Best Practices

BP.1 Remove Unnecessary Initializations

Description:

The smart contract unnecessarily initializes variables with their default values. In Solidity, variables are automatically initialized with their default values (e.g., 0 for integers, false for booleans, etc.) when they are declared. Explicitly initializing these variables with their default values is redundant and can lead to unnecessary gas costs and code complexity. It is recommended to remove the unnecessary initializations of variables with their default values.

Files Affected:

- IndexFactory.sol
- IndexSwap.sol
- Exchange.sol
- IndexSwapLibrary.sol
- OffChainIndexSwap.sol
- AbstractLPHandler.sol
- SlippageControl.sol
- OneInchHandler.sol
- ParaswapHandler.sol
- ZeroExHandler.sol
- RebalanceLibrary.sol
- Rebalancing.sol
- AssetManagerConfig.sol

Status - Fixed

BP.2 Ommit Unnecessary Approval of Contract to Its Own Address

Description:

The contract unnecessarily approves the contract to its own address. It grants the contract an allowance of _amount tokens from its own balance. However, a contract already has the ability to transfer its own tokens without needing to grant itself an allowance. This unnecessary approval can lead to confusion and potential misuse. It is recommended to remove the unnecessary approval of the contract to its own address.

Files Affected:

BP.2.1: IndexSwap.sol

```
TransferHelper.safeApprove(_token, address(this), _amount);
```

Status - Fixed

BP.3 Unnecessary Use of SafeMath & SafeMathUpgradeable Libraries

Description:

The smart contract unnecessarily uses the SafeMath and SafeMathUpgradeable libraries for arithmetic operations. Starting from Solidity version 0.8.0, the language has built-in overflow and underflow protection, making the use of these libraries redundant. This can lead to unnecessary gas costs and code complexity. It is recommended to remove the use of the SafeMath and SafeMathUpgradeable libraries and rely on Solidity's built-in overflow and underflow protection for arithmetic operations. This can be done by simply performing arithmetic operations normally, without using the SafeMath or SafeMathUpgradeable functions. This will reduce gas costs and simplify the contract's code.

Files Affected:

All contracts that use SafeMath or SafeMathUpgradeable.

Status - Fixed

BP.4 Remove Unused Ether Call

Description:

In the IndexSwap contract, the investInFund function sends msg.value (the amount of Ether sent with the function call) to the exchange contract, and the exchange contract never returns any of it back to the IndexSwap contract. As a result, address(this).balance (the balance of the IndexSwap contract) will always be zero at the end of the function call, unless someone sent Ether directly to the contract through the receive function. The last lines of the function, which check the contract's balance and sends it back to the user, are therefore unnecessary and can be removed.

Files Affected:

BP.4.1: IndexSwap.sol

```
    // refund leftover ETH to user
    (bool success, ) = payable(_to).call{value: address(this).balance
      ↪ }("");
    // require(success, "Transfer ETH failed");
    if (!success) {
      revert ErrorLibrary.ETHTransferFailed();
    }
```

Status - Fixed

BP.5 Redundant External Call in OffChainIndexSwap Contract

Description:

In the OffChainIndexSwap contract, the _getTokenBalanceInBNB function makes an external call to the getPriceTokenUSD18Decimals function of the oracle contract. However, the return value of this call is not used in the function. This is a redundant operation that consumes unnecessary gas and can be removed.

To improve the efficiency of the contract, it is recommended to remove the redundant external call to getPriceTokenUSD18Decimals. This will reduce the gas cost of the _getTokenBalanceInBNB function and make the contract code cleaner and easier to understand.

Files Affected:

BP.5.1: OffChainIndexSwap.sol

```
function _getTokenBalanceInBNB(
    address _token,
    uint256 _tokenAmount
  ) internal view returns (uint256 tokenBalanceInBNB) {
    oracle.getPriceTokenUSD18Decimals(_token, _tokenAmount);
    uint256 tokenBalanceInUSD = oracle.getPriceTokenUSD18Decimals(_token
        ↪ , _tokenAmount);
    tokenBalanceInBNB = oracle.getUsdEthPrice(tokenBalanceInUSD);
  }
```

Status - Fixed

BP.6 Inefficient Loop in _swapTokenToTokens Function

Description:

In the _swapTokenToTokens function, the vault address is retrieved in each iteration of the loop. This is inefficient as it consumes unnecessary gas. The vault address does not change during the loop execution, so it can be retrieved once before the loop starts and then used in each iteration.

Files Affected:

BP.6.1: Exchange.sol

```
function _swapTokenToTokens(
    FunctionParameters.SwapTokenToTokensData memory inputData
  ) external payable virtual onlyIndexManager returns (uint256
     ↪ investedAmountAfterSlippage) {
    IIndexSwap _index = IIndexSwap(inputData._index);
    address[] memory _tokens = _index.getTokens();
    for (uint256 i = 0; i < _tokens.length; i++) {
      address vault = _index.vault();
      address _token = _tokens[i];
      uint256 swapAmount = getSwapAmount(
        inputData._totalSupply,
        inputData._tokenAmount,
        inputData.amount[i],
        uint256(_index.getRecord(_token).denorm)
      );
```

Status - Fixed

BP.7 Redundant Check in Weight Calculation

Description:

In the loop where weights are being calculated, there is a redundant check for weightToSwap being equal to zero. This check is unnecessary because it is already ensured that _newWeights[i] is greater than _oldWeights[i], which means weightToSwap will always be greater than zero.

Files Affected:

BP.7.1: Rebalancing.sol

```
for (uint256 i = 0; i < tokens.length; i++) {
      if (_newWeights[i] > _oldWeights[i]) {
        uint256 weightToSwap = _newWeights[i].sub(_oldWeights[i]);
        if (weightToSwap == 0) {
          revert ErrorLibrary.WeightNotGreaterThan0();
        }
```

Status - Fixed

BP.8 Remove Unused Variables and Events

Description:

Throughout the codebase, there are several instances where variables or events are declared but never used. This can lead to confusion for developers reading the code and can potentially waste gas when the contract is deployed. Some examples would be the RewardTokensDistributed event and the rewardTokens.

Files Affected:

BP.8.1: Exchange.sol

```
event RewardTokensDistributed(address indexed _index, address indexed
    ↪ _rewardToken, uint256 indexed diff);
```

BP.8.2: TokenRegistry.sol

```
  struct TokenRecord {
    bool primary;
    bool enabled;
    address handler;
    address[] rewardTokens;
  }
```

Status - Fixed

5 Tests

Results:

→ Tests running for Handler: Venus

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: Venus

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: Alpaca

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

```
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: Alpaca
✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: BiSwap
✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: BiSwap
✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: ApeSwap-lending

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: ApeSwap-lending

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: ApeSwap-lp

✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: ApeSwap-lp
✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: BeefyFinance
✓ should lend tokens
✓ return values of deposit should be greater than 0
✓ should redeem tokens
✓ gets underlying asset of the token
✓ should get token balance of the token holder
✓ should get the token price in USD
→ Tests running for Handler: BeefyFinance
```

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests running for Handler: BeefyFinanceLP

✓ should lend tokens

✓ return values of deposit should be greater than 0

✓ should redeem tokens

✓ gets underlying asset of the token

✓ should get token balance of the token holder

✓ should get the token price in USD

→ Tests for Mock Fee

✓ should revert back if the custodial is true and no address is passed in _owner

✓ should revert back if the _custodial is true and threshold is more than owner length

✓ Initialize 1st IndexFund Tokens

✓ Calculate fees should return fee values

✓ Invest 1BNB into Top10 fund

✓ Invest 2BNB into Top10 fund

✓ Invest 2BNB into Top10 fund

✓ Should charge fees for index 1

✓ Should charge fees for index 1

→ Tests for IndexFactory contract

✓ should revert back if the custodial is true and no address is passed in _owner

✓ should revert back if the _custodial is true and threshold is more than owner length

✓ asset manager should create a private transferable fund and make it non-transferable

✓ asset manager should be able to make the previous private fund transferable to whitelisted addresses

✓ asset manager should be able to convert the previous transferable private fund to public

✓ asset manager should be able to make the previous public fund non-transferable

✓ asset manager should not be able to make the previous public fund transferable to only whitelisted addresses

✓ asset manager should be able to make the previous public fund transferable

✓ should check Index token name and symbol

✓ should check if module owner of all fund is exchange contract

✓ initialize should revert if total Weights not equal 10,000

✓ initialize should revert if tokens and denorms length is not equal

✓ initialize should revert if token not whitelisted

✓ Initialize 1st IndexFund Tokens

✓ Initialize 2nd IndexFund Tokens

✓ Initialize 3rd IndexFund Tokens

✓ Initialize 4th IndexFund Tokens

✓ Owner of vault for 1st fund should be exchangeHandler address

✓ Owner of vault for 2nd fund should be deployer's addressess

✓ Owner of vault for 3rd fund should be exchangeHandler address

✓ Owner of vault for 4th fund should be exchangeHandler address

✓ Calculate fees should return fee values

✓ expect owner to be IndexFactory

✓ Invest 0.1BNB into Top10 fund should fail for slippage greater than 10

✓ Invest 0.1BNB into Top10 fund

✓ Invest 2BNB into Top10 2nd index fund

✓ Invest 0.1BNB into Top10 3rd index fund

✓ Invest 0.1BNB into Top10 3rd index fund

✓ Invest 2BNB into Top10 4th index fund

- ✓ Invest 2BNB into Top10 4th index fund
- ✓ Invest 2BNB into Top10 4th index fund should revert if bnb value is greater than 0 and investment token is not bnb
- ✓ Invest 2BNB into Top10 4th index fund on behalf of addr3 should fail if user addr3 is not whitelisted
- ✓ Add addr3 whitelisted user
- ✓ Invest 2BNB into Top10 4th index fund on behalf of addr3
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Add addr1 whitelisted user
- ✓ non owner should not be able to add whitelist manager admin
- ✓ owner should be able to add asset whitelist manager admin
- ✓ owner should not be able to add index manager
- ✓ owner should not be able to add rebalancing manager
- ✓ non whitelist manager admin should not be able to add asset manager
- ✓ new whitelist manager admin should be able to add whitelist manager
- ✓ owner should be able to add whitelist manager
- ✓ non whitelist manager should not be able to update merkle root

- ✓ Whitelist manager should be able to update merkle root
- ✓ Whitelist manager should be able to add and remove a whitelisted user
- ✓ non whitelist manager admin should not be able to revoke whitelist manager
- ✓ whitelist manager admin should be able to revoke whitelist manager
- ✓ Whitelist manager should not be able to add user to whitelist after his role was revoked
- ✓ New (addr1) whitelisted user invest 2BNB into Top10 2nd index fund
- ✓ New (addr2) whitelisted user invest 2BNB into Top10 2nd index fund
- ✓ Non whitelisted user invest 2BNB into Top10 2nd index fund should fail
- ✓ Should charge fees for index 1
- ✓ Should charge fees for index 2
- ✓ Management fees for index 3 should be 0
- ✓ Invest 0.00001 BNB into Top10 fund should fail
- ✓ asset manager should be able to add token which is approved in registry for all the indexes
- ✓ Invest 2BNB into Top10 fund
- ✓ Invest 1BNB into Top10 2nd Index fund
- ✓ Invest 1BNB into Top10 fund
- ✓ Invest 1BNB into Top10 2nd Index fund
- ✓ Investment should fail when contract is paused

✓ update Weights should revert if total Weights not equal 10,000

✓ Update Weights and Rebalance should revert if one of the weight is zero

✓ should Update Weights and Rebalance

✓ should Update Weights and Rebalance for 2nd Index Fund

✓ should Update Weights and Rebalance for 2nd Index Fund

✓ should Update Weights and Rebalance

✓ should Update Weights and Rebalance

✓ updateTokens should revert if total Weights not equal 10,000

✓ updateTokens should revert if token is not whitelisted

✓ updateTokens should revert if token is not enabled

✓ updateTokens should revert if protocol is paused

✓ updateTokens should revert if swapHandler is not enabled

✓ Non Rebalancing access address calling update function

✓ update tokens should revert is any two tokens are same

✓ should update tokens

✓ print values

✓ should update tokens

✓ withdrawal should revert when contract is paused

✓ should unpause

✓ should pause

✓ should revert unpause

✓ should unpause

✓ should update tokens for 2nd Index

✓ when withdraw fund more then balance

✓ should fail withdraw when balance falls below min investment amount

✓ should fail withdraw when balance falls below min investment amount

✓ should withdraw fund and burn index token successfully

✓ should withdraw fund and burn index token successfully

✓ should withdraw fund and burn index token successfully for account that has been removed from whitelist

✓ Invest 0.1BNB into Top10 2nd Index fund

✓ transfer idx for a non transferable portfolio should fail

✓ transfer idx from owner to non whitelisted account should fail

✓ transfer idx from owner to a whitelisted account

✓ transfer idx from owner to another account (Index 3)

✓ transfer idx from owner to another account (Index 4)

✓ new owner of idx withdraws funds from Index 3

✓ Invest 1BNB into Top10 fund after last withdrawal

✓ withdraw check values

✓ new owner of idx withdraws funds from Index 4

✓ should withdraw fund and burn index token successfully for 2nd Index

✓ should withdraw fund and burn index token successfully for account that received idx

✓ Invest 2BNB into Top10 fund

✓ Invest 0.1BNB into Top10 fund

✓ Invest 0.1BNB into Top10 2nd Index fund

✓ Invest 0.1BNB into Top10 2nd Index fund

✓ should withdraw tokens directly instead of BNB

✓ should withdraw tokens directly instead of BNB for 2nd Index

✓ non owner should not be able to add asset manager admin

✓ owner should be able to add asset manager admin

✓ non asset manager admin should not be able to add asset manager

✓ new asset manager admin should be able to add asset manager

✓ owner should be able to add asset manager

✓ non-owner should be able to pause protocol

✓ should not upgrade Proxy Exchnage To New Contract for 1st Index

✓ should protocol pause

✓ should upgrade Proxy Exchnage To New Contract for 1st Index and 2nd Index

✓ should not upgrade if msg.sender is not owner

✓ non owner of indexFactory should not be able to upgrade Exchange

✓ should upgrade Proxy IndexSwap To New Contract for 1st Index

✓ should upgrade Proxy OffChainIndexSwap To New Contract for 1st Index

✓ should unpause protocol

✓ Invest 2BNB into Top10 1st index fund after upgrade

✓ Invest 2BNB into Top10 1st index fund after upgrade

✓ should pause protocol

✓ should upgrade Proxy IndexSwap To New Contract for 2nd Index

✓ should unpause protocol

✓ Invest 2BNB into Top10 2nd index fund after upgrade

✓ Upgrade TokenRegistry

✓ Upgrade IndexFactory, and not able to create Index

✓ should unpause index creation and creat index

✓ should set new cool down period

✓ Invest 2BNB into Top10 2nd index fund after upgrade

✓ Invest 1BNB into Top10 2nd index fund after upgrade and should no revert

✓ should withdraw fund and burn index token successfully should fail

✓ transfer tokens should fail, if cooldownperiod is not passed

✓ should transfer token and withdraw fund and burn index token successfully

- ✓ should fail to create an index with management fee greater than max fee
- ✓ should fail to create an index with management fee greater than max fee
- ✓ Non asset manager should not be able to propose new management fee
- ✓ Asset manager should propose new management fee
- ✓ Asset manager should not be able to update management fee before 28 days passed
- ✓ Non asset manager should not be able to delete proposed new management fee
- ✓ Asset manager should be able to delete proposed new management fee
- ✓ Non asset manager should not be able to update management fee
- ✓ Non asset manager should not be able to propose new performance fee
- ✓ Asset manager should propose new performance fee
- ✓ Asset manager should be able to update performance fee before 28 days passed
- ✓ Non asset manager should not be able to delete proposed new performance fee
- ✓ Asset manager should be able to delete proposed new performance fee
- ✓ Non asset manager should not be able to update performance fee
- ✓ Non asset manager should not be able to update the asset manager treasury

✓ Asset manager should not be able to update the asset manager treasury

✓ Non asset manager should not be able to update the velvet treasury

✓ Asset manager should be able to update the velvet treasury

✓ Non owner should not be able to update protocol slippage

✓ Owner should not be able to update to a slippage more than 10

✓ Owner should not be able to update protocol slippage

→ Tests for MixedIndex - Mixed Protocols

✓ should check Index token name and symbol

✓ initialize should revert if total Weights not equal 10,000

✓ Initialize should fail if the number of tokens exceed the max limit set during deployment (current = 15)

✓ should retrieve the current max asset limit from the TokenRegistry

✓ should update the max asset limit to 10 in the TokenRegistry

✓ should retrieve the current max asset limit from the TokenRegistry

✓ Initialize should fail if the number of tokens exceed the max limit set by the Registry (current = 10)

✓ Initialize IndexFund Tokens

✓ should add pid

✓ should remove pid

- ✓ asset manager should not be able to add token which is not approved in registry
- ✓ Invest 0.16 BNB should not revert , if investing token is not initialized
- ✓ Invest 10BUSD should revert , if investing token is not initialized
- ✓ asset manager should be able to add token which is approved in registry
- ✓ Invest 0.1BNB into Top10 fund should fail if LP slippage is invalid
- ✓ Invest 0.1BNB into Top10 fund
- ✓ Invest 10BUSD into Top10 fund
- ✓ Invest 0.00001 BNB into Top10 fund should fail
- ✓ Invest 2BNB into Top10 fund
- ✓ should return false if both of the token in pool is not bnb
- ✓ Invest 1BNB into Top10 fund
- ✓ Investment should fail when contract is paused
- ✓ update Weights should revert if total Weights not equal 10,000
- ✓ should Update Weights and Rebalance
- ✓ updateTokens should revert if total Weights not equal 10,000
- ✓ owner should be able to add asset manager
- ✓ non owner should not be able to add asset manager
- ✓ new asset manager should update tokens
- ✓ withdrawal should revert when contract is paused

✓ should unpause

✓ should pause

✓ should revert unpause

✓ should unpause

✓ when withdraw fund more then balance

✓ should fail withdraw when balance falls below min investment amount

✓ should fail withdraw when balance falls below min investment amount (multi asset)

✓ should withdraw fund and burn index token successfully

✓ Invest 0.1BNB into Top10 fund

✓ Invest 0.1BNB into Top10 fund

✓ Invest 1BNB into Top10 fund

✓ Invest 1BNB into Top10 fund

✓ should withdraw fund in ETH and burn index token successfully

✓ Invest 0.1BNB into Top10 fund

✓ Invest 0.1BNB into Top10 fund

✓ should withdraw tokens directly instead of BNB

→ Tests for MixedIndex - Mixed Contracts

✓ should check Index token name and symbol

✓ initialize should revert if tokens length does not match denorms length

- ✓ initialize should revert if a token address is null
- ✓ initialize should revert if a non-approved token is being used for init
- ✓ initialize should revert if total Weights not equal 10,000
- ✓ Initialize IndexFund Tokens
- ✓ Initialize 2nd IndexFund Tokens
- ✓ should confirm that the correct tokens are initialised
- ✓ should confirm that the correct tokens are initialised
- ✓ non-admin should not be able to call the access control setupRole function
- ✓ admin should be able to call the access control setupRole function
- ✓ should update a price Oracle feed
- ✓ should not be able to obtain the decimals of a token pair price feed where aggregator is zero address
- ✓ should not be able to add pid if arrray lengths don't match
- ✓ should not be able to delete pid if array lengths don't match
- ✓ should add pid
- ✓ should delete pid
- ✓ should fetch the router address of the pancake LP handler
- ✓ should get the swap address from the pancake swap handler
- ✓ should check if a token is enabled or not in the registry
- ✓ should disable a token in the registry

- ✓ should reiterate the WETH address of the token registry
- ✓ should not be able to enable a zero address permitted token in Token-Registry
- ✓ should not be able to enable if empty array is passed to TokenRegistry
- ✓ should not be able to enable a token which is already enabled
- ✓ should not be able to enable token in registry if the oracle array length does not match the length of other arrays
- ✓ should not be able to enable token in registry if the token array length does not match the length of other arrays
- ✓ should not be able to enable token in registry if the handler array length does not match the length of other arrays
- ✓ should not be able to enable token in registry if the reward token array length does not match the length of other arrays
- ✓ should not be able to enable token in registry if the reward token array length does not match the length of other arrays
- ✓ disable token in registry should fail if zero address is passed
- ✓ disable token in registry should fail if token is not enabled at all
- ✓ disable token in registry should fail if empty array is passed
- ✓ should disable a permitted token in TokenRegistry
- ✓ isPermitted function from TokenRegistry should not return output for zero address
- ✓ should update an enabled token's data in the TokenRegistry

- ✓ Non-primary tokens should not get enabled on the registry level
- ✓ asset manager should not be able to add token which is not approved in registry
- ✓ asset manager should not be able to delete a zero address as permitted token
- ✓ asset manager should not be able to delete a non-permitted token
- ✓ asset manager should not be able to delete permitted tokens if an empty array is passed
- ✓ isTokenPermitted should not return output for asset manager config
- ✓ Invest 0.1 BNB should not revert, if investing token is not initialized
- ✓ Invest 0.1 BNB in 2nd index
- ✓ Invest 1 BNB in 2nd index
- ✓ Invest 10BUSD should not revert, if investing token is not initialized
- ✓ asset manager should be able to permit token which is approved in registry
- ✓ should not be able to get underlying of a zero address Wombat lp token
- ✓ should not be able to get token balance of a zero address Wombat lp token
- ✓ should not be able to get token balance of a zero address Wombat lp token holder
- ✓ should not be able to get underlying balance of a zero address Wombat lp token

- ✓ should not be able to get underlying balance of a zero address Wombat lp token holder
- ✓ should not be able to get token balance of a zero address Alpaca token
- ✓ should not be able to get underlying token of a zero address Alpaca token
- ✓ should not be able to get underlying balance of a zero address Alpaca token holder
- ✓ should not be able to get underlying balance of a zero address Alpaca token
- ✓ should not be able to get underlying token of a zero address Beefy token
- ✓ should not be able to get token balance of a zero address Beefy token
- ✓ should not be able to get underlying balance of a zero address Beefy moo token
- ✓ should not be able to get underlying balance of a zero address Beefy moo token holder
- ✓ should be able to get underlying balance of a Beefy LP token
- ✓ should not be able to get underlying token of a non-Venus token via the Venus handler
- ✓ should not be able to get underlying balance of a zero address Venus token
- ✓ should not be able to get underlying balance of a zero address Venus token holder
- ✓ should not be able to get token balance of a zero address Venus token

- ✓ should not be able to get token balance of a zero address Venus token holder
- ✓ should not be able to get underlying token of a zero address Venus token
- ✓ should add reward token to registry and verify it
- ✓ should remove reward token from registry and verify it
- ✓ should add reward token to registry and verify it
- ✓ should revert when add reward token to registry sending 0 address token address
- ✓ should revert when add reward token to registry sending 0 address handler address
- ✓ Invest 10BUSD into Top10 fund
- ✓ Invest 0.00001 BNB into Top10 fund should fail
- ✓ Invest 10BNB into Top10 fund
- ✓ Invest 10BNB into Top10 fund
- ✓ Investment should fail when contract is paused
- ✓ should be able to claim tokens for portfolio tokens
- ✓ update Weights should revert if total Weights not equal 10,000
- ✓ update weights should revert if weights and slippage array length don't match
- ✓ update weights should revert if slippage array length don't match the token count

- ✓ update weights should revert if swap handler is not enabled
- ✓ should Update Weights and Rebalance
- ✓ should Update Weights and Rebalance
- ✓ should Update Weights and Rebalance
- ✓ updateTokens should revert if total Weights not equal 10,000
- ✓ owner should be able to add asset manager
- ✓ non owner should not be able to add asset manager
- ✓ disable swaphandler in registry should not work if handler array length is 0
- ✓ disable swaphandler in registry should not work if the handler is already disabled
- ✓ update tokens should not work if the protocol is paused
- ✓ update tokens should not work if swaphandler is not enabled
- ✓ update tokens should not work if non-enabled token is being used
- ✓ new asset manager should update tokens
- ✓ withdrawal should revert when contract is paused
- ✓ should unpause
- ✓ should pause
- ✓ should revert unpause
- ✓ should unpause
- ✓ when withdraw fund more then balance

✓ should fail withdraw when slippage array length is not equal to index length

✓ should fail withdraw when balance falls below min investment amount

✓ should fail withdraw when balance falls below min investment amount (multi asset)

✓ should fail withdraw fund when the output token is not permitted in the asset manager config and is not WETH

✓ should fail withdraw when the protocol is paused

✓ should withdraw fund and burn index token successfully

✓ Invest 1BNB into Top10 fund

✓ should withdraw fund in BUSD and burn index token successfully

✓ Invest 1BNB into Top10 fund

✓ should withdraw tokens directly instead of BNB

→ Tests for OffChainIndex contract

✓ Initialize IndexFund Tokens

✓ should add pid

✓ Initialize 2nd IndexFund Tokens

✓ Invest 1 BNB into 1st fund

✓ Invest 2 BNB into Top10 2nd fund

✓ Invest 2 BNB into Top10 2nd fund

✓ Invest 51.8 BUSD in 1st Index fund

✓ Invest 1 BUSD in 1st Index fund should fail (under min amount)

✓ Invest 50 DOGE in 1st Index fund

✓ Invest 50 DOGE in 2nd Index fund

✓ Invest 50 DOGE should fail, if user input is incorrect in 2nd Index fund

✓ Invest 1 ETH should fail if user has sent wrong input in 2nd Index fund

✓ Invest 1 ETH should fail if user tries to manipulate weight in 2nd Index

✓ Invest 1 ETH should fail if user has sent wrong input in 1st Index fund

✓ Invest 1 ETH should fail if user tries to manipulate weight

✓ Invest 0.01 BTC in 1st Index fund

✓ Invest 1 BNB into 1st Top10 fund

✓ Invest 10 BUSD in 2nd Index fund

✓ Invest 0.1 BNB in 2nd Index fund

✓ Invest 1 BNB into 1st Top10 fund

✓ redeem should fail if a non-permitted and non-WETH token is passed as the out asset

✓ should withdraw properly with rebalance in between

✓ Invest 1 BNB into 1st Top10 fund

✓ should revert if sellToken address length is manupilated and trigger-multiple withdrawal

✓ Invest 1 BNB into 1st Top10 fund

✓ should Update Weights and Rebalance for 2nd Index

- ✓ Invest 2 BNB in 2nd Index fund
- ✓ Invest 2 BNB in 1st Index fund
- ✓ should fail if offchainHandler is not valid
- ✓ Invest 1 BNB in 1st Index fund should revert if bnb value is greater than 0 and investment token is not bnb
- ✓ withdraw should fail if user balance falls below min amount
- ✓ should withdraw fund and burn index token successfully for 1st Index ,Simultaneously for both user
- ✓ addr2 should invest using offchain
- ✓ addr2 should emergency withdraw
- ✓ owner should invest using offchain
- ✓ TriggerMultiple TokenWithdrawal withdraw should fail is protocol is paused and work if protocol is unpaused
- ✓ Non owner should not triggerMultiple TokenWithdrawal withdraw
- ✓ Invest 1 BNB into 1st Top10 fund
- ✓ Withdraw and triggerMultipleWithdrawal should fail if the protocol is paused

→ Tests for priceOracle contract

- ✓ should revert if aggregator is already added
- ✓ should revert if base array length does not match the length of other arrays

- ✓ should revert if quote array length does not match the length of other arrays
- ✓ should revert if quote array length does not match the length of other arrays
- ✓ Get ETH/WBNB price
- ✓ Get BTC/ETH price
- ✓ Get BUSD/WBNB price
- ✓ Get BTC/USD price
- ✓ Get BTC/USD price
- ✓ Get ETH/USD price
- ✓ Get BUSD/USD price
- ✓ Get DAI/USD price
- ✓ Get WBNB/USD price
- ✓ Get DOGE/USD price
- ✓ Get USD/WBNB price
- ✓ Get BTC/WETH price
- ✓ Get WETH/BTC price
- ✓ Get ETH/WETH price
- ✓ Get WETH/ETH price
- ✓ Get DOGE/WETH price
- ✓ Get WETH/DOGE price

- ✓ Get USD/DOGE price
- ✓ Get DOGE/wbnb price
- ✓ Get wbnb/DOGE price
- ✓ Get doge/wbnb price
- ✓ Get wbnb/doge price
- ✓ Get DOGE price in 18 decimals
- ✓ Get BUSD price in 18 decimals
- ✓ Get ETH price in 18 decimals
- ✓ Get BTC price in 18 decimals
- ✓ Get WBNB_BUSD price in 18 decimals
- ✓ Get CAKE_BUSD price in 18 decimals
- ✓ Get CAKE_WBNB price in 18 decimals
- ✓ Get ADA_WBNB price in 18 decimals
- ✓ Get BAND_WBNB price in 18 decimals
- ✓ Get DOT_WBNB price in 18 decimals
- ✓ Get DOGE_WBNB price in 18 decimals
- ✓ Get BSWAP_WBNB_BUSD price in 18 decimals
- ✓ Get BSWAP_BUSDT_BUSD price in 18 decimals
- ✓ Get BSWAP_BUSDT_WBNB price in 18 decimals
- ✓ Get BSWAP_ETH_BTC price in 18 decimals

✓ Get BSWAP_BTC_WBNB price in 18 decimals

✓ Get BSWAP_DOGE_WBNB price in 18 decimals

✓ Get APESWAP_WBNB_BUSD price in 18 decimals

✓ Get APESWAP_ETH_BTCB price in 18 decimals

✓ Get APESWAP_ETH_WBNB price in 18 decimals

✓ Get APESWAP_USDT_WBNB price in 18 decimals

✓ Get APESWAP_DOGE_WBNB price in 18 decimals

✓ owner updates the oracleTimeout to 35 hours

✓ non owner should not be able to update oracleTimeout

→ Tests for MetaAggregator

✓ Initialize 1st IndexFund Tokens

✓ Initialize 2nd IndexFund Tokens

✓ Initialize 3rd IndexFund Tokens

✓ Initialize 4th IndexFund Tokens

✓ Initialize 5th IndexFund Tokens

✓ Initialize 6th IndexFund Tokens

✓ Initialize 7th IndexFund Tokens

✓ Initialize 8th IndexFund Tokens

✓ Invest 0.1BNB into Top10 fund

✓ Invest 0.1BNB into 5th fund

✓ Invest 1BNB into 6th fund

✓ Invest 2BNB into index fund

✓ Invest 2BNB into index fund

✓ Invest 2BNB into index fund

✓ Invest 1BNB into Top10 fund

✓ Invest 1BNB into Top10 2nd Index fund

✓ Invest 1BNB into 7th Index fund

✓ Invest 1BNB into 8th index fund

✓ should revert back if swapHandler is not enabled

✓ swaps using 1Inch Protocol

✓ revert redeem

✓ non assetManager should not revert if 15 minutes is not passed

✓ non assetManager should revert if 15 minutes is passed

✓ redeems token for 0x

✓ swaps reverts if token address is wrong

✓ swaps reverts if sellAmount is wrong

✓ swaps reverts if sellAmount is wrong in calldata

✓ swaps reverts if sellAddress is wrong in calldata

✓ swaps using 0x Protocol

✓ swaps using Paraswap Protocol

- ✓ should revert back if the calldata includes fee and the overall slippage is more than 1
- ✓ Invest 2BNB into index fund
- ✓ should revert back if the calldata includes fee and the overall slippage is more than 1
- ✓ should revert back if the calldata includes fee and the overall slippage is more than 1
- ✓ update external handler slippage should fail if value is greater than MAX_SLIPPAGE
- ✓ should update external handler slippage
- ✓ should set max slippage as 0 and disabling slippage checks
- ✓ Swaps directly to protocol token WBNB and ETH
- ✓ Swaps directly to protocol token ERC20
- ✓ Swaps WBNB directly to protocol token ERC20
- ✓ Swaps WBNB directly to derivative protocol token ERC20
- ✓ Invest 0.1BNB into Top10 fund
- ✓ swaps into primary using ZeroEx Protocol from primary
- ✓ swaps into derivative token using oneInch Protocol from primary
- ✓ swaps into derivative using ZeroEx Protocol from primary
- ✓ swaps into lp token reverts if sellAmount is not equal using ZeroEx Protocol from primary

- ✓ swaps into lp token using ZeroEx Protocol from primary
- ✓ Direct Swap reverts if passed underlying token length more than 1
- ✓ Direct Swap reverts if underlying is not same
- ✓ Direct Swap reverts if length of tokens are not same
- ✓ Direct Swap reverts if length of tokens and sellAmount are not same
- ✓ redeem should revert back if index not paused
- ✓ should pause
- ✓ redeem should revert back if token getting redeem is not valid
- ✓ should revert back if the buy token is not registered
- ✓ should revert back if not redeemed
- ✓ should revert back if redeem is called by non asset manager
- ✓ should revert back if metaAggregatorSwap is called by non asset manager
- ✓ Invest 1BNB into Top10 fund

→ Tests for Time Dependent contract

- ✓ Initialize 1st IndexFund Tokens
- ✓ Initialize 2nd IndexFund Tokens
- ✓ Initialize 3rd IndexFund Tokens
- ✓ Initialize 4th IndexFund Tokens
- ✓ Invest 1BNB into Top10 fund

- ✓ Invest 2BNB into Top10 2nd index fund
- ✓ Invest 1BNB into Top10 3rd index fund
- ✓ Invest 2BNB into Top10 4th index fund
- ✓ should revert if the price did not updated for more than 25 hours
- ✓ should revert if the price did not updated for more than 25 hours
- ✓ should update threshold of the oracle
- ✓ Asset manager should propose new management fee
- ✓ Asset manager should propose new management fee
- ✓ Asset manager should be able to update management fee after 28 days passed
- ✓ Asset manager should be able to update management fee after 28 days passed
- ✓ should claim tokens
- ✓ should swap reward token using pancakeSwap Handler into derivative token
- ✓ should claim tokens
- ✓ should swap reward token using pancakeSwap Handler into LP token
- ✓ should claim tokens
- ✓ swaps reward token should fail using 0x Protocol if buyToken is not In-dexToken
- ✓ swaps reward token using 0x Protocol

✓ should claim tokens

✓ should swap reward token using pancakeSwap Handler into WETH base token

✓ should claim tokens

✓ should swap reward token using pancakeSwap Handler into base token

→ Tests for ZeroEx contract

✓ Initialize IndexFund Tokens

✓ should add pid

✓ should check if off chain handler is enabled or not

✓ Initialize 2nd IndexFund Tokens

✓ Invest 1 BNB into Top10 fund

✓ Invest 1 BNB into Top10 fund

✓ Invest 1 BNB in first index fund

✓ Should disable external swap handler

✓ update weights should fail if any one weight is zero

✓ update weights should fail if sum of weight is not 10000

✓ Update Weights

✓ print values after updating weights to [1000, 2000, 7000]

✓ should _revert after enable Rebalance(1st Transaction)

✓ should _revert after externalSell (2nd Transaction)

✓ should update weights

✓ Invest 1 BNB into Top10 fund

✓ Invest 1 BNB into Top10 fund

✓ Should not update tokens if tokens is not approved

✓ Should not update tokens if tokens is not whitelisted

✓ Should not update if any one weight is zero

✓ Should not update if weight is not equal to 10000

✓ print values before

✓ Should Update Tokens

✓ print values after

✓ should fail to revert back if all transaction is completed

✓ non assetManager should not be able to update portfolio to new tokens

✓ should update portfolio to new tokens

✓ should update tokens

✓ Invest 1 BNB into Top10 fund

✓ Invest 1 BNB into Top10 fund

✓ Invest 1 BNB into Top10 fund

✓ Should add one more token

✓ print values after adding one more token ([3000, 1000, 2000, 4000])

✓ Invest 1 BNB into Top10 fund

✓ Should remove one token

✓ Invest 1 BNB into Top10 fund

✓ Should Update Tokens and replace two tokens for vETH and MAIN_LP_BUSD

✓ Invest 1 BNB into Top10 fund

✓ should fail if we call wrong revert function

✓ non-assetManager should revert if 15minutes of Pause is passed

✓ non-assetManager should not be able revert if 15minutes of Pause is not passed

✓ it should fail if assetmanager tries to execute 3rd transacton after 1st

621 passing

Coverage:

The code coverage results were obtained by running npx hardhat coverage in the project. We found the following results :

- Statements Coverage : 96.57%
- Branches Coverage : 73.81%
- Functions Coverage : 89.74%
- Lines Coverage : 90.43%

6 Conclusion

In this audit, we examined the design and implementation of Velvet Capital V2 contract and discovered several issues of varying severity. Velvet Capital team addressed 22 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Velvet Capital Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

7 Scope Files

7.1 Audit

Files	MD5 Hash
contracts/FunctionParameters.sol	74d5b94e912ff4c250210e558c03ae9a
contracts/IndexFactory.sol	c4114ce0e631695e49c6db37cadc368d
contracts/vault/VelvetSafeModule.sol	94841e20705158e25da0d06fd66af055
contracts/registry/AssetManagerConfig.sol	bbea36e152a646641605a8d97989584b
contracts/registry/TokenRegistry.sol	df7da244af7e32cdd41bc3590bad4f45
contracts/rebalance/OffChainRebalance.sol	45713b873d1bcd7576c3c218d8ea9202
contracts/rebalance/RebalanceAggregator.sol	105a31d7cf2011b25ba9a4be87058848
contracts/rebalance/RebalanceLibrary.sol	0e4e63027abe3c59853b8cad8469f164
contracts/rebalance/Rebalancing.sol	e948a43187c14d24f3db04f44a668099
contracts/oracle/IPriceOracle.sol	3e946f5f6a22f548cbdb4ab94e38f249
contracts/oracle/PriceOracle.sol	aef16301361093574cc54c0d918358e1
contracts/library/ErrorLibrary.sol	244da0dea43f8dd80e510379d6c6a69f
contracts/library/GnosisDeployer.sol	0263333d87c831a53fbd302a8c69a487
contracts/handler/AbstractLPHandler.sol	6f0efcf1602c14353eead7c2abf4af9b
contracts/handler/ApproveControl.sol	9461a1be702de46fb4046e6eea6d2a83
contracts/handler/BaseHandler.sol	c78c5785a88e02861a9dbfb98f8e4ba0
contracts/handler/DustHandler.sol	57796d364f26541b19df6ce8d1316f72

contracts/handler/ExternalSlippageControl.sol	b8203912308b95a4beaeb8e80a38805a
contracts/handler/PancakeSwapHandler.sol	4911d88d3d4df6a72d253d8517367c1b
contracts/handler/SlippageControl.sol	7bb4e0a60b64191e18d453f5b2a7e485
contracts/handler/Wombat/WombatHandler.sol	709f905fbb139307dd6882538c581186
contracts/handler/venus/VenusHandler.sol	86239e0790e0345f98257078efec9ca4
contracts/handler/PancakeSwapLP/PancakeSwapLPHandler.sol	0ce767d16ba34711087f385a59e677c2
contracts/handler/libraries/FullMath.sol	1c9d54bfd986d35524095efb0c41f610
contracts/handler/ExternalSwapHandler/OneInchHandler.sol	deae9f6b0e8276bf2a17ee38fc479fd7
contracts/handler/ExternalSwapHandler/ParaswapHandler.sol	3bf4623632f6428a5170eb594d724559
contracts/handler/ExternalSwapHandler/ZeroExHandler.sol	708ca75564f785b38166b110ac90b5c6
contracts/handler/ExternalSwapHandler/Helper/ExchangeData.sol	f372f110cc29b9f254836aecb4d1eec8
contracts/handler/BiSwapLP/BiSwapLPHandler.sol	1204291b2a91a2b6e4319d2dd257c359
contracts/handler/Beefy/BeefyHandler.sol	65c07c6f1de8d5ed0626a0a7d9ff96d2
contracts/handler/Beefy/BeefyLPHandler.sol	9dbe7dc7f4dd3b8822e5f4de4ca79bf3
contracts/handler/ApeSwap/ApeSwapLendingHandler.sol	071c7e3ea2f86cbceb56949d14eccade
contracts/handler/ApeSwap/ApeSwapLPHandler.sol	fbf6dfb96a40e89978755594b45d4227
contracts/handler/alpaca/AlpacaHandler.sol	a526de406badf02a5fbc5e7d66060954
contracts/fee/FeeLibrary.sol	c347feaa59aa2977c5cf1603bd6cd58d

contracts/fee/FeeModule.sol	fcccde1d2d57283b08d7c7bda9344318
contracts/core/Exchange.sol	212559cff900fc936166b444d8082795
contracts/core/IndexSwap.sol	f9dda9f817fc6dec0ada382f2485c322
contracts/core/IndexSwapLibrary.sol	4e7cd179b5a336d5a93a010336658163
contracts/core/OffChainIndexSwap.sol	c84884e133dc787813b36e1d8bf1df02
contracts/access/AccessController.sol	a9523257273d905f54b09e89167f4502

7.2 Re-Audit

Files	MD5 Hash
contracts/FunctionParameters.sol	dbaf59b3bf9760eeb80df2900842e9b4
contracts/IndexFactory.sol	a99092164c72673fd6ade6f8832a05c9
contracts/vault/VelvetSafeModule.sol	b98fbee2e8e6e69dcde20ba7c1cf2486
contracts/registry/AssetManagerConfig.sol	c20158049ddfbbd5a801e6a5783a6ad9
contracts/registry/TokenRegistry.sol	b32d8aa507a2003fda8e231cc44a23e7
contracts/rebalance/OffChainRebalance.sol	6e5e576f731bc20bb0afe5ddd78ed284
contracts/rebalance/RebalanceAggregator.sol	f8bee57f68898b878ea5e8dff589bb55
contracts/rebalance/RebalanceLibrary.sol	c2592def84f96455bf5abe96b6c17dfc
contracts/rebalance/Rebalancing.sol	b9e8e0b740d1d48fd88026a18167c7a7
contracts/oracle/PriceOracle.sol	b799722c01a41b3738ee0d35baeffc19
contracts/oracle/aggregators/AggregatorV3Interface.sol	c09b2fc2eb6637f1159df7787b9ee342

contracts/oracle/aggregators/UniswapV2LPAggregator.sol	b1113349cd57bd71df5c1d25819b068c
contracts/library/ErrorLibrary.sol	f1183752d271003baa403fc2e88ceaf0
contracts/library/GnosisDeployer.sol	0263333d87c831a53fbd302a8c69a487
contracts/handler/AbstractLPHandler.sol	6d31149c968acba2570f37b4b2d4ccf2
contracts/handler/ApproveControl.sol	9461a1be702de46fb4046e6eea6d2a83
contracts/handler/BaseHandler.sol	57cef964e92d2125936c455897686d3d
contracts/handler/DustHandler.sol	b66055a0fba610a319207a8e6a9b42be
contracts/handler/ExternalSlippageControl.sol	3600b32af41ac1538fb2dc3841b17c0c
contracts/handler/PancakeSwapHandler.sol	119e217c53498de17be466989c171041
contracts/handler/SlippageControl.sol	aa6ab9d8729140c50c6f3547d5a2d0f1
contracts/handler/Wombat/WombatHandler.sol	8ab07965ba2c7d4de2d7ef4148077aeb
contracts/handler/venus/VenusHandler.sol	ef2ab64047244703901500a04ad9598d
contracts/handler/PancakeSwapLP/PancakeSwapLPHandler.sol	04dc3f9b2ed413f67bdf480fc6e43ed8
contracts/handler/libraries/FullMath.sol	ae17c1a9e0c2a3dab384e0ec6df61744
contracts/handler/ExternalSwapHandler/OneInchHandler.sol	0951a3becf608996e4e1d2552318bd40
contracts/handler/ExternalSwapHandler/ParaswapHandler.sol	401207b58791b7d9318eea6494a21f21
contracts/handler/ExternalSwapHandler/ZeroExHandler.sol	52ef9299fac218a9372dcac098c80fc3
contracts/handler/ExternalSwapHandler/Helper/ExchangeData.sol	3fc39414cc12f40bbc03443577846691

contracts/handler/BiSwapLP/BiSwapLPHandler.sol	12d5307b9d0a766d6946d7d2feb8329f
contracts/handler/BiSwapLP/interfaces/IMasterChef.sol	81a12127050bb962576b465abee2cb61
contracts/handler/Beefy/BeefyHandler.sol	498fe730e82fe4c5364294233470c777
contracts/handler/Beefy/BeefyLPHandler.sol	092bebf2b962ad82018d94196abb92bd
contracts/handler/ApeSwap/ApeSwapLendingHandler.sol	5b92c60f32092e8627f53c3386fc5501
contracts/handler/ApeSwap/ApeSwapLPHandler.sol	a107e7f490a07a6dc8e72d3c9f807bb6
contracts/handler/alpaca/AlpacaHandler.sol	2700ad95469cc35248cf0a41b47c3a36
contracts/fee/FeeLibrary.sol	d062a72c0425b3538c09e8469904d2c8
contracts/fee/FeeModule.sol	838f9806008bcbd2ea8b995e028e19b4
contracts/core/CommonReentrancyGuard.sol	4f08a48517b1fe6064f52bb1270c8d45
contracts/core/Exchange.sol	d444d5bdbbf1487a3bd7658ab3deddec
contracts/core/IndexSwap.sol	a1e85cee61d9dd5c81b7cc68931a41fc
contracts/core/IndexSwapLibrary.sol	4f11a2629fe9aaacee68a669b9e16010
contracts/core/OffChainIndexSwap.sol	8176291c936290f6f29ddffc13846f08
contracts/access/AccessController.sol	9ebb52b030cac6a92cd628edccdbc9eb

8 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com