CERTIK

PROFESSIONAL SERVICES

Fetch.ai

Security Assessment

December 22nd, 2020

For :

Fetch.ai Market Maker Contracts

By :

Camden Smallwood @ CertiK

camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Fetch.ai: Market Maker Contracts
Description	Controller for non-finalized Balancer pool AMM that updates weights in response to underlying asset price change. Vault contract maintains the spot price of each commodity which is maintained with the help of Oracle
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 3a937c995a5eef2c9428fe286fbf8d3fccfa900c 2. 393f841555b85aea3eb26ebaf7e90844c7553bf2

Audit Summary

Delivery Date	Dec. 22, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 31, 2020 - Dec. 11, 2020

Vulnerability Summary

Total Issues	30	*(29 Resolved)*
● **Total Medium**	4	*(4 Resolved)*
● **Total Minor**	9	*(9 Resolved)*
● **Total Informational**	17	*(16 Resolved)*

Executive Summary

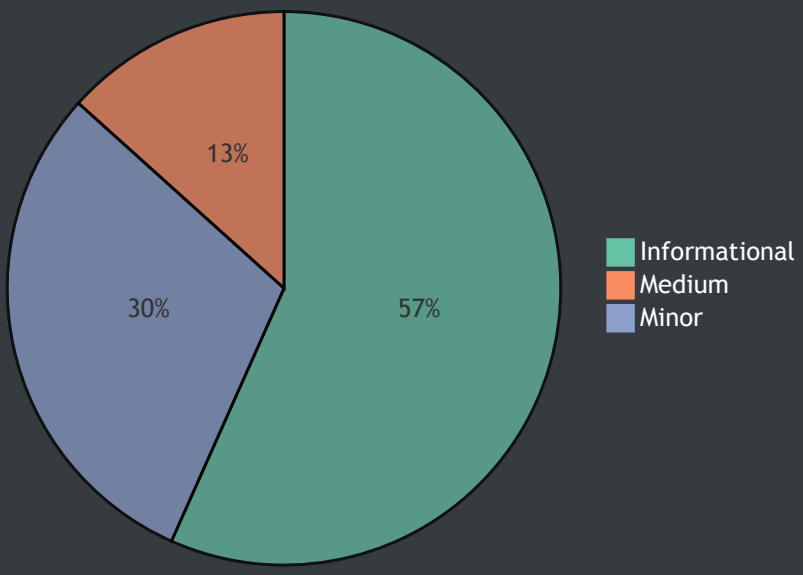
The codebase was found to be well-written, but had duplicate `Address` and `SafeERC20` library implementations, did not verify if all addresses were valid across their domain (non-zero, not the address of the containing contract), and had areas which could result in integer overflow and underflow.

While the codebase was being audited, the Fetch.ai team created another `StrategyBalancerMettalexV4` contract which still contains the issues from the `StrategyBalancerMettalexV3` contract which were addressed and resolved in this report. We recommend that the Fetch.ai team apply the same fixes where available to the new `StrategyBalancerMettalexV4` contract in order to prevent the possibility of regressions.

Files In Scope

ID	File
SEV	on-chain/mettalex-vault/contracts/libraries/SafeERC20.sol
VAU	on-chain/mettalex-vault/contracts/vault/Vault.sol
ADR	on-chain/pool-controller/contracts/lib/Address.sol
SEP	on-chain/pool-controller/contracts/lib/SafeERC20.sol
SMP	on-chain/pool-controller/contracts/lib/SafeMath.sol
SSM	on-chain/pool-controller/contracts/lib/SignedSafeMath.sol
SBM	on-chain/pool-controller/contracts/strategy/StrategyBalancerMettalexV3.sol

Findings



ID	Title	Type	Severity	Resolved
SEV-01	Duplicate SafeERC20 library	Implementation	Informational	✓
VAU-01	Addresses not verified in constructor	Implementation	Minor	✓
VAU-02	Addresses not verified in claimFee function	Implementation	Minor	✓
VAU-03	Addresses not verified in updateOracle function	Implementation	Minor	✓
VAU-04	Addresses not verified in	Implementation	Minor	✓

	updateAMMPoolController function			
VAU-05	Potential overflow in bulkSettlePositions function	Arithmetic	Medium	✓
VAU-06	Potential integer truncation in _calculateFeeAndPositions	Implementation	Medium	✓
VAU-07	Inefficient _settle function implementation	Performance	Informational	ⓘ
ADR-01	Duplicate Address library	Implementation	Informational	✓
SEP-01	Duplicate SafeERC20 library	Implementation	Informational	✓
SMP-01	Duplicate SafeMath library	Implementation	Informational	✓
SBM-01	Non-conforming callOnce function implementation	Implementation	Informational	✓
SBM-02	Comparison to bool constant in deposit function	Implementation	Informational	✓
SBM-03	Unnecessary supply calculation in deposit function	Implementation	Informational	✓
SBM-04	Comparison to bool constant in withdraw amount function	Implementation	Informational	✓
SBM-05	Comparison to bool constant in withdraw token function	Implementation	Informational	✓
SBM-06	Comparison to bool constant in handleBreach	Implementation	Informational	✓
SBM-07	Address not verified in setGovernance function	Implementation	Minor	✓
SBM-08	Address not verified in setController function	Implementation	Minor	✓

SBM-09	setBreaker function should be declared external	Implementation	Informational	✓
SBM-10	updatePoolController function should be declared external	Implementation	Informational	✓
SBM-11	Address not verified in updatePoolController function	Implementation	Minor	✓
SBM-12	getExpectedOutAmount function should be declared external	Implementation	Informational	✓
SBM-13	Potential underflow/division-by-0 in getExpectedOutAmount	Arithmetic	Medium	✓
SBM-14	Potential underflow/division-by-0 in getExpectedInAmount	Implementation	Medium	✓
SBM-15	Unlabeled magic numbers in _calcDenormWeights function	Implementation	Informational	✓
SBM-16	Inefficient array element swapping in _sortAndRebind	Implementation	Informational	✓
SBM-17	Ignoring result of call to IERC20.transfer function	Implementation	Minor	✓
SBM-18	Ignoring result of call to IERC20.transfer function	Implementation	Minor	✓
SBM-19	Comparison to bool constant in _depositInternal	Implementation	Informational	✓

SEV-01: Duplicate SafeERC20 library

Type	Severity	Location
Implementation	Informational	SafeERC20.sol

Description:

- The project contains a duplicate implementation of the `SafeERC20` library, which is unnecessary.

Recommendation:

- Since the project makes use of `@openzeppelin/contracts`, consider removing the `on-chain/mettalex-vault/contracts/libraries/SafeERC20.sol` file in favor of importing the `@openzeppelin/contracts/token/ERC20/SafeERC20.sol` file.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-01: Addresses not verified in constructor

Type	Severity	Location
Implementation	Minor	Vault.sol L107-L111

Description:

- The `constructor` of the `Vault` contract does not verify if the supplied addresses are non-0 or unique from each other.

Recommendation:

- Consider adding requirements in order to verify that the supplied addresses are non-0 and unique from each other:

```
require(_collateralToken != address(0));
require(_longPosition != address(0));
require(_shortPosition != address(0));
require(
  (_collateralToken != _longPosition) &&
  (_collateralToken != _shortPosition) &&
  (_longPosition != _shortPosition)
);
require(_oracleAddress != address(0));
require(_ammPoolController != address(0));
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-02: Addresses not verified in claimFee function

Type	Severity	Location
Implementation	Minor	Vault.sol L160-L170

Description:

- The `claimFee` function in the `Vault` contract does not verify that the supplied `_to` address is non-0 or the address of the `Vault` contract:

```
function claimFee(address _to) external onlyOwner {
  uint256 claimedCollateral = feeAccumulated;
  feeAccumulated = 0;
  IToken(collateralToken).safeTransfer(_to, claimedCollateral);
  emit FeeClaimed(_to, claimedCollateral);
}
```

Recommendation:

- Consider adding a requirement to the `claimFee` function in order to verify that the supplied `_to` address is non-0 and not the address of the `Vault` contract:

```
require(
  (_to != address(0)) && (_to != address(this)),
  "invalid to address"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-03: Addresses not verified in updateOracle function

Type	Severity	Location
Implementation	Minor	Vault.sol L186-L193

Description:

- The `updateOracle` function in the `Vault` contract does not verify if the supplied `_newOracle` address is non-0 or the address of the `Vault` contract:

```
function updateOracle(address _newOracle) external onlyOwner {
  emit OracleUpdated(oracle, _newOracle);
  oracle = _newOracle;
}
```

Recommendation:

- Consider adding a requirement to the `updateOracle` function in order to verify that the supplied `_newOracle` address is non-0 and not the address of the `Vault` contract:

```
require(
  (_newOracle != address(0)) && (_newOracle != address(this)),
  "invalid oracle address"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-04: Addresses not verified in updateAMMPoolController function

Type	Severity	Location
Implementation	Minor	Vault.sol L195-L206

Description:

- The `updateAMMPoolController` function in the `Vault` contract does not verify if the supplied `_newAMMPoolController` is non-0 or the address of the `Vault` contract:

```
function updateAMMPoolController(address _newAMMPoolController)
  external
  onlyOwner
{
  emit AMMPoolControllerUpdated(ammPoolController,
_newAMMPoolController);
  ammPoolController = _newAMMPoolController;
}
```

Recommendation:

- Consider adding a requirement to the `updateAMMPoolController` function in order to verify that the supplied `_newAMMPoolController` is non-0 and not the address of the `Vault` contract:

```
require(
  (_newAMMPoolController != address(0)) && (_newAMMPoolController !=
address(this)),
  "invalid amm pool controller"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-05: Potential overflow in bulkSettlePositions function

Type	Severity	Location
Arithmetic	Medium	Vault.sol L319-L321

Description:

- The `bulkSettlePositions` function in the `Vault` contract has the potential to overflow the `totalLongBurned`, `totalShortBurned` and `totalCollateralReturned` variables due to using primitive incrementation while looping over the supplied `_settlers` array:

```
totalLongBurned += longBurned;
totalShortBurned += shortBurned;
totalCollateralReturned += collateralReturned;
```

Recommendation:

- Since the `SafeMath` library is already imported in the `Vault` contract for `uint256` values, consider utilizing the `SafeMath.add` function in order to revert in the event of an overflow:

```
totalLongBurned = totalLongBurned.add(longBurned);
totalShortBurned = totalShortBurned.add(shortBurned);
totalCollateralReturned =
totalCollateralReturned.add(collateralReturned);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-06: Potential integer truncation in _calculateFeeAndPositions

Type	Severity	Location
Implementation	Medium	Vault.sol L366-L369

Description:

- The `_calculateFeeAndPositions` function in the `Vault` contract performs a multiplication on the result of a division, which can truncate. This affects the `mintFromCollateralAmount` function:

```
uint256 quantityToMint =
_collateralAmount.div(collateralWithFeePerUnit);
uint256 collateralFee = collateralFeePerUnit.mul(quantityToMint);
```

Recommendation:

- Consider ordering the multiplication before the division when calculating the collateral fee in order to prevent truncation:

```
uint256 collateralFee = _collateralAmount
  .mul(collateralFeePerUnit)
  .div(collateralWithFeePerUnit);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

VAU-07: Inefficient _settle function implementation

Type	Severity	Location
Performance	Informational	Vault.sol L383-L411

Description:

- The `_settle` function in the `Vault` contract takes `_collateral`, `_long` and `_short` parameters instead of using the `collateralToken`, `longPositionToken` and `shortPositionToken` state variables. It also returns the `longBalance`, `shortBalance` and `collateralReturned` local variables explicitly instead of using the function signature.

Recommendation:

- Consider removing the `_collateral`, `_long` and `_short` parameters in favor of the `collateralToken`, `longPositionToken` and `shortPositionToken` state variables, as well as naming the return values of the `_settle` function and removing the local variable declarations:

```
function _settle(
  address _settler
) private returns (
  uint256 longBalance, uint256 shortBalance, uint256 collateralReturned
) {
  longBalance = _long.balanceOf(_settler);
  shortBalance = _short.balanceOf(_settler);
  //...
}
```

Alleviation:

The recommendation was not applied.

ADR-01: Duplicate Address library

Type	Severity	Location
Implementation	Informational	Address.sol

Description:

- The project contains a duplicate implementation of the `Address` library, which is unnecessary.

Recommendation:

- Since the project makes use of `@openzeppelin/contracts-ethereum-package`, consider removing the `on-chain/pool-controller/contracts/lib/Address.sol` file in favor of importing the `@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol` file.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SEP-01: Duplicate SafeERC20 library

Type	Severity	Location
Implementation	Informational	SafeERC20.sol

Description:

- The project contains a duplicate implementation of the `SafeERC20` library, which is unnecessary.

Recommendation:

- Since the project makes use of `@openzeppelin/contracts-ethereum-package`, consider removing the `on-chain/pool-controller/contracts/lib/SafeERC20.sol` file in favor of importing the `@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol` file.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SMP-01: Duplicate SafeMath library

Type	Severity	Location
Implementation	Informational	SafeMath.sol

Description:

- The project contains a duplicate implementation of the `SafeMath` library, which is unnecessary.

Recommendation:

- Since the project makes use of `@openzeppelin/contracts-ethereum-package`, consider removing the `on-chain/pool-controller/contracts/lib/SafeMath.sol` file in favor of importing the `@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol` file.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-01: Non-conforming callOnce function implementation

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L98-L105

Description:

- The `callOnce` modifier in the `StrategyBalancerMettalexV3` states that it will "throw", but is wrapped in an `if` statement and does not actually revert:

```
modifier callOnce {
  if (!isBreachHandled) {
    _;
  }
}
```

Recommendation:

- Consider refactoring the `if` into a `require` statement in order to successfully revert if the breach has already been handled:

```
modifier callOnce {
  require(!isBreachHandled, "breach already handled");
  _;
}
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-02: Comparison to bool constant in deposit function

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L131

Description:

- The `deposit` function in the `StrategyBalancerMettalexV3` contract has a requirement that performs a comparison to a bool constant, which is unnecessary:

```
require(breaker == false, "!breaker");
```

Recommendation:

- Consider utilizing the NOT operator instead of comparing directly to a bool constant:

```
require(!breaker, "!breaker");
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-03: Unnecessary supply calculation in deposit function

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L133-L143

Description:

- The logic for calculating supply of the `deposit` function in the `StrategyBalancerMettalexV3` contract is unnecessary:

```
uint256 wantBeforeMintandDeposit =
IERC20(want).balanceOf(address(this));
```

```
uint256 wantAfterMintandDeposit =
IERC20(want).balanceOf(address(this));
supply =
supply.add(wantBeforeMintandDeposit.sub(wantAfterMintandDeposit));
```

Recommendation:

- Since the `supply` state variable is not actually used within the implementation of the `StrategyBalancerMettalexV3`, the supply calculation should be removed.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-04: Comparison to bool constant in withdraw amount function

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L154

Description:

- The `withdraw(uint256 _amount)` function in the `StrategyBalancerMettalexV3` contract has a requirement that performs a comparison to a bool constant, which is unnecessary:

```
require(breaker == false, "!breaker");
```

Recommendation:

- Consider utilizing the NOT operator instead of comparing directly to a bool constant:

```
require(!breaker, "!breaker");
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-05: Comparison to bool constant in withdraw token function

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L188

Description:

- The `withdraw(address _token)` function in the `StrategyBalancerMettalexV3` has a requirement that performs a comparison to a bool constant, which is unnecessary:

```
require(breaker == false, "!breaker");
```

Recommendation:

- Consider utilizing the NOT operator instead of comparing directly to a bool constant:

```
require(!breaker, "!breaker");
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-06: Comparison to bool constant in handleBreach

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L298

Description:

- The `handleBreach` function in the `StrategyBalancerMettalexV3` contract has a requirement that performs a comparison to a bool constant, which is unnecessary:

```
require(breaker == false, "!breaker");
```

Recommendation:

- Consider utilizing the NOT operator instead of comparing directly to a bool constant:

```
require(!breaker, "!breaker");
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-07: Address not verified in setGovernance function

Type	Severity	Location
Implementation	Minor	StrategyBalancerMettalexV3.sol L350-L353

Description:

- The `setGovernance` function in the `StrategyBalancerMettalexV3` contract does not verify if its supplied `_governance` address parameter is non-0 or the address of the `StrategyBalancerMettalexV3` contract.

Recommendation:

- Consider adding a requirement to the `setGovernance` function in order to verify that its supplied `_governance` address parameter is non-0 and not the address of the `StrategyBalancerMettalexV3` contract:

```
require(
  (_governance != address(0)) && (_governance != address(this)),
  "invalid governance address"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-08: Address not verified in setController function

Type	Severity	Location
Implementation	Minor	StrategyBalancerMettalexV3.sol L360-L363

Description:

- The `setController` function in the `StrategyBalancerMettalexV3` contract does not verify if its supplied `_controller` address parameter is non-0 or the address of the `StrategyBalancerMettalexV3` contract.

Recommendation:

- Consider adding a requirement to the `setController` function in order to verify that its supplied `_controller` address parameter is non-0 and not the address of the `StrategyBalancerMettalexV3` contract:

```
require(
  (_controller != address(0)) && (_controller != address(this)),
  "invalid controller address"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-09: setBreaker function should be declared external

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L380

Description:

- The `setBreaker` function in the `StrategyBalancerMettalexV3` contract should be declared external, as it is not used from within the `StrategyBalancerMettalexV3` itself:

```
    function setBreaker(bool _breaker) public {
```

Recommendation:

- Refactor the visibility of the `setBreaker` function from `public` to `external` :

```
    function setBreaker(bool _breaker) external {
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-10: updatePoolController function should be declared external

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L390

Description:

- The `updatePoolController` function in the `StrategyBalancerMettalexV3` contract should be declared external:

```
function updatePoolController(address _controller) public {
```

Recommendation:

- Refactor the visibility of the `updatePoolController` function from `public` to `external` :

```
function updatePoolController(address _controller) external {
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-11: Address not verified in updatePoolController function

Type	Severity	Location
Implementation	Minor	StrategyBalancerMettalexV3.sol L390-L395

Description:

- The `updatePoolController` function in the does not verify if its supplied `_controller` address parameter is non-0 or the address of the `StrategyBalancerMettalexV3` contract.

Recommendation:

- Consider adding a requirement to the `updatePoolController` function in order to verify that its supplied `_controller` address parameter is non-0 and not the address of the `StrategyBalancerMettalexV3` contract:

```
require(
  (_controller != address(0)) && (_controller != address(this)),
  "invalid controller address"
);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-12: getExpectedOutAmount function should be declared external

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L347

Description:

- The `getExpectedOutAmount` function in the `StrategyBalancerMettalexV3` contract should be declared external, as it is not used from within the `StrategyBalancerMettalexV3` itself:

```
function getExpectedOutAmount(
  address fromToken, address toToken, uint256 fromTokenAmount
) public view returns (
  uint256 tokensReturned, uint256 priceImpact
) {
```

Recommendation:

- Refactor the visibility of the `getExpectedOutAmount` function from `public` to `external` :

```
function getExpectedOutAmount(
  address fromToken, address toToken, uint256 fromTokenAmount
) external view returns (
  uint256 tokensReturned, uint256 priceImpact
) {
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-13: Potential underflow/division-by-0 in getExpectedOutAmount

Type	Severity	Location
Arithmetic	Medium	StrategyBalancerMettalexV3.sol L461

Description:

- The `getExpectedOutAmount` function in the `StrategyBalancerMettalexV3` contract has the potential for division-by-0 and underflow due to performing a primitive subtraction and division between the `effectivePrice` and `spotPrice` local variables:

```
priceImpact = ((effectivePrice - spotPrice) * 10**18) / spotPrice;
```

Recommendation:

- Since the `SafeMath` library is already imported in the `StrategyBalancerMettalexV3` contract for `uint256` values, consider using the `SafeMath` function equivalents in order to protect against division-by-0 and underflow:

```
priceImpact = effectivePrice.sub(spotPrice).mul(10**18).div(spotPrice);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-14: Potential underflow/division-by-0 in getExpectedInAmount

Type	Severity	Location
Implementation	Medium	StrategyBalancerMettalexV3.sol L501

Description:

- The `getExpectedInAmount` function in the `StrategyBalancerMettalexV3` contract has the potential for division-by-0 and underflow due to performing a primitive subtraction and division between the `effectivePrice` and `spotPrice` local variables:

```
priceImpact = ((effectivePrice - spotPrice) * 10**18) / spotPrice;
```

Recommendation:

- Since the `SafeMath` library is already imported in the `StrategyBalancerMettalexV3` contract for `uint256` values, consider using the `SafeMath` function equivalents in order to protect against division-by-0 and underflow:

```
priceImpact = effectivePrice.sub(spotPrice).mul(10**18).div(spotPrice);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-15: Unlabeled magic numbers in _calcDenormWeights function

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L594-L604

Description:

- The `_calcDenormWeights` function in the `StrategyBalancerMettalexV3` contract contains the use of unlabeled magic numbers `47` and `50`:

```
if (
  price.floor.add(x) >= price.spot || price.cap.sub(x) <= price.spot
) {
  wt[0] = wt[0].mul(47).add(1 ether);
  wt[1] = wt[1].mul(47).add(1 ether);
  wt[2] = wt[2].mul(47).add(1 ether);
} else {
  wt[0] = wt[0].mul(50);
  wt[1] = wt[1].mul(50);
  wt[2] = wt[2].mul(50);
}
```

Recommendation:

- Consider either creating named `constant` variables, or at least adding a comment to explain the usage of the magic numbers.

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-16: Inefficient array element swapping in _sortAndRebind

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L644-L700

Description:

- The `_sortAndRebind` function in the `StrategyBalancerMettalexV3` contracts performs inefficient array element swapping using temporary variables instead of tuple assignment swapping.

Recommendation:

- Consider using a tuple swap expression and removing the temporary variables in order to save on the overall cost of gas:

```
(delta[0], delta[1]) = (delta[1], delta[0]);
(balance[0], balance[1]) = (balance[1], balance[0]);
(wt[0], wt[1]) = (wt[1], wt[0]);
(tokens[0], tokens[1]) = (tokens[1], tokens[0]);
//etc
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-17: Ignoring result of call to IERC20.transfer function

Type	Severity	Location
Implementation	Minor	StrategyBalancerMettalexV3.sol L759

Description:

- The `_swapFromCoin` function in the `StrategyBalancerMettalexV3` function ignores the result of the call to `IERC20.transfer`:

```
IERC20(want).transfer(msg.sender, tokenAmountOut);
```

Recommendation:

- Since the `SafeERC20` library is already imported in the `StrategyBalancerMettalexV3`, consider using the `SafeERC20.safeTransfer` function to safely handle non-conforming ERC-20 token implementations:

```
IERC20(tokenOut).safeTransfer(msg.sender, tokenAmountOut);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-18: Ignoring result of call to IERC20.transfer function

Type	Severity	Location
Implementation	Minor	StrategyBalancerMettalexV3.sol L815

Description:

- The `_swapPositions` function in the `StrategyBalancerMettalexV3` function ignores the result of the call to `IERC20.transfer`:

```
IERC20(tokenOut).transfer(msg.sender, tokenAmountOut);
```

Recommendation:

- Since the `SafeERC20` library is already imported in the `StrategyBalancerMettalexV3`, consider using the `SafeERC20.safeTransfer` function to safely handle non-conforming ERC-20 token implementations:

```
IERC20(tokenOut).safeTransfer(msg.sender, tokenAmountOut);
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

SBM-19: Comparison to bool constant in _depositInternal

Type	Severity	Location
Implementation	Informational	StrategyBalancerMettalexV3.sol L893

Description:

- The `_depositInternal` function in the `StrategyBalancerMettalexV3` contract performs comparisons to bool constants, which is unnecessary:

```
if (isStkBound != true && isLtkBound != true && isWantBound != true) {
```

Recommendation:

- Consider utilizing the NOT operator instead of comparing directly to a bool constant:

```
if (!isStkBound && !isLtkBound && !isWantBound) {
```

Alleviation:

The recommendation was applied in commit 393f841555b85aea3eb26ebaf7e90844c7553bf2.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Arithmetic

Arithmetic exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.