CERTIK

Unslashed Dapp Insurance

Security Assessment

November 23rd, 2020

By :

Camden Smallwood @ CertiK

camden.smallwood@certik.org

Sheraz Arshad @ CertiK

sheraz.arshad@certik.org

Disclaimer

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 indeed 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	Unslashed Dapp Insurance
Description	Unslashed's Dapp Insurance smart contracts repository.
Platform	Ethereum; Solidity
Codebase	GitLab Repository
Commits	1. 93fcd6a533109255843b0566fc9f43f1ce6b2544

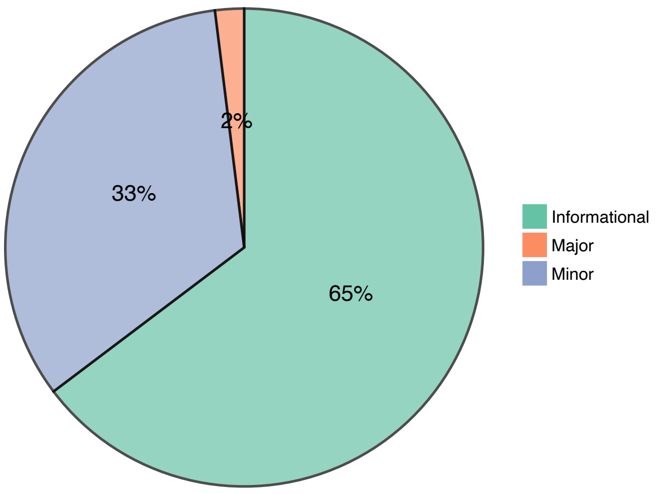
Audit Summary

Delivery Date	Nov. 23, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 12, 2020 - Oct. 21 2020

Vulnerability Summary

Total Issues	51	*47 Resolved, 4 Acknowledged*
Total Critical	-	
Total Major	1	*1 Resolved*
Total Minor	17	*16 Resolved, 1 Acknowledged*
Total Informational	33	*31 Resolved, 2 Acknowledged*

Findings



ID	Title	Type	Severity	Resolved
UDI-01	Potential for uint16 addition overflow	Arithmetic	Minor	✓
UDI-02	Potential for uint16 subtraction underflow	Arithmetic	Minor	✓
UDI-03	Potential for uint256 subtraction underflow	Arithmetic	Minor	✓
UDI-04	Public function should be declared external	Performance	Informational	✓
UDI-05	Public function should be declared external	Performance	Informational	✓
UDI-06	Potential for uint256 addition overflow	Arithmetic	Minor	✓
UDI-07	Inefficient `Basket.isMarketInBasket` implementation	Performance	Informational	✓
UDI-08	Public function should be declared external	Performance	Informational	✓
UDI-09	Inefficient `Basket.isPausedByMarket` implementation	Performance	Informational	✓
UDI-10	Imprecise arithmetic operations order	Arithmetic	Major	✓
UDI-11	Inefficient `Basket.getCollateralAvailableForCoverOptimized` implementation	Implementation	Informational	◷
UDI-12	Public function should be declared external	Performance	Informational	✓
UDI-13	Inefficient `Basket.getCurrentMarketCollateralLimitOptimized` implementation	Implementation	Minor	✓

UDI-14	Public function should be declared external	Performance	Informational	✓
UDI-15	Inefficient loop over array state variable	Performance	Informational	✓
UDI-16	Potential for uint256 addition overflow	Arithmetic	Minor	✓
UDI-17	Potential for uint256 addition overflow	Arithmetic	Minor	✓
UDI-18	Redundant function definition	Implementation	Informational	✓
UDI-19	Public function should be declared external	Performance	Informational	✓
UDI-20	Public function should be declared external	Performance	Informational	✓
UDI-21	Unnecessary usage of `SafeMath.add` function	Performance	Informational	✓
UDI-22	Inefficient loop over array parameter	Performance	Informational	✓
UDI-23	Potential for uint256 multiplication overflow	Arithmetic	Minor	✓
UDI-24	Inefficient loop over array parameter	Performance	Informational	✓
UDI-25	Inefficient `BulkDataGetter.calculateUserCollateral` implementation	Performance	Minor	✓
UDI-26	Inefficient `BulkDataGetter.filterMarketsForUserPremium` implementation	Performance	Informational	✓
UDI-27	Public function should be declared external	Performance	Informational	✓
UDI-28	Public function should be declared external	Performance	Informational	✓
UDI-29	Public function should be declared external	Performance	Informational	✓
UDI-30	Potential for uint256 subtraction underflow	Arithmetic	Minor	✓
UDI-31	Missing return statement leads to always returning false	Implementation	Minor	✓
UDI-32	Public function should be declared external	Performance	Informational	✓
UDI-33	Public function should be declared external	Performance	Informational	✓
UDI-34	Public function should be declared external	Performance	Informational	✓
UDI-35	Potential for uint256 addition overflow	Arithmetic	Minor	✓
UDI-36	Potential for uint256 subtraction underflow	Arithmetic	Minor	✓
UDI-37	Public function should be declared external	Performance	Informational	✓
UDI-38	Public function should be declared external	Performance	Informational	✓
UDI-39	Inefficient `Market.premiumToPay` implementation	Implementation	Informational	✓

UDI-40	Inefficient `Market.accountPayPremium` implementation	Implementation	Minor	✓
UDI-41	Unnecessary `Market._updateRolloverPrice` return type	Implementation	Informational	◷
UDI-42	Public function should be declared external	Performance	Informational	✓
UDI-43	Potential for uint256 addition overflow	Arithmetic	Minor	◷
UDI-44	Public function should be declared external	Performance	Informational	✓
UDI-45	Potential for uint256 subtraction underflow	Arithmetic	Minor	✓
UDI-46	Public function should be declared external	Performance	Informational	✓
UDI-47	Public function should be declared external	Performance	Informational	✓
UDI-48	Public function should be declared external	Performance	Informational	✓
UDI-49	Potential for uint256 addition overflow	Arithmetic	Minor	◷
UDI-50	Public function should be declared external	Performance	Informational	✓
UDI-51	Public function should be declared external	Performance	Informational	✓

UDI-01: Potential for uint16 addition overflow

Type	Severity	Location
Arithmetic	Minor	Basket.sol L112

Description:

The `pauseFromMarket` function in the `Basket` contract increments the `_marketPauseCounter` state variable using a primitive addition without checking the value for overflow:

```
_marketPauseCounter += 1;
```

Recommendation:

We recommended requiring the `_marketPauseCounter` to be less than the maximum uint16 value before performing the addition in order to prevent against overflow:

```
require(_marketPauseCounter < 0xFFFF, "Market pause counter overflow");
_marketPauseCounter += 1;
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-2: Potential for uint16 subtraction underflow

Type	Severity	Location
Arithmetic	Minor	Basket.sol L117

Description:

The `upauseFromMarket` function in the `Basket` contract decrements the `_marketPauseCounter` state variable using a primitive addition without checking the value for overflow:

```
_marketPauseCounter -= 1;
```

Recommendation:

We recommended requiring the `_marketPauseCounter` to be greater than zero before performing the subtraction in order to prevent against underflow:

```
require(_marketPauseCounter > 0, "Market pause counter underflow");
_marketPauseCounter -= 1;
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-3: Potential for uint256 subtraction underflow

Type	Severity	Location
Arithmetic	Minor	Basket.sol L183

Description:

The `withdrawCollateral` function in the `Basket` contract has a requirement that performs a primitive subtraction without verifying that the right-hand value is less than the left, which has the potential for underflow and to cause the requirement to incorrectly succeed:

```
require(block.timestamp - timestamp > WITHDRAW_COOLDOWN, 'Basket: too early to withdraw');
```

Recommendation:

We recommended extending the requirement to include that that `block.timestamp` global value should be greater than or equal to the supplied `timestamp` parameter value in order to safely protect against subtraction underflow:

```
require(
    (block.timestamp >= timestamp) && (block.timestamp - timestamp > WITHDRAW_COOLDOWN),
    'Basket: too early to withdraw'
);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-4: Public function should be declared external

Type	Severity	Location
Performance	Informational	Basket.sol L211

Description:

The `increaseUsedCollateral` function in the `Basket` contract is declared as `public`, which is inefficient due to not being used within the `Basket` contract:

```
function increaseUsedCollateral(uint256 value) public override whenNotPaused
```

Recommendation:

We recommended changing the `increaseUsedCollateral` function from `public` to `external` in order to save on the overall cost of gas:

```
function increaseUsedCollateral(uint256 value) external override whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-5: Public function should be declared external

Type	Severity	Location
Performance	Informational	Basket.sol L217

Description:

The `decreaseUsedCollateral` function in the `Basket` contract is declared as `public`, which is inefficient due to not being used within the `Basket` contract:

```
function decreaseUsedCollateral(uint256 value) public override whenNotPaused
```

Recommendation:

We recommended changing the `decreaseUsedCollateral` function from `public` to `external` in order to save on the overall cost of gas:

```
function decreaseUsedCollateral(uint256 value) external override whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-6: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	Basket.sol L234

Description:

The `getTotalCollateral` function in the `Basket` contract performs an addition between the `_collateral` and `_claimLockedCollateral` state variables without checking either value for potential overflow beforehand:

```
return _collateral + _claimLockedCollateral;
```

Recommendation:

Since `SafeMath` is already imported into the `Basket` contract for `uint256` values, we recommended using the safe `add` function in order to protect against overflow:

```
return _collateral.add(_claimLockedCollateral);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-7: Inefficient `Basket.isMarketInBasket` implementation

Type	Severity	Location
Performance	Informational	Basket.sol L249

Description:

The `isMarketInBasket` function in the `Basket` contract explicitly returns the result of performing a greater-than comparison between a `uint256` value and a zero constant, which is inefficient due to the fact that `uint256` values cannot be negative and comparison operators have a higher cost of gas than equality operators in view functions:

```
return _marketToIndex[marketAddress] > 0;
```

Recommendation:

We recommended refactoring the `isMarketInBasket` function to have a named return variable and converting the greater-than comparison into an inequality check in order to save on the overall cost of gas:

```
function isMarketInBasket(address marketAddress) public override view returns (bool result)
{
    result = _marketToIndex[marketAddress] != 0;
}
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-8: Public function should be declared external

Type	Severity	Location
Performance	Informational	Basket.sol L263

Description:

The `getUsedCollateral` function in the `Basket` contract is declared as `public`, which is inefficient due to not being used within the `Basket` contract:

```
function getUsedCollateral() public override view returns (uint256)
```

Recommendation:

We recommended changing the `getUsedCollateral` function from `public` to `external` in order to save on the overall cost of gas:

```
function getUsedCollateral() public override view returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-9: Inefficient `Basket.isPausedByMarket` implementation

Type	Severity	Location
Performance	Informational	Basket.sol L269

Description:

The `isPausedByMarket` function in the `Basket` contract performs a greater-than comparison between a `uint16` value and a zero constant before explicitly returning `true` or `false`, which is inefficient due to the fact that `uint16` values cannot be negative and comparison operators have a higher cost of gas than equality operators in view functions:

```
if(_marketPauseCounter > 0)
    return true;
return false;
```

Recommendation:

We recommended refactoring the `isPausedByMarket` function to have a named return variable anf store the result of `_marketPauseCounter` being inequal to zero in order to save on the overall cost of gas:

```
function isPausedByMarket() public view returns (bool result) {
    result = _marketPauseCounter != 0;
}
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-10: Imprecise arithmetic operations order

Type	Severity	Location
Arithmetic	Major	Basket.sol L283

Description:

The `getCollateralAvailableForWithdrawal` function in the `Basket` contract performs a multiplication on the result of a division, which will truncate in any case where the `_collateralMultiplier` state variable is greater than the result returned from the call to the `_getMinAvailableMarketCollateral` function:

```
return (_getMinAvailableMarketCollateral() / _collateralMultiplier) *
_inverseMaximumMarketShare;
```

Recommendation:

We recommended re-arranging the arithmetic to perform multiplication before division in order to prevent truncation:

```
return _getMinAvailableMarketCollateral() * _inverseMaximumMarketShare /
_collateralMultiplier;
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-11: Inefficient `Basket.getCollateralAvailableForCoverOptimized` implementation

Type	Severity	Location
Implementation	Informational	Basket.sol L287-297

Description:

The `getCollateralAvailableForCoverOptimized` function in the `Basket` contract explicitly returns one of two local variables based on which one is the least, which is inefficient compared to using a return variable:

```
if (marketAvailable < basketTotalAvailable)
    return marketAvailable;
else
    return basketTotalAvailable;
```

Recommendation:

We recommended declaring a return variable over a local variable for the available collateral and updating it if the market total collateral is lower in order to reduce the overall cost of gas:

```
function getCollateralAvailableForCoverOptimized(
    uint256 basketCollateralUsedByMarket
) public override view returns (uint256 collateral) {
    collateral = getAvailableCollateral();
    uint256 marketShareLimit = getMarketShareCollateralLimit();
    if (basketCollateralUsedByMarket < marketShareLimit)
    {
        uint256 marketAvailable = marketShareLimit - basketCollateralUsedByMarket;
        if (marketAvailable < collateral)
        collateral = marketAvailable;
    }
}
```

Alleviation:

The recommendation was not applied as it would make the code not understandable and the function is not efficiency critical.

UDI-12: Public function should be declared external

Type	Severity	Location
Performance	Informational	Basket.sol L300

Description:

The `getCollateralAvailableForCover` function in the `Basket` contract is declared as `public`, which is inefficient due to not being used within the `Basket` contract:

```
function getCollateralAvailableForCover(address marketAddress) public override view returns
(uint256)
```

Recommendation:

We recommended changing the `getCollateralAvailableForCover` function from `public` to `external` in order to save on the overall cost of gas:

```
function getCollateralAvailableForCover(address marketAddress) external override view
returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-13: Inefficient `Basket.getCurrentMarketCollateralLimitOptimized` implementation

Type	Severity	Location
Implementation	Minor	Basket.sol L323-338

Description:

The `getCurrentMarketCollateralLimitOptimized` function in the `Basket` contract performs an addition between the `_collateral` and `_claimLockedCollateral` state variables without checking either value for potential overflow beforehand:

```
return _collateral + _claimLockedCollateral;
```

It also explicitly returns one of two local variables based on which one is the least, which is inefficient compared to using a return variable:

```
if (basketTotalLimit < generalLimit)
    return basketTotalLimit;
else
    return generalLimit;
```

Recommendation:

Since `SafeMath` is already imported into the `Basket` contract for `uint256` values, we recommended using the safe `add` function in order to protect against overflow on line 332. We recommended declaring a return variable over a local variable for the general limit and updating it if the basket total limit is lower in order to reduce the overall cost of gas:

```
function getCurrentMarketCollateralLimitOptimized(
    uint256 marketUsedCollateral
) public override view returns (uint256 limit) {
    // this is the general flat limit given by total effective collateral and max market
share
    limit = getMarketShareCollateralLimit();
    // this could be a lower limit in case the collateral is almost all used up
    uint256 basketTotalLimit = marketUsedCollateral.add(getAvailableCollateral());
    // set the limit to the min of those
    if (basketTotalLimit < generalLimit)
        limit = basketTotalLimit;
}
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-14: Public function should be declared external

Type	Severity	Location
Performance	Informational	Basket.sol L341

Description:

The `getCurrentMarketCollateralLimit` function in the `Basket` contract is declared as `public`, which is inefficient due to not being used within the `Basket` contract:

```
function getCurrentMarketCollateralLimit(address marketAddress) public view returns
(uint256)
```

Recommendation:

We recommended changing the `getCurrentMarketCollateralLimit` function from `public` to `external` in order to save on the overall cost of gas:

```
function getCurrentMarketCollateralLimit(address marketAddress) external view returns
(uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-15: Inefficient loop over array state variable

Type	Severity	Location
Performance	Informational	Basket.sol L361

Description:

The `_getMinAvailableMarketCollateral` function in the `Basket` contract performs a loop which references the length of the `_markets` array state variable during every iteration, which is inefficient:

```
for (uint256 i = 0; i < _markets.length; i++) {
```

Recommendation:

We recommended storing the length of the `_markets` array state variable in a `uint256` local variable in order to save on the overall cost of gas:

```
uint256 marketCount = _markets.length;
for (uint256 i = 0; i < marketCount; i++) {
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-16: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	BasketableMarket.sol L76

Description:

The `payPremium` function in the `BasketableMarket` contract calculates the sum of the `directCollateral` local variable and the `_usedBasketCollateral` state variable using a primitive addition without checking the value for overflow:

```
directCollateral + _usedBasketCollateral
```

Recommendation:

Since `SafeMath` is already imported into the `BasketableMarket` contract for `uint256` values, we recommended using the safe `add` function in order to protect against overflow:

```
directCollateral.add(_usedBasketCollateral)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-17: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	BasketableMarket.sol L91

Description:

The `getCollateral` function in the `BasketableMarket` contract calculates the sum of the `getDirectCollateral()` and `getBasketCollateral()` using a primitive addition without checking the value for overflow:

```
return getDirectCollateral() + getBasketCollateral();
```

Recommendation:

Since `SafeMath` is already imported into the `BasketableMarket` contract for `uint256` values, we recommended using the safe `add` function in order to protect against overflow:

```
return getDirectCollateral().add(getBasketCollateral());
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-18: Redundant function definition

Type	Severity	Location
Implementation	Informational	BasketableMarket.sol L100-103

Description:

The `getEffectiveCollateral` in the `BasketableMarket` contract contains a redundant definition of the `getCollateral` function and is also marked with a comment stating that it should be removed as it is a test function:

```
/// Note for test update, TODO delete - previously getNonLockedCollateral()
function getEffectiveCollateral() public override view returns (uint256) {
    return getDirectCollateral().add(getBasketCollateral());
}
```

Recommendation:

Evaluate the implementation of the `getEffectiveCollateral` function and determine if it is still necessary in the system. If not, we recommended removing it from the `IBasket` interface to clean up any unused code.

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-19: Public function should be declared external

Type	Severity	Location
Performance	Informational	BasketableMarket.sol L123

Description:

The `getCollateralAvailableForCover` function in the `BasketableMarket` contract is declared as `public`, which is inefficient due to not being used within the `BasketableMarket` contract:

```
function getCollateralAvailableForCover() public view override returns (uint256)
```

Recommendation:

We recommended changing the `getCollateralAvailableForCover` function from `public` to `external` in order to save on the overall cost of gas:

```
function getCollateralAvailableForCover() external view override returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-20: Public function should be declared external

Type	Severity	Location
Performance	Informational	BasketableMarket.sol L127

Description:

The `potentialCover` function in the `BasketableMarket` contract is declared as `public`, which is inefficient due to not being used within the `BasketableMarket` contract:

```
function potentialCover(uint256 tokens) public view returns (uint256)
```

Recommendation:

We recommended changing the `potentialCover` function from `public` to `external` in order to save on the overall cost of gas:

```
function potentialCover(uint256 tokens) public view returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-21: Unnecessary usage of `SafeMath.add` function

Type	Severity	Location
Performance	Informational	BasketableMarket.sol L167-168

Description:

The `withdrawRefund` function in the `BasketableMarket` contract contains unnecessary usage of the `SafeMath.add` function do to the requirement on the preceding line:

```
require(marketLocked + basketLocked >= allAmount, "BMarket: not enough locked");
uint256 basketAmount = allAmount.mul(basketLocked).div(basketLocked.add(marketLocked));
```

Recommendation:

We recommended switching to a primitive addition in order to save on the overall cost of gas:

```
require(marketLocked + basketLocked >= allAmount, "BMarket: not enough locked");
uint256 basketAmount = allAmount.mul(basketLocked).div(basketLocked + marketLocked);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-22: Inefficient loop over array parameter

Type	Severity	Location
Performance	Informational	BulkDataGetter.sol L59

Description:

The `getUserData` function in the `BulkDataGetter` contract performs a loop which references the length of the supplied `marketIds` array parameter during every iteration, which is inefficient:

```
for (uint256 i = 0; i < marketIds.length; i++) {
```

Recommendation:

We recommended storing the length of the supplied `marketIds` array parameter in a `uint256` local variable in order to save on the overall cost of gas:

```
uint256 marketIdCount = marketIds.length;
for (uint256 i = 0; i < marketIdCount; i++) {
```

Alleviation:

The `BulkDataGetter.sol` file was found to be removed from the repository when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-23: Potential for uint256 multiplication overflow

Type	Severity	Location
Arithmetic	Minor	BulkDataGetter.sol L112

Description:

The `getUserData` function in the `BulkDataGetter` contract performs a primitive multiplication on the user's premium token balance and the calculated cover-per-premium without checking the value for overflow:

```
coveredAssetAmount: userPremiumTokenBalance * calculateCoverPerPremium(
    mkParams.rolloverPrice18eRatio,
    mkParams.maxPrice18eRatio,
    mkParams.cycleDurationInMonths
)
```

Recommendation:

Since `SafeMath` is already imported into the `Basket` contract for `uint256` values, we recommended using the safe `mul` function in order to protect against overflow:

```
coveredAssetAmount: userPremiumTokenBalance.mul(
    calculateCoverPerPremium(
        mkParams.rolloverPrice18eRatio,
        mkParams.maxPrice18eRatio,
        mkParams.cycleDurationInMonths
    )
)
```

Alleviation:

The `BulkDataGetter.sol` file was found to be removed from the repository when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-24: Inefficient loop over array parameter

Type	Severity	Location
Performance	Informational	BulkDataGetter.sol L116

Description:

The `getAllData` function in the `BulkDataGetter` contract performs a loop which references the length of the supplied `marketIds` array parameter during every iteration, which is inefficient:

```
for (uint256 i = 0; i < marketIds.length; i++) {
```

Recommendation:

We recommended storing the length of the supplied `marketIds` array parameter in a `uint256` local variable in order to save on the overall cost of gas:

```
uint256 marketIdCount = marketIds.length;
for (uint256 i = 0; i < marketIdCount; i++) {
```

Alleviation:

The `BulkDataGetter.sol` file was found to be removed from the repository when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-25: Inefficient `BulkDataGetter.calculateUserCollateral` implementation

Type	Severity	Location
Performance	Minor	BulkDataGetter.sol L177-186

Description:

The `calculateUserCollateral` function in the `BulkDataGetter` contract checks if the supplied `collateralTokenSupply` is equal to zero before returning zero or performing a primitive multiplication on the supplied `userCollateralTokenBalance` and `collateralMarketAssetBalance` parameters before using the `SafeMath.div` function on the supplied `collateralTokenSupply` that was previously checked for zero, which makes this function implementation inefficient:

```
if (collateralTokenSupply == 0) {
    return 0;
}
return (userCollateralTokenBalance *
collateralMarketAssetBalance).div(collateralTokenSupply);
```

Recommendation:

We recommended declaring a `collateral` return variable in the function signature, removing the `if` statement the checks if the supplied `collateralTokenSupply` parameter is equal to zero, using the `SafeMath.mul` function in place of the primitive multiplication to protect against overflow, and assigning it to the newly-defined return variable:

```
function calculateUserCollateral(
    uint256 collateralMarketAssetBalance,
    uint256 userCollateralTokenBalance,
    uint256 collateralTokenSupply
) private pure returns (uint256 collateral) {
    collateral = userCollateralTokenBalance
        .mul(collateralMarketAssetBalance)
        .div(collateralTokenSupply);
}
```

Alleviation:

The `BulkDataGetter.sol` file was found to be removed from the repository when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-26: Inefficient `BulkDataGetter.filterMarketsForUserPremium` implementation

Type	Severity	Location
Performance	Informational	BulkDataGetter.sol L260

Description:

The `filterMarketsForUserPremium` function in the `BulkDataGetter` contract performs a loop which references the length of the supplied `marketIds` array parameter during every iteration, which is inefficient:

```
for (uint256 i = 0; i < marketIds.length; i++) {
```

It also explicitly declares and returns a local variable in favor of using a named return variable, which is inefficient:

```
uint256[] memory resizedfilteredMarketIds = new uint256[](counter);
for (uint256 i = 0; i < counter; i++) {
    resizedfilteredMarketIds[i] = filteredMarketIds[i];
}
return resizedfilteredMarketIds;
```

Recommendation:

We recommended declaring a named `outMarketIds` return variable, storing the length of the supplied `marketIds` array parameter in a `uint256` local variable in order to save on the overall cost of gas in the loop, then using the newly-declared `outMarketIds` return variable in place of the local `resizedfilteredMarketIds` variable and ommitting the return statement on line 276:

```
uint256 marketIdCount = marketIds.length;
for (uint256 i = 0; i < marketIdCount; i++) {
```

```
outMarketIds = new uint256[](counter);
for (uint256 i = 0; i < counter; i++) {
    outMarketIds[i] = filteredMarketIds[i];
}
```

Alleviation:

The `BulkDataGetter.sol` file was found to be removed from the repository when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-27: Public function should be declared external

Type	Severity	Location
Performance	Informational	ERC20MintableBurnable.sol L12

Description:

The `mint` function in the `ERC20MintableBurnable` contract is declared as `public`, which is inefficient due to not being used within the `ERC20MintableBurnable` contract:

```
function mint(address account, uint256 amount) public onlyOwner
```

Recommendation:

We recommended changing the `mint` function from `public` to `external` in order to save on the overall cost of gas:

```
function mint(address account, uint256 amount) public onlyOwner
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-28: Public function should be declared external

Type	Severity	Location
Performance	Informational	ERC20MintableBurnable.sol L17

Description:

The `burn` function in the `ERC20MintableBurnable` contract is declared as `public`, which is inefficient due to not being used within the `ERC20MintableBurnable` contract:

```
function burn(address account, uint256 amount) public onlyOwner
```

Recommendation:

We recommended changing the `burn` function from `public` to `external` in order to save on the overall cost of gas:

```
function burn(address account, uint256 amount) public onlyOwner
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-29: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L137

Description:

The `withdrawPremium` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function withdrawPremium(uint256 tokenToWithdraw) public virtual override whenNotPaused
```

Recommendation:

We recommended changing the `withdrawPremium` function from `public` to `external` in order to save on the overall cost of gas:

```
function withdrawPremium(uint256 tokenToWithdraw) public virtual override whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-30: Potential for uint256 subtraction underflow

Type	Severity	Location
Arithmetic	Minor	Market.sol L193

Description:

The `withdrawCollateral` function in the `Market` contract has a requirement that performs primitive subtractions without verifying that the right-hand value is less than the left, which has the potential for underflow and to cause the requirement to incorrectly succeed:

```
require(block.timestamp - timestamp > WITHDRAW_COOLDOWN, 'Market: too soon');
```

```
uint256 cooldownPremium = PremiumPaymentLibrary.premiumPerTime(
    block.timestamp - timestamp,
    _collateral,
    _getNonLockedPremiumTokens(),
    _marketParams
);
```

Recommendation:

We recommended extending the requirement to include that that `block.timestamp` global value should be greater than or equal to the supplied `timestamp` parameter value in order to safely protect against subtraction underflow:

```
require(
    (block.timestamp >= timestamp) && (block.timestamp - timestamp > WITHDRAW_COOLDOWN),
    'Basket: too early to withdraw'
);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-31: Missing return statement leads to always returning false

Type	Severity	Location
Implementation	Minor	Market.sol L243

Description:

The `resetCycle` function in the `Market` contract is missing a return statement, which leads to the function always returning false.

```
    _updateRolloverPrice();
}
```

Recommendation:

We recommended if the `bool` return value is necessary in the `resetCycle` function and either remove the return type from the function signature or return an appropriate value. Since the `_updateRolloverPrice` function is only called here and always returns `true`, both functions can safely have their return types removed unless there is an external need for them to always return `true`:

```
function resetCycle() external whenNotPaused;
```

```
function _updateRolloverPrice() internal;
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-32: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L335

Description:

The `getCollateralToken` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function getCollateralToken() public override view returns (address)
```

Recommendation:

We recommended changing the `getCollateralToken` function from `public` to `external` in order to save on the overall cost of gas:

```
function getCollateralToken() external override view returns (address)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-33: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L340

Description:

The `getPremium` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function getPremium() public override view returns (uint256)
```

Recommendation:

We recommended changing the `getPremium` function from `public` to `external` in order to save on the overall cost of gas:

```
function getPremium() external override view returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-34: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L344

Description:

The `getPremiumToken` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function getPremiumToken() public override view returns (address)
```

Recommendation:

We recommended changing the `getPremiumToken` function from `public` to `external` in order to save on the overall cost of gas:

```
function getPremiumToken() external override view returns (address)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-35: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	Market.sol L357

Description:

The `getTotalCollateral` function in the `Market` contract performs an addition between the `_collateral` and `_claimLockedCollateral` state variables without checking either value for potential overflow beforehand:

```
return _collateral + _claimLockedCollateral;
```

Recommendation:

Since `SafeMath` is already imported into the `Market` contract for `uint256` values, we recommended using the safe `add` function in order to protect against overflow:

```
return _collateral.add(_claimLockedCollateral);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-36: Potential for uint256 subtraction underflow

Type	Severity	Location
Arithmetic	Minor	Market.sol L395

Description:

The `_getUsedCollateral` function in the `Market` contract performs a primitive subtraction without verifying that the right-hand value is less than the left, which has the potential for underflow and to cause the requirement to incorrectly succeed:

```
require(block.timestamp - timestamp > WITHDRAW_COOLDOWN, 'Basket: too early to withdraw');
```

Recommendation:

We recommended extending the requirement to include that that `block.timestamp` global value should be greater than or equal to the supplied `timestamp` parameter value in order to safely protect against subtraction underflow:

```
require(
    (block.timestamp >= timestamp) && (block.timestamp - timestamp > WITHDRAW_COOLDOWN),
    'Basket: too early to withdraw'
);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-37: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L420

Description:

The `getAvailableCollateral` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function getAvailableCollateral() public view returns (uint256)
```

Recommendation:

We recommended changing the `getAvailableCollateral` function from `public` to `external` in order to save on the overall cost of gas:

```
function getAvailableCollateral() external view returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-38: Public function should be declared external

Type	Severity	Location
Performance	Informational	Market.sol L424

Description:

The `superGetUsedCollateral` function in the `Market` contract is declared as `public`, which is inefficient due to not being used within the `Market` contract:

```
function superGetUsedCollateral() public view returns (uint256)
```

Recommendation:

We recommended changing the `superGetUsedCollateral` function from `public` to `external` in order to save on the overall cost of gas:

```
function superGetUsedCollateral() external view returns (uint256)
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-39: Inefficient `Market.premiumToPay` implementation

Type	Severity	Location
Implementation	Informational	Market.sol L434-445

Description:

The `premiumToPay` function in the `Market` contract explicitly checks if the uint256 `_premium` state variable is equal to zero before explicitly returning zero:

```
if (_premium == 0)
    return 0;
```

Otherwise, it declares a uint256 `toPay` local variable to store the calculated the premium-per-time with before returning that same local variable in favor of using a named return variable, which is inefficient:

```
uint256 toPay = PremiumPaymentLibrary.premiumPerTime(
    duration,
    collateral,
    _getNonLockedPremiumTokens(),
    _marketParams
);
return toPay;
```

Recommendation:

We recommended declaring a named uint256 `premium` return variable in the function signature and refactoring to only assign a value to it if the uint256 `_premium` state variable is not equal to zero in order to save on the overall cost of gas:

```
function premiumToPay(
    uint256 duration,
    uint256 collateral
) public view returns (uint256 premium) {
    // we need to have a deposit in premium to start premium payment
    if (_premium != 0) {
        premium = PremiumPaymentLibrary.premiumPerTime(
            duration,
            collateral,
            _getNonLockedPremiumTokens(),
            _marketParams
        );
    }
}
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-40: Inefficient `Market.accountPayPremium` implementation

Type	Severity	Location
Implementation	Minor	Market.sol L447-452

Description:

The `accountPayPremium` function in the `Market` contract contains a redundant uint256 `toReceive` parameter, as the function is internal and is only used once by the `payPremium` function, which supplies the same value for both the `toPay` and `toReceive` parameters on Market.sol L458:

```
accountPayPremium(toPay, toPay);
```

It also performs a primitive addition and subtraction on the uint256 `_premium` and `_collateral` state variables without checking for potential under/overflow:

```
_premium -= toPay;
_collateral += toReceive;
```

Recommendation:

We recommended removing the redundant `toReceive` parameter on line 447 and adjusting the call within the `payPremium` function on line 458:

```
function accountPayPremium(uint256 toPay) internal {
```

```
accountPayPremium(toPay);
```

Since `SafeMath` is already imported into the `Market` contract for `uint256` values, we recommended using the safe `sub` and `add` functions in order to protect against under/overflow:

```
_premium = _premium.sub(toPay);
_collateral = _collateral.add(toPay);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-41: Unnecessary `Market._updateRolloverPrice` return type

Type	Severity	Location
Implementation	Informational	Market.sol L484

Description:

The `_updateRolloverPrice` function in the `Market` contract always returns true, which is unnecessary:

```
  return true;
}
```

Recommendation:

Since the `_updateRolloverPrice` function is only called by the `resetCycle` function and always returns `true`, both functions can safely have their return types removed unless there is an external need for them to always return `true`:

```
function resetCycle() external whenNotPaused;
```

```
function _updateRolloverPrice() internal;
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-42: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L34

Description:

The `registerMarket` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function registerMarket(
    address payable market,
    MarketType marketType,
    string memory name,
    address claims,
    address dapp
) override public onlyOwner whenNotPaused
```

Recommendation:

We recommended changing the `registerMarket` function from `public` to `external` and its `name` parameter's type from `memory` to `calldata` in order to save on the overall cost of gas:

```
function registerMarket(
    address payable market,
    MarketType marketType,
    string calldata name,
    address claims,
    address dapp
) override external onlyOwner whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-43: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	Registry.sol L45

Description:

The `registerMarket` function in the `Registry` contract increments the uint256 `_marketCounter` state variable without checking for potential overflow:

```
_marketMap[market] = _marketCounter + 1;
_marketCounter += 1;
```

Recommendation:

We recommended importing the OpenZeppelin `SafeMath` contract and using it for uint256 values within the `Registry` contract:

```
import '@openzeppelin/contracts/math/SafeMath.sol';
```

```
contract Registry is Ownable, IRegistry, Pausable {
    using SafeMath for uint256;
```

Then use the `SafeMath.add` function in the `registerMarket` function in order to safely prevent against overflow, shifting the market map internally in the same statement in order to save on the overall cost of gas:

```
_marketMap[market] = _marketCounter = _marketCounter.add(1);
```

Alleviation:

The recommendation was not applied as it was found to be irrelevant, can't overflow.

UDI-44: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L55

Description:

The `pause` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function pause() public onlyOwner
```

Recommendation:

We recommended changing the `pause` function from `public` to `external` in order to save on the overall cost of gas:

```
function pause() external onlyOwner
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-45: Potential for uint256 subtraction underflow

Type	Severity	Location
Arithmetic	Minor	Registry.sol L65

Description:

The `getId` function in the `Registry` contract gets the entry associated with the supplied address `market` parameter from the `_marketMap` state variable without checking if the address is present in the mapping before primitively subtracting a one constant from its value, which has the potential to underflow:

```
return _marketMap[market] - 1;
```

Recommendation:

We recommended importing the OpenZeppelin `SafeMath` contract and using it for uint256 values within the `Registry` contract:

```
import '@openzeppelin/contracts/math/SafeMath.sol';
```

```
contract Registry is Ownable, IRegistry, Pausable {
 using SafeMath for uint256;
```

Then use the `SafeMath.sub` function in the `getId` function in order to safely prevent against underflow, shifting the market map internally in the same statement in order to save on the overall cost of gas:

```
return _marketMap[market].sub(1);
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-46: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L93

Description:

The `addMarket` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function addMarket(uint256 id, address payable basket) override public onlyOwner
whenNotPaused
```

Recommendation:

We recommended changing the `addMarket` function from `public` to `external` in order to save on the overall cost of gas:

```
function addMarket(uint256 id, address payable basket) override external onlyOwner
whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-47: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L102

Description:

The `removeMarket` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function removeMarket(uint256 id, address payable basket) override public onlyOwner
whenNotPaused
```

Recommendation:

We recommended changing the `removeMarket` function from `public` to `external` in order to save on the overall cost of gas:

```
function removeMarket(uint256 id, address payable basket) override public onlyOwner
whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-48: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L115

Description:

The `registerBasket` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function registerBasket(
 address payable basket,
 string memory name
) override public onlyOwner whenNotPaused
```

Recommendation:

We recommended changing the `registerBasket` function from `public` to `external` and its `name` parameter's type from `memory` to `calldata` in order to save on the overall cost of gas:

```
function registerBasket(
 address payable basket,
 string calldata name
) override external onlyOwner whenNotPaused
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-49: Potential for uint256 addition overflow

Type	Severity	Location
Arithmetic	Minor	Registry.sol L125-127

Description:

The `registerBasket` function in the `Registry` contract increments the uint256 `_basketCounter` state variable without checking for potential overflow:

```
_basketMap[basket] = _basketCounter + 1;
_basketCounter += 1;
```

Recommendation:

We recommended importing the OpenZeppelin `SafeMath` contract and using it for uint256 values within the `Registry` contract:

```
import '@openzeppelin/contracts/math/SafeMath.sol';
```

```
contract Registry is Ownable, IRegistry, Pausable {
 using SafeMath for uint256;
```

Then use the `SafeMath.add` function in the `registerBasket` function in order to safely prevent against overflow, shifting the basket map internally in the same statement in order to save on the overall cost of gas:

```
_basketMap[market] = _basketCounter = _basketCounter.add(1);
```

Alleviation:

The recommendation was not applied as it was found to be irrelevant, can't overflow.

UDI-50: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L137

Description:

The `basketPause` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function basketPause(address payable basket) public override onlyOwner
```

Recommendation:

We recommended changing the `basketPause` function from `public` to `external` in order to save on the overall cost of gas:

```
function basketPause(address payable basket) external override onlyOwner
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.

UDI-51: Public function should be declared external

Type	Severity	Location
Performance	Informational	Registry.sol L146

Description:

The `marketPause` function in the `Registry` contract is declared as `public`, which is inefficient due to not being used within the `Registry` contract:

```
function marketPause(address payable market) public override onlyOwner
```

Recommendation:

We recommended changing the `marketPause` function from `public` to `external` in order to save on the overall cost of gas:

```
function marketPause(address payable market) external override onlyOwner
```

Alleviation:

The recommendation was found to be applied when referencing commit e0a8a6cf9cf6fbc90c36a2d2bd1b2ad6aed39398.