CERTORA

GMX V2 Smart Contract Security Analysis and Formal Verification Preliminary Report

Date of report release:
13/11/2023

Table of Contents

CERTORA

Summary

This document describes the specification and verification of **GMX V2 protocol** using the Certora Prover and manual code review findings. The work was undertaken from **13 June 2023** to **27 August 2023**. The latest commit reviewed and ran through the Certora Prover was 4a816ec.

The Certora Prover proved the implementation of the contracts above is correct with respect to the formal rules written by the Certora team. The team also performed a manual audit of the contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the code, as listed in the table below.

All the rules will be publicly available in the project's repository.

The following contracts list is included in the **scope**:

`contracts/adl/AdlUtils.sol`
`contracts/bank/Bank.sol`
`contracts/bank/StrictBank.sol`
`contracts/callback/CallbackUtils.sol`
`contracts/callback/IDepositCallbackReceiver.sol`
`contracts/callback/IOrderCallbackReceiver.sol`
`contracts/callback/IWithdrawalCallbackReceiver.sol`
`contracts/chain/ArbSys.sol`
`contracts/chain/Chain.sol`
`contracts/config/Config.sol`
`contracts/config/Timelock.sol`
`contracts/data/DataStore.sol`
`contracts/data/Keys.sol`
`contracts/deposit/Deposit.sol`
`contracts/deposit/DepositEventUtils.sol`
`contracts/deposit/DepositStoreUtils.sol`
`contracts/deposit/DepositUtils.sol`
`contracts/deposit/DepositVault.sol`
`contracts/deposit/ExecuteDepositUtils.sol`
`contracts/error/Errors.sol`
`contracts/error/ErrorUtils.sol`
`contracts/event/EventEmitter.sol`
`contracts/event/EventUtils.sol`
`contracts/exchange/AdlHandler.sol`
`contracts/exchange/BaseOrderHandler.sol`
`contracts/exchange/DepositHandler.sol`
`contracts/exchange/ExchangeUtils.sol`
`contracts/exchange/LiquidationHandler.sol`
`contracts/exchange/OrderHandler.sol`
`contracts/exchange/WithdrawalHandler.sol`

contracts/feature/FeatureUtils.sol
contracts/fee/FeeHandler.sol
contracts/fee/FeeUtils.sol
contracts/gas/GasUtils.sol
contracts/liquidation/LiquidationUtils.sol
contracts/market/Market.sol
contracts/market/MarketEventUtils.sol
contracts/market/MarketFactory.sol
contracts/market/MarketPoolValueInfo.sol
contracts/market/MarketStoreUtils.sol
contracts/market/MarketToken.sol
contracts/market/MarketUtils.sol
contracts/nonce/NonceUtils.sol
contracts/oracle/IPriceFeed.sol
contracts/oracle/OracleModule.sol
contracts/oracle/OracleStore.sol
contracts/oracle/OracleUtils.sol
contracts/order/BaseOrderUtils.sol
contracts/order/DecreaseOrderUtils.sol
contracts/order/IncreaseOrderUtils.sol
contracts/order/Order.sol
contracts/order/OrderEventUtils.sol
contracts/order/OrderUtils.sol
contracts/order/OrderVault.sol
contracts/order/SwapOrderUtils.sol
contracts/position/DecreasePositionCollateralUtils.sol
contracts/position/DecreasePositionSwapUtils.sol
contracts/position/DecreasePositionUtils.sol
contracts/position/IncreasePositionUtils.sol
contracts/position/Position.sol
contracts/position/PositionEventUtils.sol
contracts/position/PositionStoreUtils.sol
contracts/position/PositionUtils.sol
contracts/price/Price.sol
contracts/pricing/PositionPricingUtils.sol
contracts/pricing/PricingUtils.sol
contracts/pricing/SwapPricingUtils.sol
contracts/reader/Reader.sol
contracts/reader/ReaderPricingUtils.sol
contracts/reader/ReaderUtils.sol
contracts/referral/IReferralStorage.sol
contracts/referral/ReferralEventUtils.sol
contracts/referral/ReferralTier.sol
contracts/referral/ReferralUtils.sol

contracts/role/Role.sol
contracts/role/RoleModule.sol
contracts/role/RoleStore.sol
contracts/router/ExchangeRouter.sol
contracts/router/Router.sol
contracts/swap/SwapHandler.sol
contracts/swap/SwapUtils.sol
contracts/token/IWNT.sol
contracts/token/TokenUtils.sol
contracts/utils/AccountUtils.sol
contracts/utils/Array.sol
contracts/utils/BasicMulticall.sol
contracts/utils/Bits.sol
contracts/utils/Calc.sol
contracts/utils/Cast.sol
contracts/utils/EnumerableValues.sol
contracts/utils/GlobalReentrancyGuard.sol
contracts/utils/PayableMulticall.sol
contracts/utils/Precision.sol
contracts/utils/Printer.sol
contracts/utils/Uint256Mask.sol
contracts/withdrawal/Withdrawal.sol
contracts/withdrawal/WithdrawalEventUtils.sol
contracts/withdrawal/WithdrawalStoreUtils.sol
contracts/withdrawal/WithdrawalUtils.sol
contracts/withdrawal/WithdrawalVault.sol

CERTORA

Summary of Findings

The table below summarizes the issues discovered during the audit, categorized by severity.

Severity	Total discovered	Total fixed	Total acknowledged
High	3	2	1
Medium	2	0	2
Information	1	0	1
Total (High, Medium, Low)	**6**	**2**	**4**

Disclaimer

The Certora Prover takes a contract, and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification, and the Certora Prover does not check any cases not covered by the specification.

We hope this information is helpful, but we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

Main Issues Discovered

H-01: Config keepers cannot configure global non-specific parameters.

Severity: High
File(s): `Config.sol`
Bug description: Config keepers cannot configure global non-specific parameters.
The list of unconfigurable keys includes:

```
HOLDING_ADDRESS,
MIN_HANDLE_EXECUTION_ERROR_GAS,
MAX_SWAP_PATH_LENGTH,
MAX_CALLBACK_GAS_LIMIT,
MIN_POSITION_SIZE_USD,
MIN_ORACLE_BLOCK_CONFIRMATIONS,
MAX_ORACLE_PRICE_AGE,
MAX_ORACLE_REF_PRICE_DEVIATION_FACTOR,
POSITION_FEE_RECEIVER_FACTOR,
SWAP_FEE_RECEIVER_FACTOR,
BORROWING_FEE_RECEIVER_FACTOR,
ESTIMATED_GAS_FEE_BASE_AMOUNT,
ESTIMATED_GAS_FEE_MULTIPLIER_FACTOR,
EXECUTION_GAS_FEE_BASE_AMOUNT,
EXECUTION_GAS_FEE_MULTIPLIER_FACTOR,
SINGLE_SWAP_GAS_LIMIT,
INCREASE_ORDER_GAS_LIMIT,
DECREASE_ORDER_GAS_LIMIT,
SWAP_ORDER_GAS_LIMIT,
NATIVE_TOKEN_TRANSFER_GAS_LIMIT,
REQUEST_EXPIRATION_BLOCK_AGE,
MAX_UI_FEE_FACTOR,
SKIP_BORROWING_FEE_FOR_SMALLER_SIDE.
```

These baseKeys should not be configurable at all since all the references in the code are to keys calculated using `MAX_PNL_FACTOR` as the `baseKey`:

```
MAX_PNL_FACTOR_FOR_TRADERS,
MAX_PNL_FACTOR_FOR_ADL,
MAX_PNL_FACTOR_FOR_DEPOSITS,
MAX_PNL_FACTOR_FOR_WITHDRAWALS.
```

Implications: Config keepers are allowed to configure all the parameters with `allowedBaseKeys[PARAMETER_BASE_KEY] = true` listed in the `config.sol` file. Some of these parameters are non-specific, meaning the corresponding key in the datastore is the `baseKey` itself, while others are (such as parameters that are market-specific or address-specific), and their corresponding keys in the datastore are the result of applying the keccak256 function over the concatenation of the `baseKey` and the `bytes` representing the specific data.

In order to configure a parameter, the config keeper can call one of the set functions in `config.sol` with the baseKey, the specific data, and the new parameter value. The full key which is later changed in the datastore is calculated as follows:
`bytes32 fullKey = keccak256(bytes.concat(baseKey, data));`
For non-specific parameters, the full key is incorrect even if the data is empty, since the key that will actually be changed is `keccak256(baseKey)` instead of the baseKey itself, causing some random irrelevant value in the datastore to change instead of the intended value.
GMX's response: We acknowledged the bug and provided a fix - commit hash link. [Certora approves that the bug is fixed in the commit above]

H-02: Lack of mechanism at the configKeeper.

Severity: High
File(s): `Keys.sol`
Bug description: Lack of mechanism for the configKeeper to set the values under `CLAIMABLE_COLLATERAL_TIME_DIVISOR, WNT, MIN_ORACLE_SIGNER`.
Implications: For the following keys, there doesn't seem to be any mechanism to set values in dataStore:
`CLAIMABLE_COLLATERAL_TIME_DIVISOR`
`WNT`
`MIN_ORACLE_SIGNERS`
These keys aren't whitelisted and they can't be set by the configKeeper.
Additionally, there aren't any setters in the controller contracts using those keys.
It was said that no EOAs would be a controller, therefore only the `ROLE_ADMIN` Timelock contract would be able to set those. But perhaps too late. The following places are impacted until the values are set:
`CLAIMABLE_COLLATERAL_TIME_DIVISOR`: revert due to a division by 0 in MarketUtils.incrementClaimableCollateralAmount L487
`WNT`: `Bank.sol` can't receive native ETH and the majority of the protocol's functionalities are impacted (Deposits, Orders, Withdraws...)
`MIN_ORACLE_SIGNERS`: `Oracle._getSigners` will always revert due to a < 0 statement (noticeably called via `Oracle.validatePrices` and `Oracle.setPrices`)
GMX's response: For `CLAIMABLE_COLLATERAL_TIME_DIVISOR, WNT` we are unlikely to change these keys so there is no config for them.
for `MIN_ORACLE_SIGNERS`, we are working on integrating the new Chainlink price feeds after v2 is live so there is also no plan to change this value.

H-03: First depositor attack.

Severity: Medium
File(s): `MarketUtils.sol`
Bug description: For some tokens, the first depositor into a market might be able to artificially inflate the share price (marketToken price), potentially causing future depositors to get credit for 0 marketTokens due to rounding.
Implications: When a deposit is being executed, the function `usdToMarketTokenAmount()` is called to determine the amount of shares to be minted as a result of the user's deposit. The first deposit into a pool (when total supply is `0`) sets an intended fixed price per share of `10^-18` USD. However, by owning exactly `1` marketTokens after the first deposit (This is only possible for tokens whose basic unit is worth less than `10^-18` USD), a malicious depositor could repeatedly deposit tokens into the pool, doubling the pool's value with each deposit without minting any new tokens. This could be done by depositing an amount just short of the current `poolValue`, causing the calculated minted shares to round down to `0`. As the initial share price is `10^-18` USD, approximately `60` deposits are required to inflate the share price to 1 USD, and any additional inflation would require more (10 additional deposits to make it ~`1,000` USD, `20` additional deposits to make it ~`1,000,000` USD, etc.).
Note: As one deposit may call `usdToMarketTokenAmount()` more than once (for example, if the depositor deposits to both the short and long side of the pool), it is possible to reduce the amount of deposits needed.
GMX's response: We acknowledged the bug.

M-01: Keepers can cancel users' deposits/withdrawals/orders at will.

Severity: Medium
File(s): `DepositHandler.sol, WithdrawalHandler.sol, OrderHandler.sol`
Bug description: Keepers can cancel users' deposits/withdrawals/orders at will while charging the users for the execution and canceling costs.
Implications: A keeper can call `executeDeposit/executeWithdrawal/executeOrder` with not enough gas for `_executeDeposit/_executeWithdrawal/_executeOrder` to finish (any value between slightly over `minHandleErrorGas` to the amount of gas `_executeDeposit/_executeWithdrawal/_executeOrder` consumes). This would cause the action's execution to revert and `_handleDepositError/_handleWithdrawalError/_handleOrderError` to be called, canceling the action and charging the user for the costs.
GMX's response: Acknowledged as known issue under "Order keepers may cause requests to be cancelled instead of executed by executing the request with insufficient gas".

M-02: Charging execution fees when not needed.

Severity: Medium

File(s): `CallbackUtils.sol`
Bug description: User-defined callbacks that consume more gas than a certain threshold (`~58,000 * 63 =~ 3,600,000, or ~22,000 * 63 =~ 1,400,000` if `ExecutionFee` is Smaller than the actual execution cost) can be purposefully reverted by the keeper. At the same time, the user will still be charged for the execution costs.
Implications: This is possible due to `EIP-150`, which states that the caller will always maintain at least `1/64` of the gas left (so the gas available for the called function is always capped at `63/64` of the gas left). If the callback function consumes more than `63` times the gas needed to complete the transaction, the keeper can perform the transaction with not enough gas for the callback (which will revert) but just enough to receive the payment and complete the transaction.
GMX's response: Acknowledged as known issue.

I-01: Unused constants.

Severity: Information
File(s): `Config.sol, MarketStoreUtils.sol`
Bug description: Unused constants
Implications: The following constants are unused:

```
Config.MAX_FEE_FACTOR
MarketStoreUtils.MARKET_KEY
```

Formal Verification Properties

Since the protocol consists of different contracts, we will present the relative properties of the main contracts. The rule's name will be shown as: (`ruleName`)

OrderHandler

1. ⏰if the price value is the same, no sequence of actions should result in a net profit or another way to phrase it would be that markets should always be solvent if price does not change.
2. ✅ If the market has a long token that is the same as the index token and the reserveFactor is less than 1, then the market should always be solvent regardless of the price of the index token
3. ✅ if the market has a long token that is not the same as the index token and the max pnl factor for traders is less than 1, then the market should always be solvent regardless of the price of the index token. we defined solvency as all open positions are backed by the contract's balances (both orderVault and depositVault). This is proved for all the methods of `OrderHandler` aside from `simulateExecuteOrder` because `simulateExecuteOrder` always reverts.
4. ⏰ If price does not change, no sequence of actions should lead to a decrease in the market pool value, all funding fees and protocol fees can be claimed and all market tokens can be redeemed.

SwapHandler

1. ✅ For a MarketSwap or LimitSwap, the swap output amount is `amount of tokens in * tokenInPrice / tokenOutPrice` for some value of the fees and price impact.

RoleStore

1. ✅ After calling `grantRole(account, role)`, the account should have the role assigned. The number of roles for the given account should increase by `1` if it did not have this role assigned, otherwise, it should stay unchanged. (`countIncreaseByOneWhenGrantRole`)

2. ✅ After calling `renounceRole(account, role)`, the account should not have the role assigned. The number of roles for the given account should decrease by `1` if it had this role assigned, otherwise, it should stay unchanged (`countDecreaseByOneWhenRenounceRole`)

3. ✅ Any function in the contract should not affect two different accounts, i.e., if the number of roles changes for one account, it does not for any other. (`memberCountNonInterference`)

4. ✅ A function call can only change the value of `hasRole(account, role)` for one role and one account. (`nonInterferenceOfRolesAndAccounts`)

5. ✅ If a role is `added to/removed from roleCache` or `roleMembers` (or if the `roles.length()` changes), it must have been due to a call from someone with an admin role. (`onlyAdminCanGrantOrRevokeRoles`)

6. ✅ The data in `roleCache` and `roleMembers` is consistent, i.e. `hasRole(account,role) == roleMembers(role).contains(account)`. (`dataConsistency`)

StrictBank / DepositVault

1. ✅ `syncTokenBalance` should not change the `tokenBalances[token']` where `token' != token` (`syncTokenBalanceChangesNoOtherBalance`)

2. ✅ `_recordTransferIn` does not change then `tokenBalances[token']` where `token' != token`. `_recordTransferIn` also sets `tokenBalances[token]` to `token.balanceOf(this)` (`recordTransferInChangesNoOtherBalance`)

3. ✅ `_afterTransferOut` should not change the `tokenBalances[token']` where `token' != token` (`afterTransferOutChangesNoOtherBalance`)

TokenUtils

1. ✅ After a successful call to `transfer`:
   contracts balance should decrease by the amount transferred.
   The receiver or The HoldingAddress balance should increase by the amount transferred. (`integrityOfTransfer`)

2. ✅After a successful call to `depositAndSendWrappedNativeToken`:
   The receiver or The HoldingAddress balance should increase by the amount transferred.
   (`integrityOfDepositAndSendWrappedNativeToken`)

3. ✅ After a successful call to `nonRevertingTransferWithGasLimit`:
   The contract's balance should decrease by the amount transferred.
   The receiver balance should increase by the amount transferred.
   (`integrityOfNonRevertingTransfer`)

4. ✅ Transfer between two users doesn't affect the other user's balances.
   (`nonRevertingTransferDontChangeOther`)

OracleStore

1. ✅ For addSigner, if the caller does not have the controller role the contract reverts and the result of calling getSigner will not change before/after the function.
(`non_controller_add_signer`)

2. ✅ For removeSigner, if the caller does not have the controller role
the contract reverts and the result of calling getSigner will not change before/after the function.
(`non_controller_remove_signer`)

3. ✅ Calling removeSigner with an address that has not been added
to the list of signers previously will have no affect on: getSigner(s), getSignerCount.
(`remove_signer_not_in_list`)

4. ✅ Calling getSigner with an invalid index "fails gracefully".
(`get_invalid_index`)

5. ✅ Calling addSigner with the controller role will: increase getSignerCount, and add the signer to the result of getSinger for some index.
(`add_signer_valid_liveness`)

6. ✅ Calling removeSigner as a controller and on an address that has been added to the list of signers previously will: decrease getSigners, ensure the address will not appear in the result of getSigner(s) for any index.
(`remove_signer_valid_liveness`)

7. ✅ Calling getSignerCount() twice in a row with no other interleaving calls
results in the same value.
(`double_get_signer_count`)

8. ✅ Removing a signer will not cause any other signer to be removed.
(`remove_signer_deletes_no_others`)