// Private Smart Contract Security Assessment 04.11.2024 - 05.10.2024

Strategies Contracts

GoldLink

HALBORN

Strategies Contracts - GoldLink

Prepared by: HALBORN

Last Updated 05/19/2024

Date of Engagement by: April 11th, 2024 - May 10th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
12	1	2	2	1	6

TABLE OF CONTENTS

7.12 Inaccurate comments in the code

1. Introduction

GoldLink engaged Halborn to conduct a security assessment on their smart contracts beginning on April 11th, 2024 and ending on May 10th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team.

2. Assessment Summary

The team at Halborn assigned a full-time security engineer to verify the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were successfully addressed by the GoldLink team. The main ones were the following:

- Update the logic of the 'getOrderValueUSD' function to return 0 only when the order type is different than a market increase.
- Set the value of the position size when creating decrease orders.
- Assign correctly the addresses for the long and short tokens.
- Implement adequately the conditions regarding the size of the swap rebalances.
- Ensure that all initial values are set in an initializer function.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9

SEVERITY	SCORE VALUE RANGE
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: goldlink-contracts

(b) Assessed Commit ID: 41af1fe

(c) Items in scope:

- contracts/strategies/gmxFrf/configuration/DeploymentConfigurationManager.sol
- contracts/strategies/gmxFrf/configuration/MarketConfigurationManager.sol
- contracts/strategies/gmxFrf/impl/GmxStrategyStorage.sol
- contracts/strategies/gmxFrf/libraries/AccountGetters.sol
- contracts/strategies/gmxFrf/libraries/ClaimLogic.sol
- contracts/strategies/gmxFrf/libraries/DeltaConvergenceMath.sol
- contracts/strategies/gmxFrf/libraries/GmxMarketGetters.sol
- contracts/strategies/gmxFrf/libraries/GmxStorageGetters.sol
- contracts/strategies/gmxFrf/libraries/Limits.sol
- contracts/strategies/gmxFrf/libraries/LiquidationLogic.sol
- contracts/strategies/gmxFrf/libraries/OrderHelpers.sol
- contracts/strategies/gmxFrf/libraries/OrderLogic.sol
- contracts/strategies/gmxFrf/libraries/OrderValidation.sol
- contracts/strategies/gmxFrf/libraries/Pricing.sol
- contracts/strategies/gmxFrf/libraries/SwapCallbackLogic.sol
- contracts/strategies/gmxFrf/libraries/WithdrawalLogic.sol
- contracts/strategies/gmxFrf/GmxFrfStrategyAccount.sol
- contracts/strategies/gmxFrf/GmxFrfStrategyDeployer.sol
- contracts/strategies/gmxFrf/GmxFrfStrategyErrors.sol
- contracts/strategies/gmxFrf/GmxFrfStrategyManager.sol

Out-of-Scope: contracts/strategies/gmxFrf/SwapCallbackRelayer.sol, Interactions with SwapCallbackRelayer contract., Third party dependencies., Economic attacks.

REMEDIATION COMMIT ID:

- 9c7387b9c7387b
- 0006b890006b89
- 0836ca30836ca3
- c23d6ecc23d6ec
- 907b0c4907b0c4

- ce86685ce86685
- ca41e98ca41e98
- f6f9c06f6f9c06
- 439cc18439cc18
- 5cba6795cba679

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	2	1	6

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - MISCALCULATION OF THE ACCOUNTS TOTAL VALUE IN TERMS OF USD	Critical	SOLVED - 05/16/2024
HAL-02 - POSITION SIZE IS NOT SET WHEN CREATING DECREASE ORDERS	High	SOLVED - 05/16/2024
HAL-03 - ADDRESSES OF THE SHORT AND LONG TOKENS ARE INCORRECTLY ASSIGNED	High	SOLVED - 05/17/2024
HAL-04 - EXECUTING SWAP REBALANCES COULD REVERT	Medium	SOLVED - 05/17/2024
HAL-05 - VARIABLE IS NOT CORRECTLY INITIALIZED IN AN UPGRADEABLE CONTRACT	Medium	SOLVED - 05/13/2024
HAL-06 - IMPLEMENTATION CONTRACT UNINITIALIZED	Low	SOLVED - 05/16/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-07 - CACHING ARRAY LENGTH IN LOOPS CAN SAVE GAS	Informational	SOLVED - 05/16/2024
HAL-08 - TOO STRICT CONDITION WHEN REBALANCING POSITIONS	Informational	SOLVED - 05/16/2024
HAL-09 - LACK OF ZERO ADDRESS CHECK	Informational	SOLVED - 05/16/2024
HAL-10 - SOME FUNCTIONS DO NOT VERIFY IF MARKETS ARE APPROVED	Informational	SOLVED - 05/17/2024
HAL-11 - FUNCTION WITH MISLEADING NAME	Informational	SOLVED - 05/16/2024
HAL-12 - INACCURATE COMMENTS IN THE CODE	Informational	SOLVED - 05/16/2024

7. FINDINGS & TECH DETAILS

7.1 (HAL-01) MISCALCULATION OF THE ACCOUNTS TOTAL VALUE IN TERMS OF USD

// CRITICAL

Description

The `getOrderValueUSD` function in the **AccountGetters** library calculates the value of an order in USD, considering that the value of any non-increase order is **0**. To verify if an order is a non-increase one, the following conditional expression is evaluated:

```
226 | if (
227 |     orderType != IGmxV2OrderTypes.OrderType.MarketIncrease ||
228 |     orderType != IGmxV2OrderTypes.OrderType.LimitIncrease ||
229 |     orderType != IGmxV2OrderTypes.OrderType.MarketSwap ||
230 |     orderType != IGmxV2OrderTypes.OrderType.LimitSwap
231 |    ) {
232 |    return 0;
233 | }
```

Currently, the protocol only supports the following order types:

- `MarketDecrease`: For this order type, the function will return **0**, as expected.
- `MarketIncrease`: For this order type, the function will also return **0**, instead of the **correct value**. It happens because the order type is always different from `LimitIncrease`, `MarketSwap` or `LimitSwap`, so the conditional expression will evaluate to true and the function will return **0**.

As a result, the accounts total value will be miscalculated, having a value much lower than expected, which triggers the following consequences:
1. The value of health score for accounts will be below the expected one, which could allow that many of them are unfairly liquidated.
2. Limitations to borrow only a lesser amount of funds than expected from the strategy reserve.
3. Limitations to withdraw only a lesser amount of collaterals than expected from the strategy bank.
4. Limitations to withdraw only a lesser amount of profit than expected from the account.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:H/D:N/Y:M (10.0)

Recommendation

It is recommended to update the logic of the mentioned function to return **0** only when the order type is different from a market increase.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/9c7387b80185bdd3cb262e842a112c7875b3b7f3

7.2 (HAL-02) POSITION SIZE IS NOT SET WHEN CREATING DECREASE ORDERS

// HIGH

Description

The creation of decrease orders mainly involves the execution of the `OrderLogic.createDecreaseOrder` function (i.e.: the **caller**) and the `DeltaConvergenceMath.getDecreaseOrderValues` function (i.e.: the **callee**). This latter function does not set the value of the position size (`result.positionSizeNextUsd`), which means that its value is 0. As a result, the following consequences are triggered:

Consequence #1:

Bypass of the order size validation during the execution of the `OrderLogic.createDecreaseOrder` function when creating decrease orders:

```
342 if (result.positionSizeNextUsd != 0 && sizeDeltaUsd != 0) {
343   // Validate the order size. Only do this if remainingPositionSize !=
344 0, since it may be impossible to
345   // reduce the position size in the event the remaining size is less
346 than the min order size.
347   OrderValidation.validateOrderSize(
348     marketConfig.orderPricingParameters.minOrderSizeUsd,
349     marketConfig.orderPricingParameters.maxOrderSizeUsd,
350     order.numbers.sizeDeltaUsd
      );
    }
```

Consequence #2:

Bypass of the position size validation during the execution of the `OrderValidation.validatePositionSize` function when creating decrease orders:

Code Location

The `getDecreaseOrderValues` function does not set the value of `result.positionSizeNextUsd`:

```
981 function getDecreaseOrderValues(
982   IGmxFrfStrategyManager manager,
983   uint256 sizeDeltaUsd,
984   DeltaCalculationParameters memory values
985 ) internal view returns (DecreasePositionResult memory result) {
986   PositionTokenBreakdown memory breakdown = getAccountMarketDelta(
987     manager,
988     values.account,
989     values.marketAddress,
```

```
    sizeDeltaUsd,
    false
  );

  // The total cost amount is equal to the sum of the fees associated
with the decrease, in terms of the collateral token.
  // This accounts for negative funding fees, borrowing fees,
  uint256 collateralLostInDecrease = breakdown
    .positionInfo
    .fees
    .totalCostAmount;

  {
    uint256 profitInCollateralToken = SignedMath.abs(
      breakdown.positionInfo.pnlAfterPriceImpactUsd
    ) / values.longTokenPrice;

    if (breakdown.positionInfo.pnlAfterPriceImpactUsd > 0) {
      collateralLostInDecrease -= Math.min(
        collateralLostInDecrease,
        profitInCollateralToken
      ); // Offset the loss in collateral with position profits.
    } else {
      collateralLostInDecrease += profitInCollateralToken; // adding
because this variable is meant to represent a net loss in collateral.
    }
  }

  uint256 sizeDeltaActual = Math.min(
    sizeDeltaUsd,
    breakdown.positionInfo.position.numbers.sizeInUsd
  );

  uint256 shortTokensAfterDecrease;

  {
    uint256 proportionalDecrease = sizeDeltaActual.fractionToPercent(
      breakdown.positionInfo.position.numbers.sizeInUsd
    );

    shortTokensAfterDecrease =
    breakdown.tokensShort -
    breakdown
      .positionInfo
```

```
1034        .position
1035        .numbers
1036        .sizeInTokens
1037        .percentToFraction(proportionalDecrease);
1038    }
1039
1040    uint256 longTokensAfterDecrease = breakdown.tokensLong -
1041      collateralLostInDecrease;
1042
1043    // This is the difference in long vs short tokens currently.
1044    uint256 imbalance = Math.max(
1045      shortTokensAfterDecrease,
1046      longTokensAfterDecrease
1047    ) - Math.min(shortTokensAfterDecrease, longTokensAfterDecrease);
1048
1049    if (shortTokensAfterDecrease < longTokensAfterDecrease) {
1050      // We need to remove long tokens equivalent to the imbalance to
1051  make the position delta neutral.
1052      // However, it is possible that there are a significant number of
1053  long tokens in the contract that are impacting the imbalance.
1054      // If this is the case, then if we were to simply remove the
1055  imbalance, it can result in a position with very high leverage.
1056  Therefore, we will simply remove
1057      // the minimum of `collateralAmount - collateralLostInDecrease` the
1058  difference in the longCollateral and shortTokens. The rest of the delta
1059  imbalance can be left to rebalancers.
1060      uint256 remainingCollateral = breakdown
1061      .positionInfo
1062      .position
1063      .numbers
1064      .collateralAmount - collateralLostInDecrease;
1065
1066      if (remainingCollateral > shortTokensAfterDecrease) {
1067        result.collateralToRemove = Math.min(
1068          remainingCollateral - shortTokensAfterDecrease,
1069          imbalance
1070        );
1071      }
1072    }
1073
1074    if (result.collateralToRemove != 0) {
1075      (uint256 expectedSwapOutput, , ) = manager
1076        .gmxV2Reader()
1077        .getSwapAmountOut(
```

```
        manager.gmxV2DataStore(),
        values.market,
        _makeMarketPrices(
          values.shortTokenPrice,
          values.longTokenPrice
        ),
        values.market.longToken,
        result.collateralToRemove,
        values.uiFeeReceiver
      );

    result.estimatedOutputUsd =
    expectedSwapOutput *
    values.shortTokenPrice;
  }

  if (breakdown.positionInfo.pnlAfterPriceImpactUsd > 0) {
    result.estimatedOutputUsd += SignedMath.abs(
      breakdown.positionInfo.pnlAfterPriceImpactUsd
    );
  }

  result.executionPrice = breakdown
    .positionInfo
    .executionPriceResult
    .executionPrice;
}
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:L/Y:L (8.8)

Recommendation

It is recommended to correctly set the value of the position size when creating decrease orders.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/0006b8906f7befeee8e470a516a276e79adec1ed

7.3 (HAL-03) ADDRESSES OF THE SHORT AND LONG TOKENS ARE INCORRECTLY ASSIGNED

// HIGH

Description

The `isLiquidationFinished` function in the **AccountGetters** library incorrectly assigns the addresses for the long and short tokens. It happens because when calling the `GmxMarketGetters.getMarketTokens` function, the returned values are assigned to the addresses in the opposite order, i.e.: short token has the address of the long one and viceversa.
As a consequence, the `isLiquidationFinished` function won't return accurate results, which means that some lenders wouldn't receive their payments even if the liquidation process had indeed finished.

Code Location

The `AccountGetters.isLiquidationFinished` function incorrectly assigns the addresses for the long and short tokens when calling `GmxMarketGetters.getMarketTokens`:

```
129 // Get all available markets to check funding fees for.
130 address[] memory markets = manager.getAvailableMarkets();
131
132 for (uint256 i = 0; i < markets.length; ++i) {
133   (address longToken, address shortToken) = GmxMarketGetters
134     .getMarketTokens(dataStore, markets[i]);
```

The `GmxMarketGetters.getMarketTokens` function returns the values of the short and long tokens respectively, which is the opposite from what is expected in the function above:

```
90 function getMarketTokens(
91   IGmxV2DataStore dataStore,
92   address market
93 ) internal view returns (address shortToken, address longToken) {
94   return (
95     getShortToken(dataStore, market),
96     getLongToken(dataStore, market)
97   );
98 }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:M/D:N/Y:N/R:N/S:U (8.8)

Recommendation

It is recommended to assign the addresses correctly for the long and short tokens.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/0836ca32e880a6344fd3d5acdc5695a0d699afad

7.4 (HAL-04) EXECUTING SWAP REBALANCES COULD REVERT

// MEDIUM

Description

In order to prevent gaming and token rebalances by continuously sending tokens to the account, the `swapRebalancePosition` function in the **LiquidationLogic** library requires that the size of the swap rebalance must either:

1. Leave the account with zero delta.
2. Leave the account with zero tokens that can be atomically swapped.
3. Leave the account with a balance that is greater than or equal to the minimum swap rebalance size.

However, the first condition is not correctly implemented because it compares the value of `breakdown.tokensLong - breakdown.tokensShort` against the `remaining` variable, instead of comparing it against the `rebalanceAmount` variable. As a consequence, when executing swap rebalances, the transaction could unnecessarily revert, making the operation temporarily unavailable.

Code Location

The `swapRebalancePosition` function does not correctly implement the first condition in the `require` function:

```
151 require(
152   remaining == breakdown.tokensLong - breakdown.tokensShort ||
153   remaining == 0 ||
154   remaining >= unwindConfig.minSwapRebalanceSize,
155   GmxFrfStrategyErrors
156
157 .LIQUIDATION_MANAGEMENT_REBALANCE_AMOUNT_LEAVE_TOO_LITTLE_REMAINING_ASSET
    );
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:M/D:N/Y:N (6.3)

Recommendation

It is recommended to correctly implement the conditions regarding the size of the swap rebalances.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/0836ca32e880a6344fd3d5acdc5695a0d699afad

7.5 (HAL-05) VARIABLE IS NOT CORRECTLY INITIALIZED IN AN UPGRADEABLE CONTRACT

// MEDIUM

Description

The `isInMulticall_` variable has a initial value of **1** when declared in the **GmxStrategyStorage** contract. This is equivalent to setting this value in the constructor, and as such, will not work for upgradeable contracts, which means that any upgradeable instances will not have this field set.
In this particular case, the `multicall` function in the **GmxFrfStrategyAccount** will always revert with the error message `Nested multicalls are not allowed` when invoking it because the `isInMulticall_` variable will remain with **0** as a value, instead of **1** as expected. The described issue directly affects the logic for paying execution fee / gas.

Code Location

The `isInMulticall_` variable has a initial value of **1** when declared in the **GmxStrategyStorage** contract:

```
48  /// @notice Should be set when a multicall is active to prevent nested
49  multicalls.
50  /// The value `1` implies the contract is not current executing a
51  multicall.
    /// The value `2` implies that the contract is currently executing a
    multicall.
    uint256 internal isInMulticall_ = 1;
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:L/Y:N (5.6)

Recommendation

It is recommended to make sure that all initial values are set in an initializer function.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/c23d6ec731fd998bddd25bc9b16ef840fd915bb2

7.6 (HAL-06) IMPLEMENTATION CONTRACT UNINITIALIZED

// LOW

Description

The **GmxFrfStrategyAccount** contract is using the `Initializable` module from OpenZeppelin. In order to prevent leaving the contract uninitialized, OpenZeppelin's documentation recommends adding the `_disableInitializers` function in the constructor to automatically lock the contract when it is deployed.

Code Location

The `constructor` in the **GmxFrfStrategyAccount** contract does not include the `_disableInitializers` function:

```
73  // ============ Constructor ============
74
75  /**
76   * @notice Constructor for upgradeable contract, distinct from
77  initializer.
78   *
79   *  The constructor is used to set immutable variables, and for top-
80  level upgradeable
81   *  contracts, it is also used to disable the initializer of the logic
82  contract.
83   *
84   *  Note that since this contract is used with beacon proxies, the
    immutable variables will
     *  be constant across all proxies pointing to the same beacon.
     */
    constructor(IGmxFrfStrategyManager manager) GmxStrategyStorage(manager)
    {}
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

It is recommended to call the `_disableInitializers` function in the contract's constructor.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/907b0c409870ada744b27dde81373b6e93aeb878

7.7 (HAL-07) CACHING ARRAY LENGTH IN LOOPS CAN SAVE GAS

// INFORMATIONAL

Description

Reading the length of the array at each iteration of the loop requires 6 gas (3 for `mload` and 3 to place `memory_offset`) onto the stack. Caching the length of the array on the stack saves about 3 gas per iteration. The affected functions are the following:

- `AccountGetters.isLiquidationFinished`
- `AccountGetters.getAccountOrdersValueUSD`
- `AccountGetters.getAccountPositionsValueUSD`
- `AccountGetters.getSettledFundingFeesValueUSD`
- `AccountGetters._getAccountTokenValueUSD`
- `OrderValidation.validateNoPendingOrdersInMarket`
- `GmxFrfStrategyAccount.multicall`
- `GmxFrfStrategyManager.setMarket`

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:L/Y:N (1.7)

Recommendation

It is recommended to consider caching the length of the arrays.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/ce86685eaa38dda510ef8acb1a49b0b21c06b18c

7.8 (HAL-08) TOO STRICT CONDITION WHEN REBALANCING POSITIONS

// INFORMATIONAL

Description

The `rebalancePosition` function in the **LiquidationLogic** library reverts if the position's delta proportion is **less or equal than** the threshold defined in the market's unwind configuration. However, in order to maintain the consistency along the codebase (e.g.: with `swapRebalancePosition` function), the function should revert only when the position's delta proportion is **less than** the mentioned threshold.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (0.8)

Recommendation

It is recommended to update the logic of the `rebalancePosition` function to revert only when the position's delta proportion is less than the configured threshold.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/ca41e98381b8b98a84d1c0b9f93a447f7fa57a73

7.9 (HAL-09) LACK OF ZERO ADDRESS CHECK

// INFORMATIONAL

Description

Some functions in the codebase do not include a **zero address check** for their parameters. If one of those parameters is mistakenly set to zero, it could affect the correct operation of the protocol. The affected functions are the following:

- `MarketConfigurationManager._setUiFeeReceiver`
- `GmxStrategyStorage.constructor`
- `GmxFrfStrategyAccount.initialize`
- `GmxFrfStrategyDeployer.constructor`
- `GmxFrfStrategyDeployer.deployAccount`

BVSS

AO:A/AC:L/AX:H/R:P/S:U/C:N/A:N/I:M/D:N/Y:N (0.8)

Recommendation

It is recommended to add a zero address check in the functions mentioned above.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/f6f9c0624a06706ad7b40c9929085ebf4d8bee6c

7.10 (HAL-10) SOME FUNCTIONS DO NOT VERIFY IF MARKETS ARE APPROVED

// INFORMATIONAL

Description

Some functions in the **GmxFrfStrategyAccount** contract do not verify if the markets they are interacting with have been approved by the **GmxFrfStrategyManager** contract, e.g.: by using the `onlyApprovedMarket` modifier. The affected functions are the following:

- `GmxFrfStrategyAccount.executeClaimFundingFees`
- `GmxFrfStrategyAccount.executeClaimCollateral`
- `GmxFrfStrategyAccount.executeLiquidatePosition`
- `GmxFrfStrategyAccount.executeReleveragePosition`
- `GmxFrfStrategyAccount.executeSwapRebalance`
- `GmxFrfStrategyAccount.executeRebalancePosition`

This issue has been classified as **Informational** because in the mentioned functions, there cannot be position / funding fees for non-approved markets. However, it is included in the report as part of a security-in-depth approach to harden the functions if the codebase is later refactored.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:N/D:N/Y:L (0.8)

Recommendation

It is recommended to verify in the mentioned functions if the markets are approved before further processing.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/0836ca32e880a6344fd3d5acdc5695a0d699afad

7.11 (HAL-11) FUNCTION WITH MISLEADING NAME

// INFORMATIONAL

Description

The `getMinimumAcceptablePriceForDecrease` function in the **OrderHelpers** library calculates the maximum acceptable price used as threshold when validating the prices for decrease orders. However, the name of the function suggests that the value calculated represents a minimum threshold instead of a maximum one, which could mislead users and even developers if the codebase is later refactored.

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to update the name of the mentioned function to reflect its real behavior.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/439cc18b849238de37c016b018a15ccb17b0c442

7.12 (HAL-12) INACCURATE COMMENTS IN THE CODE

// INFORMATIONAL

Description

Along the codebase, there are some functions that include inaccurate comments, which could mislead users or developers (if code is later refactored) when trying to understand the expected behavior of the functions. The affected resources are the following:

`DeltaConvergenceMath._getLeverage:`

The comment indicates the following regarding the **net collateral value**:

```
// Net Collateral Value: Calculated using the following formula:
(collateral amount tokens - funding fees tokens - borrowing fees tokens) *
token price usd + pnl usd.
```

However the code calculates its value in a different way (using `totalCostAmount`):

```
uint256 collateralInTokens = breakdown
         .positionInfo
         .position
         .numbers
         .collateralAmount - breakdown.positionInfo.fees.totalCostAmount;
```

`GmxFrfStrategyAccount.executeLiquidateAssets:`

The comment indicates the following regarding the `executeLiquidateAssets` function:

```
* @notice Liquidates the specified `asset` in the amount of `amount` to the
`receiever` address. Can only be called when the account has no active
loan. The `callback` address must be
```

However, the function uses the `hasActiveLoan` modifier, which indicates the opposite behavior:

```
function executeLiquidateAssets(
  address asset,
  uint256 amount,
  address callback,
  address receiever
) external strategyNonReentrant whenLiquidating hasActiveLoan {
```

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to update the comments in the code to reflect the actual behavior of the mentioned functions.

Remediation Plan

SOLVED: The **GoldLink team** solved the issue in the specified commit id.

Remediation Hash

https://github.com/GoldLink-Protocol/goldlink-contracts/commit/5cba679246e48f11c0f5c09619cd17ff8a65a10a

---

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.