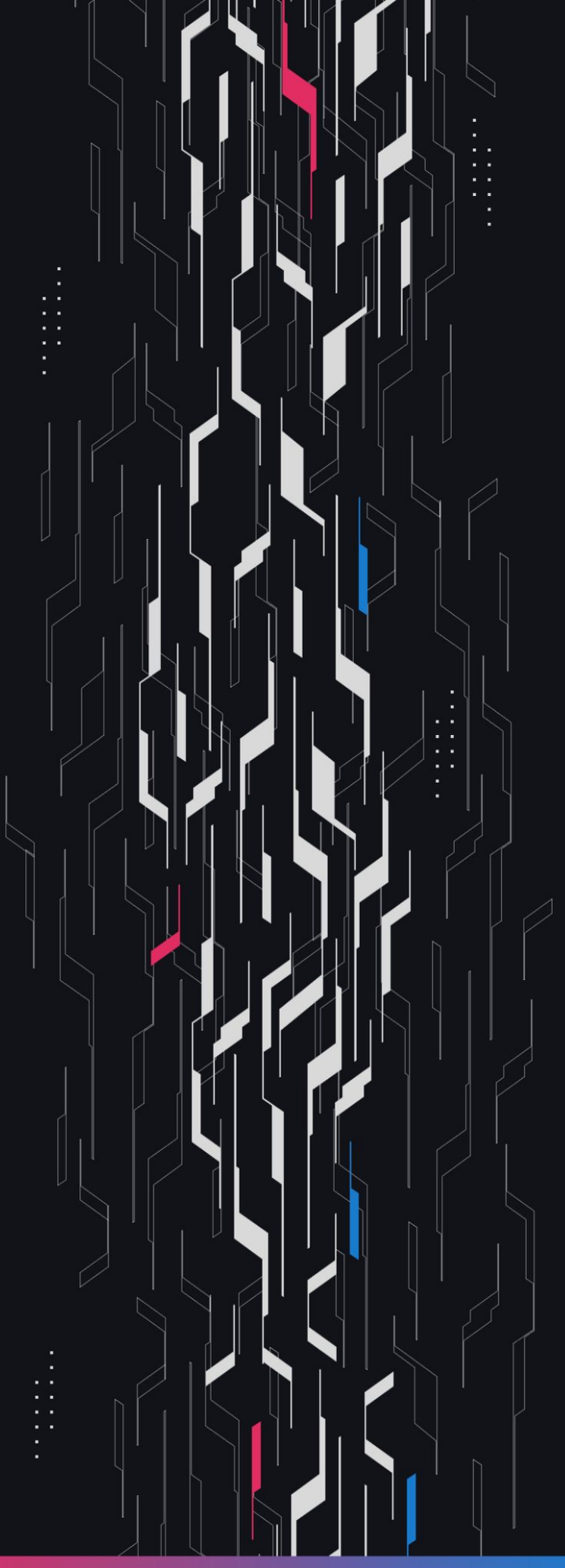
GA GUARDIAN

Gamma

GMX V2 Vault

Security Assessment

September 23rd, 2024



Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Kiki, 0xCiphky, Mark Jonathas, Vladimir Zotov

Client Firm Gamma Strategies

Final Report Date September 23, 2024

Audit Summary

Gamma Strategies engaged Guardian to review the security of its GMX V2 perpetual vault. From the 1st of August to the 15th of August, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 18 High/Critical issues were uncovered and promptly remediated by the Gamma team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the main functionality described for the vault.

Security Recommendation Given the number of High and Critical issues detected, Guardian supports an independent security review of the protocol at a finalized frozen commit. Furthermore, the Gamma team should increase testing with a variety of callback scenarios, such as ADL and liquidations.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/gamma-gmx-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Gamma GMX V2 Vault
Language	Solidity
Codebase	https://github.com/GammaStrategies/PerpetualVault
Commit(s)	Initial commit: 14cc80741a2754df310ace87fcd7801865391064 Final commit: b6cd6dcce7d3dc8e61853120a824cdfda3ed5121

Audit Summary

Delivery Date	September 23, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	4	0	0	0	0	4
High	14	0	0	1	5	8
Medium	12	0	0	2	1	9
Low	10	0	0	1	0	9

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Gamma's GMX V2 vault, fuzz-testing with Echidna was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
GAMMA-01	The distribution of fees should be proportional to each depositor's share of the total vault	✅	✅	✅	10M+
GAMMA-02	The sum of all individual depositor shares must always equal the totalShares variable in the contract.	✅	✅	✅	10M+
GAMMA-03	The value of a depositor's shares should never decrease due to the actions of other depositors.	✅	✅	✅	10M+
GAMMA-04	After all actions completed, nextAction should be empty.	✅	✅	✅	10M+
GAMMA-05	After all actions completed, swapProgressData should be empty.	✅	✅	✅	10M+
GAMMA-06	PositionKey should be zero when there is no position.	✅	✅	✅	10M+
GAMMA-07	No depositor should be able to withdraw their funds before the lockTime period has passed since their deposit.	✅	✅	✅	10M+
GAMMA-08	If GmxUtils (account) has a GMX position open, positionIsClosed == false	✅	✅	✅	10M+
GAMMA-09	After order execution FLOW should be cleared	✅	❌	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
GAMMA-10	After order execution GMX lock should be cleared	✅	✅	✅	10M+
GAMMA-11	If withdraw function is called on a open GMX position, callback should always hit the “settle” case in the afterOrderExecution and afterOrderCancellation function.	✅	✅	✅	10M+
GAMMA-12	If user deposits they will get a non-zero amount of shares	✅	✅	✅	10M+
GAMMA-13	The keeper should never be able to do a DEX swap or a GMX swap (any swap) when there is a nonzero curPositionKey	✅	✅	✅	10M+
GAMMA-14	If user withdraws they will get non zero amount in return	✅	✅	✅	10M+
GAMMA-15	Withdraw should succeed after liquidation	✅	❌	✅	10M+
GAMMA-16	Withdraw should succeed after ADL	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	DOS attack in withdraw Function	DoS	Critical	Resolved
C-02	Missing access control on settle function	Access Control	Critical	Resolved
C-03	Callback Check Bypassed Via Self Liquidation	Access Control	Critical	Resolved
C-04	Users Avoid Losses From GMX	Logical Error	Critical	Resolved
H-01	cancelDeposit not implemented in KeeperProxy	Logical Error	High	Resolved
H-02	cancelDeposit missing totalDepositAmt deduction	Logical Error	High	Resolved
H-03	Long funding is never claimed	Logical Error	High	Resolved
H-04	Unhandled ADL case can lead to stuck funds	Logical Error	High	Partially Resolved
H-05	Decrease orders can output 2 tokens	Logical Error	High	Resolved
H-06	Missing liquidation callback case	Logical Error	High	Partially Resolved
H-07	Users Can Be Charged Extra Fees	Logical Error	High	Resolved
H-08	Disproportional share allocation	Logical Error	High	Partially Resolved
H-09	Asset Transfer During Liquidation Can Fail	Logical Error	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-10	Price Impact cap on DecreasePosition	Logical Error	High	Resolved
H-11	Retrying Actions Causes DoS	DoS	High	Partially Resolved
H-12	Vault Liquidated By Malicious Depositor	Gaming	High	Resolved
H-13	cancelDeposit Misses swapProgressData	Logical Error	High	Acknowledged
H-14	Withdrawers Lose Funds Due To Collateral Check	Logical Error	High	Partially Resolved
M-01	Hardcoded GMX address can lead to dos	Configuration	Medium	Acknowledged
M-02	Missing feature execution validation	DoS	Medium	Resolved
M-03	Deprecated latestAnswer Used in _check Function	Logical Error	Medium	Resolved
M-04	Chainlink price lacks price/sequencer validation	Validation	Medium	Resolved
M-05	Excess executionFee not refunded	Logical Error	Medium	Resolved
M-06	DOS when orders are stuck in gmx	DoS	Medium	Resolved
M-07	Gas Estimation Uses Old Key	Logical Error	Medium	Resolved
M-08	Liquidations Don't Account For longTokens	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-09	Execution Fee Required For Swaps	Logical Error	Medium	Partially Resolved
M-10	Keeper actions can be stalled by deposits	DoS	Medium	Resolved
M-11	setPerpVault Lacks Validation	Access Control	Medium	Acknowledged
M-12	Oracle Price Counts Ignored In Gas Estimation	Logical Error	Medium	Resolved
L-01	Protocol fee does not count secondary token	Logical Error	Low	Resolved
L-02	Users Always Charged Negative Impact Fees	Documentation	Low	Resolved
L-03	Redundant Boolean Expressions	Optimization	Low	Resolved
L-04	Run can be stalled via acceptable price	Logical Error	Low	Resolved
L-05	cancelDeposit Doesn't update Mapping	Logical Error	Low	Resolved
L-06	cancelDeposit Function Should Use safeTransfer	Logical Error	Low	Resolved
L-07	Max Deposit Cap Can Be Exceeded	Warning	Low	Resolved
L-08	Position Accounting Mismatch	Logical Error	Low	Acknowledged
L-09	Settlement can unexpectedly hit withdrawal case	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-10	1x Long State Siphoning collateralToken	Gaming	Low	Resolved

C-01 | DoS Attack In Withdraw Function

Category	Severity	Location	Status
DoS	● Critical	PerpetualVault.sol: 222	Resolved

Description

The `withdraw` function in the `PerpetualVault` contract allows users to specify a recipient when withdrawing collateral.

The internal `_withdraw` function is then called to withdraw funds from GMX if the position is open. Once withdrawn, the collateral tokens are transferred to the user-specified recipient address to complete the withdrawal flow. No other actions are permitted until this process is completed.

This allows a malicious actor to perform the following attack:
• Call the `withdraw` function with the recipient set to a blacklisted USDC address.
• The `afterOrderExecution` function in the `GmxUtils` contract will revert when trying to send the collateral (USDC) to the blacklisted address.
• The flow variable will still be 3 (WITHDRAW) and `_gmxLock` set to true, preventing any further actions.

This effectively causes a DOS attack by locking the contract in a state where no further actions can be taken.

Recommendation

Do not transfer tokens directly to the recipient, instead make the token claimable for them in a separate transaction and store their claimable balance in a mapping.

Resolution

Gamma Team: The issue was resolved in commit a557f1d.

C-02 | Missing Access Control On Settle Function

Category	Severity	Location	Status
Access Control	● Critical	GMXUtils.sol: 455	Resolved

Description PoC

A user is able to arbitrarily call `settle` and overwrite any order that is in the queue. By doing this, the user will not receive funds from their withdrawal.

An attacker could easily time this to repeatedly cause users to have their withdrawn funds locked in the `PerpetualVault` contract.

Recommendation

Implement access control to the function so that it can only be called from the `PerpetualVault` contract.

Resolution

Gamma Team: The issue was resolved in commit b4022d5.

C-03 | Callback Check Bypassed Via Self Liquidation

Category	Severity	Location	Status
Access Control	● Critical	GMXUtils.sol: 84	Resolved

Description

GMX allows users to set their callback contract in case of liquidation through the `setSavedCallbackContract` function.

`function setSavedCallbackContract(address market,address callbackContract) external payable nonReentrant`

This feature can be leveraged to bypass the `validCallback` function because the order is a liquidation which `GMXUtils` will not prevent.

An attacker would do so by executing the following attack:
• Attacker will set the callback contract to `GMXUtils`
• Open small highly leveraged position
• Attacker gets liquidated and the `GMXUtils` contracts `afterOrderExecution` will be called
• `queue` will be deleted, removing the pending order.
• `PerpetualVault` contracts `afterOrderExecution` will then be called and all conditionals will be skipped leaving the protocol in a flow state
• Protocol is bricked as it is stuck in a flow state, and no pending deposits/withdrawals.

The result of this attack would be DoS of the protocol as well as a lost order from the deleted queue.

Recommendation

Validate that the liquidation is for the vault `positionKey` rather than any arbitrary position in the `validCallback` modifier.

Resolution

Gamma Team: The issue was resolved in commit 57b3fae.

C-04 | Users Avoid Losses From GMX

Category	Severity	Location	Status
Logical Error	● Critical	PerpetualVault.sol	Resolved

Description

In the perpetual vault when withdrawing value from the vault there is no accounting for the negative `PnL` that a GMX perpetual position may have.

Instead the user's decrease order contains the `collateralDeltaAmount` and `sizeDeltaInUsd` amounts that are proportional to the shares the user holds.

The `sizeDeltaInUsd` value will indeed realize the proportion of the negative `PnL` that the user owns, however this negative `PnL` amount will be deducted from the position's collateral that remains, and not from the `collateralDeltaAmount` which will be rewarded to the withdrawer.

As a result withdrawers avoid losses and disburse them onto other vault holders.

Recommendation

Account for the `PnL` of a position that will be realized when computing the `collateralDeltaAmount` for a withdrawal.

Resolution

Gamma Team: The issue was resolved in commit 94347b8.

H-01 | cancelDeposit Not Implemented In KeeperProxy

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 343	Resolved

Description

The `cancelDeposit` function has an `onlyKeeper` modifier that restricts its execution to the `KeeperProxy` contract.

However, the `KeeperProxy` does not implement the necessary functionality to call this function, making it currently impossible to invoke.

The `cancelDeposit` function may be needed to cancel a reverting deposit to GMX, making it essential for the system's proper operation.

Recommendation

Implement a function in the `KeeperProxy` contract to call the `cancelDeposit` function.

Resolution

Gamma Team: The issue was resolved in commit 24f5790.

H-02 | cancelDeposit Missing totalDepositAmt Deduction

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 343	Resolved

Description

The `cancelDeposit` function allows the protocol to cancel an ongoing deposit, resetting the state and deleting the deposit request.

However, it currently forgets to deduct the deposit request's amount from the `totalDepositAmount` state variable. This variable is used to ensure that the deposited amount within the protocol remains under the deposit cap.

Consequently, the missing deduction in the `cancelDeposit` function will lead to an overestimation of the total deposits. Every time `cancelDeposit` is called, the remaining possible deposit amount (`maxDepositAmount - totalDepositAmount`) will be incorrectly reduced by the cancelled amount.

This will incorrectly limit deposits, affecting the vault strategy and, in the worst case, could cause a DOS for the entire vault.

Recommendation

Ensure that the `cancelDeposit` function deducts the cancelled deposits amount from the `totalDepositAmount` state variable.

Resolution

Gamma Team: The issue was resolved in commit 5c40cff.

H-03 | Long Funding Is Never Claimed

Category	Severity	Location	Status
Logical Error	High	GMXUtils.sol: 251	Resolved

Description

When the `afterOrderExecution` function is called, Gamma has the opportunity to collect funding fees for both the long and short tokens.

It is important for Gamma to claim these fees, otherwise they will remain in GXM. However, only the short funding fee is currently being claimed (`tokens[0] = order.addresses.initialCollateralToken`).

This issue means that some funding fees will go unclaimed, resulting in the expected yield not being distributed to users as intended.

Recommendation

Consider claiming both long and short funding fees.

Resolution

Gamma Team: The issue was resolved in commit f2008b6.

H-04 | Unhandled ADL Case Can Lead To Stuck Funds

Category	Severity	Location	Status
Logical Error	High	GMXUtils.sol: 232	Partially Resolved

Description

The GmxUtils contract doesn't account for Automatic Deleveraging (ADL) from GMX, which can partially or fully close profitable positions if pending profits exceed the market's threshold.

During an ADL event, the receiver variable is set to the account address, which in this case is the GmxUtils contract. Since the GmxUtils contract doesn't set a savedCallbackContract (which is used as the callback contract for ADLs), the callback address will default to the zero address.

As a result, the ADL callback will not be triggered, meaning the ADL event won't be properly handled. This causes any funds sent to the GmxUtils contract to become stuck with no way to retrieve them.

Additionally, these funds will no longer be included in share calculations, leading to discrepancies in share distribution and vault accounting, as they aren't in the PerpetualVault contract or the GMX position.

If the position is closed, further calls to GMX, such as withdrawal requests, could lead to a denial of service (DOS) because the PerpetualVault is incorrectly marked as open while the GMX position is closed, causing those calls to repeatedly fail.

Recommendation

The GmxUtils contract should set the savedCallbackContract so the afterOrderExecution function in the GmxUtils contract is called during ADLs or liquidations. Additionally the validCallback modifier should allow the AdlHandler to call the GmxUtils contract for the current positionKey. The afterOrderExecution function should also account for ADL calls from GMX.

This update should include a mechanism to transfer any funds sent to the GmxUtils contract due to ADL to the PerpetualVault contract. Additionally, the afterOrderExecution function in the PerpetualVault contract should check if the position is still open and handle the position decrease accordingly.

If the position is closed, it should adjust the flow to reflect that closure. It is also important to consider the various flows (deposit, signal change, withdraw, compound) that the vault can be in during the PerpetualVault callback.

Resolution

Gamma Team: Will handle liquidation risk via offchain logic.

H-05 | Decrease Orders Can Output 2 Tokens

Category	Severity	Location	Status
Logical Error	High	GMXUtils.sol: 232	Resolved

Description

The `afterOrderExecution` function in the `GmxUtils` contract assumes that only one output token will be returned. However, GMX decrease position orders can output two tokens instead of a single token if the decrease position swap fails.

As a result, this scenario is not properly accounted for in the `_handleReturn` function, which is responsible for calculating the user's withdrawal amounts and updating global states. This can result in users receiving less than they should and can corrupt the accounting.

Recommendation

Update the `afterOrderExecution` function in the `GmxUtils` contract to read and pass down the `secondaryOutputToken` and `secondaryOutputAmount` in the same way as the `outputToken` and `outputAmount`.

Then, modify the `_handleReturn` function in the `PerpetualVault` contract to account for both tokens.

Resolution

Gamma Team: The issue was resolved in commit f2008b6.

H-06 | Missing Liquidation Callback Case

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 361	Partially Resolved

Description PoC

During a liquidation, the receiver variable is set to the account address, which in this case is the GmxUtils contract. Since the GmxUtils contract doesn't set a savedCallbackContract (which is used as the callback contract for liquidations), the callback address will default to the zero address.

As a result, the callback will not be triggered, meaning the afterOrderExecution function in the GmxUtils contract, which is meant to send the liquidated funds to the perpetualVault contract won't be triggered. This causes the funds sent to the GmxUtils contract to become stuck with no way to retrieve them.

Additionally, interacting with the protocol after said liquidation can lead to DoS's and incorrect fund and share allocation. This is in part because liquidations will close the position which results in withdraw attempts failing when the settle order executes.

After the failed execution Gamma will retry the settle over and over failing each time. Deposits also present a special case where the deposit may underflow _totalAmount(marketPrices) could be less than amount: totalAmountBefore = _totalAmount(marketPrices) - amount;

Recommendation

The GmxUtils contract should set the savedCallbackContract so the afterOrderExecution function in the GmxUtils contract is called during ADLs or liquidations.

Additionally, consider handling the liquidation case where after funds are sent back to the perpetualVault there are specific actions taken to get the protocol to a state where DOS's will not occur. This includes updating the state so that Gamma has the position as closed. It also includes preparing the state for any upcoming or incoming deposits/withdrawals.

Set positionIsClosed to true and curPositionKey to bytes32(0) in afterOrderExecution(). Also, configure the callback via setSavedCallback(), and expect the call for liquidations to originate from GMX's LiquidationHandler contract in validCallback().

Resolution

Gamma Team: Liquidation issue; will handle liquidation risk via offchain logic.

H-07 | Users Can Be Charged Extra Fees

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 377	Resolved

Description

When creating an increase order for a deposit the `amountIn` is assigned to the `collateralToken` balance of the vault, however the user may have only deposited a portion of these tokens.

Indeed when a user receives shares for their deposited amount, the minted shares are computed only based on the `depositInfo[counter].amount` and does not include any additional balance that may have been used to create the order.

However the user pays fees for the balance which was not a direct part of their deposit. A malicious actor may leverage this to cause the `afterOrderExecution` execution to revert by forcing the `feeAmount` to be larger than the `depositInfo[counter].amount`.

Consider the following example:
• The vault holds a 5x Long position in GMX
• User A deposits the minimum deposit amount of 10 USDC
• The `positionFeeFactor` fee in GMX is 30 basis points
• User A then donates 990 USDC to the vault before the next action is run
• The keeper runs the next action and initiates a GMX increase order for $1,000 * 5x leverage = $5,000 `sizeDeltaUsd`
• The order is executed and the order fee is 0.003 * 5,000 = $15
• The $15 fee is larger than the user's actual deposit amount which is stored in the `depositInfo[counter].amount`
• An underflow occurs when attempting to compute the value to mint based on: `depositInfo[counter].amount - feeAmount` = 10 - 15

As a result the active increase order is not recorded as completing in Gamma's vault and the deposit flow is never completed, locking all user deposits. A malicious actor only needs to spend $1,000 in this example and could profit off of this activity by short-selling the GAMMA token and spreading negative press about the loss of funds for users.

Recommendation

Consider only charging the fee that the user's deposit amount would directly be responsible for when determining the user's shares amount. Otherwise consider only creating an increase order for the amount of USDC that the user deposited rather than the entire vault balance.

Resolution

Gamma Team: The issue was resolved in commit 97159d6.

H-08 | Disproportional Share Allocation

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 669	Partially Resolved

Description

In the `afterOrderExecution` function the amount of shares minted to a depositor is dependent on the `sizeDeltaUsd` of the order. However this method ignores price impact which the vault position experiences due to the increase order.

This allows depositors to receive more shares of the vault than they ought to when their order is negatively impacted.

For example:
• Vault is 2x long on GMX with $100,000 `sizeInUsd`
• PnL is currently 0 and the vault position is worth $50,000
• User A deposits with $5,000 and the sizeInUsd for their order is $10,000
• User A's order is negatively impacted such that it experiences a $1,000 loss relative to the current market price
• Ignoring fees, User A is credited with their 5,000 `collateralToken` deposit for share calculations
• The vault worth excluding the trader's collateral is $49,000 as the negative price impact is attributed
• The trader receives $5,000/$49,000 ~= 20.41% of the share supply
• Instead the trader should have received $4,000/$50,000 ~= 18% of the share supply by rightfully charging the impact to the depositor

Recommendation

Compute the amount of price impact that the depositor experienced by comparing the `sizeInUsd` of the order to the `sizeInTokens` increase of the position with respect to the current market prices in GMX.Then deduct this amount from the user's credited deposit value when calling the `_mint` function.

Resolution

Gamma Team: The issue was resolved in commit cb4ae42.

Guardian Team: The share calculation has been adjusted to account for the difference between the balance before and after. This modification will more accurately distribute shares, although there may still be a disproportionate distribution of shares in certain cases due to price fluctuations between creation and execution.

H-09 | Asset Transfer During Liquidation Can Fail

Category	Severity	Location	Status
Logical Error	High	GMXUtils.sol: 232	Resolved

Description

During a liquidation, the `afterOrderExecution` function in the `GmxUtils` contract is responsible for sending the `queue.tokenIn` balance from the `GmxUtils` contract to the `PerpetualVault` contract.

The problem is because the `queue.tokenIn` variable is set during the `createOrder` function's specifically for a `MarketIncrease` order and is deleted in the `afterOrderExecution` or `afterOrderCancellation` functions.

If a liquidation occurs when there is no `MarketIncrease` order in progress, the `queue.tokenIn` variable will be set to the zero address, causing the transfer of funds during liquidation to fail.

As a result, these funds in the `GmxUtils` contract become stuck with no way to retrieve them, leading to losses for users, as they are not included in the `PerpetualVault` or GMX position and will not be accounted for in share calculations.

Recommendation

The `outputToken` should be used from the eventData instead, similar to how it's done in the `afterOrderExecution` function. This ensures that the correct token address is used.

Resolution

Gamma Team: Liquidation issue; will handle liquidation risk via offchain logic

H-10 | Price Impact Cap On DecreasePosition

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 361	Resolved

Description

GMX can cap negative price impact on decrease orders, leaving the excess amount in a claimable collateral pool that must be manually claimed using the `ExchangeRouter.claimCollateral` function.

This scenario is not currently handled, meaning that if it occurs during a user withdrawal, the user could receive less than they should have.

Additionally, the `GmxUtils` contract does not currently implement the `claimCollateral` function, which means those funds cannot be retrieved, leading to potential losses for users.

Recommendation

Update the `GmxUtils` contract to implement the `claimCollateral` function, ensuring that any excess funds due to capped negative price impact can be claimed and properly accounted for.

Additionally, modify the withdrawal process to handle this scenario, ensuring users receive the full amount they are entitled to, even when negative price impact is capped.

Resolution

Gamma Team: The issue was resolved in commit 018ab42.

H-11 | Retrying Actions Causes DoS

Category	Severity	Location	Status
DoS	High	PerpVault: 448	Partially Resolved

Description

When an attempted order is canceled by GMX, the protocol chooses to handle this scenario by attempting to retry the previously attempted action.

However, there are numerous reasons why the transaction may fail again. For instance, if the reserve ratio or open interest is out of balance for a market, if a market or action is disabled, or if GMX triggers Auto Deleveraging.

A malicious user could even force this to occur by depositing, withdrawing, or swapping on GMX to attain one of these states. When this occurs Gamma's flow will be stuck in its current state, and no other actions will be executable.

Recommendation

Reset the transaction flow, and require the user or keeper to reattempt the action again. In the case of deposits, make sure that amount is refunded.

Also consider adding additional admin privileged functions to terminate a flow, similar to `cancelDeposit()`.

Resolution

Gamma Team: The issue was resolved in commit 15f3967.

Guardian Team: There may still be DOS when actions are forced to repeatedly retry. Consider implementing the entire recommendation.

H-12 | Vault Liquidated By Malicious Depositor

Category	Severity	Location	Status
Gaming	High	Global	Resolved

Description

A malicious actor may observe that Gamma currently has a high leverage and intentionally push this leverage up to force the `PerpetualVault` position to be liquidated. A user may do this by creating large deposits and then initiating withdrawals for those deposits as soon as possible to levy the `MarketDecrease` order position fee on the vault position.

Users do not pay for the `MarketDecrease` order position fee and thus this fee is deducted from the remaining position thus changing the leverage of the remaining position. Once this has been repeated several times the vault position will have a very high leverage which makes it liquidatable almost instantaneously.

The malicious actor can profit off of this liquidation by setting a limit order which can only be triggered once the liquidation has occurred and the `priceImpact` allows for immediate profit.

Recommendation

Adjust the `initialCollateralDelta` amount for the Position fees that will be experienced for the `MarketDecrease` order.

This way the `withdrawer` takes on the burden of the fee and the vault position leverage cannot be manipulated.

Resolution

Gamma Team: The issue was resolved in commit f03e60e.

H-13 | cancelDeposit Misses swapProgressData

Category	Severity	Location	Status
Logical Error	High	PerpetualVault.sol: 343	Acknowledged

Description

In the `cancelDeposit` function there is no accounting for the `swapProgressData`, this causes several issues when a deposit is cancelled after a composite swap through Paraswap and GMX where the GMX swap failed.

Firstly the transfer of collateral tokens is likely to fail as part of the deposited collateral tokens will have been swapped to the index token and are recorded in the `swapProgressData.swapped` value.

`safeTransfer` is not used for this transfer and therefore depending on the `collateralToken` used, users could lose their funds as a result.

Secondly the `swapProgressData.swapped` value is not cleared when the deposit occurs, therefore the next deposit will be credited with receiving these swapped tokens.

Therefore if the vault did have enough collateral tokens to pay out the user, this swapped value would then be double counted and awarded to the next depositor at the expense of previous vault shareholders.

Recommendation

Account for the `swapProgressData.swapped` value in the `cancelDeposit` function such that if the `swapProgressData.swapped` value is nonzero that amount of index tokens is transferred to the user and deducted from the amount of collateral tokens transferred.

Then clear the `swapProgressData` at the end of the function. Additionally, use `safeTransfer` in the `cancelDeposit` function.

Resolution

Gamma Team: `cancelDeposit` function is allowed to call when `runNextAction()` fails. Not used to call after `runNextAction`.

H-14 | Withdrawers Lose Funds Due To Collateral Check

Category	Severity	Location	Status
Logical Error	High	Global	Partially Resolved

Description

In GMX if a decrease order is deemed to leave the remaining position with insufficient collateral for its open interest, then the `initialCollateralDelta` is reassigned to 0.

This will result in withdrawals which are deemed to leave behind insufficient collateral via the `willPositionCollateralBeSufficient` check in receiving no collateral tokens out from GMX.

Therefore, `withdrawers` which are affected by this case lose all collateral and will only receive a portion of any profit which was gained from the position.

Recommendation

Validate that the `willPositionCollateralBeSufficient` check in GMX will not be triggered upon creating withdrawals from GMX, and if it is triggered upon execution of a decrease order for a withdrawal, adjust the user's shares such that they can claim their portion of the collateral that did not come out of the decrease order.

Resolution

Gamma Team: The issue was resolved in commit 75723b7.

Guardian Team: Only assign the `sizeDeltaInUsd` to the entire position size if the `willCollateralBeSufficient` check is false. And additionally be sure to carefully account for the user only receiving the funds out of the order which they should when this happens.

M-01 | Hardcoded GMX Address Can Lead To DoS

Category	Severity	Location	Status
Configuration	Medium	GMXUtils.sol: 66-71	Acknowledged

Description

The `GmxUtils` contract uses the `gExchangeRouter` and reader contracts, saving their addresses as constants during deployment.

However, it is recommended by GMX to allow these addresses to be changeable (not immutable) and to provide setter functions.

These contracts are currently used in essential functions by the protocol, meaning if they were to be changed, the protocol will not function correctly until the addresses are updated through an upgrade.

Recommendation

Reference GMX's datastore and use the address that is stored their for applicable GMX addresses.

Resolution

Gamma Team: We use TransparentProxy so we can upgrade it later. We will take account for that later.

M-02 | Missing Feature Execution Validation

Category	Severity	Location	Status
DoS	Medium	GMXUtils.sol: 366	Resolved

Description

Orders can become stuck halting protocol functionality when order execution is disabled. When an order is created GMX will check that the create order feature is enabled.If it is not, it will revert.

However, during order creation there is no check that the order execution feature will be disabled.

This means that when order execution is disabled Gamma orders will still successfully be created but will not execute due to the feature validation in `_executeOrder` until the feature is enabled again. Halting protocol functionality for a period of time.

Recommendation

Before sending an order to GMX check that the execution feature is enabled.

Resolution

Gamma Team: The issue was resolved in commit 52e272d.

M-03 | Deprecated latestAnswer Used In _check Function

Category	Severity	Location	Status
Logical Error	Medium	KeeperProxy.sol: 128	Resolved

Description

The `_check function` in the `KeeperProxy` contract checks the price difference between the given price and the `Chainlink` price.

The issue is that the `Chainlink` price is fetched using the `latestAnswer` function, which is deprecated and should no longer be used reference.

Recommendation

The `_check function` should instead use the `latestRoundData` function for price retrieval, as it allows for additional validations to ensure that the price data is current and reliable.

Refer to the Chainlink documentation for implementation details.

Resolution

Gamma Team: The issue was resolved in commit c863800.

M-04 | Chainlink Price Lacks Price/Sequencer Validation

Category	Severity	Location	Status
Validation	Medium	KeeperProxy.sol: 128	Resolved

Description

The `_check` function in the `KeeperProxy` contract retrieves the `Chainlink` price; however, the price is fetched and used without performing any validation on the price or the sequencer status.

`Chainlink` recommends following certain security practices, such as checking for stale or invalid prices.

Additionally, when using `Chainlink` with L2 chains like Arbitrum, it is crucial to check whether the L2 Sequencer is down.

Recommendation

Add validation checks to ensure the price retrieved by the `_check` function is not stale or invalid Ref.

Also, implement sequencer feed checks to confirm the L2 Sequencer is operational before using the price data.

Follow the `Chainlink` example for these checks as outlined in their documentation.

Resolution

Gamma Team: The issue was resolved in commit c863800.

M-05 | Excess executionFee Not Refunded

Category	Severity	Location	Status
Logical Error	Medium	PerpetualVault.sol: 343	Resolved

Description

In `withdraw()`, `_payExecutionFee()` is called, which validates the `msg.value` is large enough to cover GMX's execution cost.

However, if a user goes to withdraw when there are no open positions, there will be no calls made to GMX.

Additionally, when calls to GMX are made, any excess execution fee refund by GMX is not returned to the user. This can occur when a user calls `deposit()` or `withdraw()`.

Recommendation

Move the `_payExecutionFee` call into the `curPositionKey != bytes32(0)` case for withdrawing.

Additionally, implement functionality to refund excess GMX execution fees to the creator of the deposit or withdrawal.

Resolution

Gamma Team: The issue was resolved in commit 1087920.

M-06 | DoS When Orders Are Stuck In GMX

Category	Severity	Location	Status
DoS	Medium	Global	Resolved

Description

GMX keepers will not always execute orders in a reasonable amount of time. For reasons outside of the creators control.

With no functionality to manually cancel these orders an order can be sitting for a prolonged period of time halting any other order operations.

Recommendation

Consider adding functionality for keepers to cancel orders after a certain amount of time.

Resolution

Gamma Team: The issue was resolved in commit 15f3967.

M-07 | Gas Estimation Uses Old Key

Category	Severity	Location	Status
Logical Error	Medium	GmxUtils.sol: 205	Resolved

Description

In the `getExecutionGasLimit` function the `ESTIMATED_GAS_FEE_BASE_AMOUNT` key is used to estimate the gas.

However with the upgrade of GMX V2.1 the `ESTIMATED_GAS_FEE_BASE_AMOUNT_V2_1` key is now used to validate the `executionFee` base amount upon creating an order.

Recommendation

Use the updated `ESTIMATED_GAS_FEE_BASE_AMOUNT_V2_1` key in the `getExecutionGasLimit` function.

Resolution

Gamma Team: The issue was resolved in commit b4022d5.

M-08 | Liquidations Don't Account For longTokens

Category	Severity	Location	Status
Logical Error	Medium	GMXUtils: 242	Resolved

Description

It is possible for a liquidation to send a long token from GMX. For instance, a liquidation can occur when the position was in profit, but fees caused a liquidation.

Alternatively, if GMX failed to swap into the collateral token, you can receive the long token. If this were to occur, the token would be stuck within `GMXUtils.sol`, and unrescuable.

Recommendation

Check if the long token was sent with a liquidation, and if so transfer it to `PerpVault.sol`. Additionally, consider adding an admin privileged function to rescue unexpected token transfers.

Resolution

Gamma Team: Liquidation issue; will handle liquidation risk via offchain logic

M-09 | Execution Fee Required For Swaps

Category	Severity	Location	Status
Logical Error	Medium	PerpetualVault.sol: 207, 231	Partially Resolved

Description

In the deposit and withdraw functions the `_payExecutionFee` function is called regardless of if the action will require a GMX order.

For example, if the vault is 1x long and a paraswap swap will be used to execute the swap, the execution fee for a GMX swap is still collected.

This unnecessarily charges the user for an action that will not occur and offers them no refund in the event that a normal swap is used.

Recommendation

Consider refunding the `executionFee` to the user if a GMX action is not performed on a swap.

Resolution

Gamma Team: The issue was resolved in commit 8a10486.

Guardian Team: If the intention is to only refund the execution fee when it is unused then consider refunding the full execution fee when there is sufficient funds.

M-10 | Keeper Actions Can Be Stalled By Deposits

Category	Severity	Location	Status
DoS	Medium	PerpetualVault.sol: 256	Resolved

Description

The protocol only allows one action to be processed at a time and depending on the action and circumstances each execution can take up to minutes to be finalized.

As a result a malicious actor could intentionally create many small positions when the vault has no open position, and then when the vault has an open GMX position start triggering withdrawals from each account to DoS the protocol for an extended period.

If each action takes 1 minute to process and the `minimumDeposit` amount is $10 then an attacker can DoS the protocol for one hour with $600 + gas. The attacker would recoup the majority of this upfront cost after they withdraw all shares.

This cost may be profitable if the actor were a GM holder wanting to keep Gamma in a poor position for longer to reap PnL gains.

Additionally a short-seller may be able to profit from such a DoS, spreading word of the trapped funds.

Recommendation

Consider batching deposits and withdrawals to GMX to avoid DoS's triggered by small individual accounts. Otherwise assign the minimum deposit value such that this attack is unprofitable.

Resolution

Gamma Team: Will set `minDeposit` at $1,000 upon initialization.

M-11 | setPerpVault Lacks Validation

Category	Severity	Location	Status
Access Control	Medium	GMXUtils: 332	Acknowledged

Description

`setPerpVault()` will initialize `perpVault`, and then will revert anytime it is called again. If a user monitors deployment and calls `setPerpVault()`, they will be able to set `perpVault`.

This will give a user control over execution flow for numerous function calls, if not dealt with. You could redeploy the contracts again, but you would be at risk of the same attack repeating itself.

Recommendation

Make `setPerpVault()` an access controlled function.

Resolution

Gamma Team: Acknowledged. There's no reason that the attacker does that action.

M-12 | Oracle Price Counts Ignored In Gas Estimation

Category	Severity	Location	Status
Logical Error	Medium	GMXUtils.sol: 374, 456	Resolved

Description

In the GMXUtils contract, when creating orders for GMX the Oracle price count is not accounted for when determining the size of the executionFee.

However in GMX V2.1 an additional amount of executionFee is necessary to account for the number of oracle prices to execute the order.

This logic is seen here:
https://github.com/gmx-io/gmx-synthetics/blob/1938e365dc009342aa288aa6b42fc1fd3cd9e45d/contracts/gas/GasUtils.sol#L212

Recommendation

Include the oracle price count estimated costs in the execution fee sent to GMX.

Resolution

Gamma Team: The issue was resolved in commit b4022d5.

L-01 | Protocol Fee Does Not Count Secondary Token

Category	Severity	Location	Status
Logical Error	Low	PerpetualVault.sol: 982	Resolved

Description

When the outputted amount is in both the long and the short underlying token, the `amount` variable will be less than expected.

Causing the governance fee to be less than expected or potentially no fee when a position was actually in profit.

Recommendation

Consider accounting for the long token when determining how much profit a user is in for governance fees.

Or Document to members of the governance team that some profitable positions may generate less yield than expected.

Resolution

Gamma Team: The issue was resolved in commit f2008b6.

L-02 | Users Always Charged Negative Impact Fees

Category	Severity	Location	Status
Documentation	Low	PerpetualVault.sol: 385	Resolved

Description

When computing the `feeAmount` which a user pays the `forPositiveImpact` is `hardcoded` as false, therefore even if a user's order positively impacted the GM market skew and received a lower fee rate the user has to pay the higher rate when Gamma accounts for their orders fee.

Recommendation

This behavior is likely more trouble to fix than it is worth, but be aware of this disagreement between the fees charged by the two systems and clearly document this for users.

Resolution

Gamma Team: This behavior is likely more trouble to fix than it's worth, so we will let users know about that. Always pay fee for negative price impact case.

L-03 | Redundant Boolean Expressions

Category	Severity	Location	Status
Optimization	Low	Global	Resolved

Description

Throughout the contracts the boolean comparison `== true` is used to check if a boolean value is true. However the boolean value itself can be used to save gas.

Recommendation

Remove all instances of `== true`.

Resolution

Gamma Team: The issue was resolved in commit a3f8827.

L-04 | Run Can Be Stalled Via Acceptable Price

Category	Severity	Location	Status
Logical Error	Low	GMXUtils.sol: 400	Resolved

Description

A strict acceptable price can lead to a `run` failing. It would fail because of the position that is being closed is large enough that the price impact would exceed the threshold. This can happen naturally as the positions get larger and larger. This can also happen as an attack.

Since a user would be able to know roughly when a `run` is going to occur they can either open a position right before Gamma which would increase the price impact gamma will face. Or the attacker could withdraw from GMX, with less capital in the market newly created positions will face greater price impact.

By preventing the run from executing the rest of the protocol will not be able to be used for a period of time. And more importantly the protocol will be forced to hold a position that does not align with their strategy since they won't be able to close or change it.

Recommendation

On `runs` consider passing in an `acceptablePrice`. This flexibility would allow position changes despite market conditions.

Also consider allowing the acceptable price to be changed when the `afterOrderCancellation` is called. Preventing situations where the order repeatedly fails.

Resolution

Gamma Team: The issue was resolved in commit 5f387ce.

L-05 | cancelDeposit Doesn’t Update Mapping

Category	Severity	Location	Status
Logical Error	Low	PerpetualVault.sol: 343	Resolved

Description

The `cancelDeposit` function allows the protocol to cancel a deposit request, but it currently doesn’t fully remove the deposit as it forgets to remove the entry from the `userDeposits` mapping.

As a result, the `getUserDeposits` function, which retrieves all deposit IDs for a user, will still include the cancelled deposit.

Recommendation

Ensure the cancelled deposit is removed from the `userDeposits` mapping, similar to how it is handled in the `_burn` function.

Resolution

Gamma Team: The issue was resolved in commit 5c40cff.

L-06 | cancelDeposit Function Should Use safeTransfer

Category	Severity	Location	Status
Logical Error	Low	PerpetualVault.sol: 343	Resolved

Description

The `cancelDeposit` function allows the protocol to cancel a deposit request, and the user's deposited funds are then sent back to the user.

However, the funds are currently transferred using the transfer method instead of the `safeTransfer` method, which is used elsewhere in the code.

This could be problematic, as some tokens may not revert on failure and instead return false, leading to undetected transfer failures.

Recommendation

Replace the transfer method with `safeTransfer` in the `cancelDeposit` function to ensure that token transfers are properly handled and any potential failures are detected.

Resolution

Gamma Team: The issue was resolved in commit 1087920.

L-07 | Max Deposit Cap Can Be Exceeded

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

It is possible to exceed the deposit cap by donating USDC to the vault after a deposit, these funds would essentially be donated to the vault, but if there were a single large holder they would be donating to themselves and surpassing the deposit cap.

Recommendation

Be aware of this possibility and ensure vaults are made up of a diverse number of depositors.

Resolution

Gamma Team: The issue was resolved in commit b4022d5.

L-08 | Position Accounting Mismatch

Category	Severity	Location	Status
Logical Error	Low	PerpetualVault: 401	Acknowledged

Description

If a user, who holds the entirety of the shares, withdraws while a position is open, it will lead to the GMX position being closed.

This will cause `curPositionKey` to be set to `bytes32(0)`, but `positionIsClosed` will not be updated in the withdrawal flow.

This will lead to unexpected issues when calling `run()`, as the current state will not match the expected state.

Recommendation

Check if the position has been fully closed in the withdrawal flow. If that is the case, set `positionIsClosed` to `true` so there are no discrepancies between `curPositionKey` and `positionIsClosed`.

Resolution

Gamma Team: Acknowledged.

L-09 | Settlement Can Unexpectedly Hit Withdrawal Case

Category	Severity	Location	Status
Logical Error	Low	PerpetualVault.sol: 373	Resolved

Description

If the current action is a `settle` then in the `afterOrderExecution` the condition to be met is based on `sizeDelta` and output amount.

This relies on strict equality and may not be future-proof for any GMX changes to how `outputAmount` is calculated.

If this case occurred and the `settle` condition was not met, the execution flow will enter the withdrawal case directly.

Consequently, the user will get the `settle` amount, instead of their withdrawal amount. Additionally the shares from the withdrawal will be burned but the position will be relatively unchanged.

Recommendation

Add a `SETTLE` to the Flow enum and use that in the `afterOrderExecution` callback.

Resolution

Gamma Team: The issue was resolved in commit 565919f.

L-10 | 1x Long State Siphoning collateralToken

Category	Severity	Location	Status
Gaming	Low	Global	Resolved

Description

In the PerpetualVault when the vault is in a 1x long state deposits are only valued based upon the proportion of index tokens they contribute to the vault.

However upon withdrawals the user receives their shares proportional value of the collateral tokens in the vault.

As a result if there are any leftover collateral tokens in the vault from unswapped funding fees or for any other reason, these collateral tokens can be siphoned by an arbitraguer.

Recommendation

Consider including the collateral token balance value when computing the shares to issue for a deposit when the vault is in a 1x long state.

Resolution

Gamma Team: The issue was resolved in commit f2008b6.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

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

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits